République Algérienne Démocratique et Populaire
Université Abou Bakr Belkaid– Tlemcen
Faculté des Sciences
Département d'Informatique

Mémoire de fin d'études

pour l'obtention du diplôme de Master en Informatique

Option: *Réseaux et Systèmes Distribués (RSD)*

Thème

Etude des techniques d'économie d'énergie dans les réseaux SDN

Réalisé par :

- **Tlemçani Fatima Zahra**
- **Moussaoui Kheira**

Présenté le 03 Juillet 2022 devant le jury composé de

- Meme. Abdeldjlil Hanane *(Président)*
- Mr. Bambrik Ilyas *(Encadreur)*
- Meme. Benmahdi Bouchra *(Examinateur)*

Année universitaire: 2021-2022

Dédicace

***A**vec une profonde gratitude et des mots sincères, que je dédie ce mémoire principalement à mon papa, qui a sacrifié pour mon succès et mon bonheur. J'espère qu'un jour, je pourrai rendre un peu pour lui de ce qu'il a fait pour moi.*

***À** ma maman. Je vous remercie pour tout le soutien et l'amour que vous me portez depuis mon enfance. Que ce modeste travail soit l'exaucement de vos vœux tant formulés, le fruit de vos innombrables sacrifices. Puisse Dieu, le Très Haut, vous accorder santé, bonheur et longue vie.*

***À** mon cher fiancé,*
***À** mes adorables sœurs et frères : Ahlem, Rimass et Rayane.*
***À** toute la famille de mon fiancé « Fellah ».*

***À** toutes mes copines préférées: Feriel, Fatima, Djihane, Douaa et Thouria.*

***A**insi qu'à toute ma famille : petits et grands, Surtout ma chère tante, « Malika », qui m'a soutenu dans les moments difficiles.*

***S**ans jamais oser oublier mon partenaire de mémoire « Fatima », je vous remercie beaucoup pour vos efforts et votre travail acharné pendant cette période difficile.*

Je vous aime.

Kheira

Dédicace

Ce projet est spécialement dédié à mon défunt heromy cher père «Mohammed» qui *malheureusement ne sera pas présent pour partager mon stress, ma joie et ma réussite le jour de ma remise des diplômes, j'espère juste qu'il est fier de moi.*

***À** ma merveilleuse mère, même si aucun dévouement ne peut s'exprimer ma profonde gratitude et mon appréciation pour tous les sacrifices qu'elle a consentis pour moi, merci pour votre confiance et votre soutien.*

***À** mes chers frères : Aymen et Riyad. À ma charmante sœur : Zineb.*

***À** toute la famille « arabe » ma famille maternelle qui m'a soutenu pendant les moments difficiles.*

***À** mes copines les plus adorables : Chaimaa, Ilhem, Kawter et Wafaa.*

***E**nfin, à mon plus gentil partenaire «kheira», merci d'avoir été assidu et patient tout au long de cette période difficile.*

Je vous aime.

***F**atima **Z**ahra*

Remerciements

Je remercie tout d'abord Dieu le tout puissant et miséricordieux, qui nous a donné la force et la patience d'accomplir ce modeste mémoire inclus.

Tout d'abord nous tenons à exprimer notre gratitude et appréciation à notre excellent encadreur, le docteur Ilyas Bembrik, pour avoir accepté si généreusement de nous encadrer et pour l'aide qu'il nous a apportée tout au long de notre travail, sa patience, sa disponibilité et surtout ses judicieux conseils, qui ont contribué à alimenter notre réflexion.

Nous adressons aussi nos vifs remerciements aux honorables membres de Jury à qui nous ont fait l'honneur de juger ce travail. Les critiques et suggestions que vous apportez seront les bienvenues et contribueront à l'amélioration du produit final.

Nous tenons à saisir cette occasion et adresser nos profonds remerciements et reconnaissances aux responsables et aux enseignants de la faculté de sciences, département d'informatique.

Nos derniers mots vont à nos chers parents, toutes nos familles et nos amies du fond du cœur pour l'aide et l'amour qu'ils nous ont apportés.

Si ce document a pu voir le jour, c'est grâce aux actions conjuguées de plusieurs personnes à qui je dis sincèrement merci.

Table des matières

Liste des Figures

Liste des Tableaux

Abréviations

SDN	Software Defined Network
IoT	Internet of Things
NOS	Network Operating System
APIs	Application Programming Interface
ONF	Open Network Foundation
TCAM	Ternary Content Addressable Memory
RIB	Routing Information Base
SBI	Southbound Interface
NBI	Northbound Interface
VM	Virtual Machine
TLS	Transport Layer Security
SSL	Secure Socket Layer
NIB	Network Information Base
CARPO	Correlation-Aware Power Optimization
NFM	Network Flow Management
FSA	Flow Scheduling Algorithm
NC	Network Configuration
DPC	Devices Power Control
TCAM	Ternary Content Addressable Memory
DCN	Data Center Network
FTRS	Flow Table Reduction Schem
LAN	Local Area Network
PS	Packet Size
IDT	Inter Departure Time
TC	Traffic Control
ONOS	Open Network Operating System

Introduction Générale

Au cours de la dernière décennie, l'économie d'énergie dans les réseaux informatiques est devenue un axe de recherche populaire au sein de la communauté scientifique ainsi que l'industrie. Afin de limiter les coûts opérationnels, l'objectif consiste à satisfaire les besoins des utilisateurs tout en limitant le gaspillage d'énergie. Évidemment, la tâche s'annonce complexe car les services utilisateur (VoIP, vidéoconférence et le streaming, etc) sont variés en termes de besoins en terme de QoS (Quality of Service) et l'économie d'énergie est faisable avec des techniques inverses par rapport à la prévision de la QoS. En effet, pour livrer des flux multimédias, il est très important que la politique d'économie d'énergie prenne en charge les besoins en termes de bande passante, de délai et du taux de pertes.

En outre, la plupart des équipements réseaux et les protocoles de communication sont inconscients de l'énergie qu'ils dissipent. Dans le réseau informatique classique, TCP/IP (Transmission Control Protocol/Internet Protocol) est une suite de protocoles de communication exploités pour coupler des dispositifs de réseau sur internet. Traditionnellement, les agents configurent les équipements composant l'infrastructure à travers la ligne de commande. Par contre, cette alternative peut demeurer limitée aux fonctionnalités préinstallées. En outre, les grands réseaux peuvent comporter des milliers de composants. Au cas où une nouvelle politique doit être appliquée, la reconfiguration de tous les équipements est nécessaire.

Dans cette optique, afin de combler les limites et difficultés issues de la prise en compte de la consommation d'énergie dans les réseaux traditionnels, l'objectif principal de SDN (Software Defined Networks) et OpenFlow est de détacher l'équipement de la couche logicielle de contrôle permettant aux opérateurs réseau de réduire les frais opérationnels. Cela signifie que la gestion du réseau et la manipulation des trafics de données est effectuée de manière centralisée. Par conséquent, l'implémentation des politiques ainsi que la reconfiguration est plus facile. En raison de ses avantages, il est intéressant d'étudier la manière dont la technologie SDN peut être utilisée, pour la mise au point d'un mécanisme conscient de la consommation énergétique.

Problématique

Le réseau informatique est devenu indispensable dans l'industrie et a pour fonctionnalité principale de fournir des services aux utilisateurs que ce soit des entreprises ou des particuliers. De nos jours, ce dernier représente une plateforme pour l'échange de données et de partage des ressources. Ces fonctionnalités sont cruciales, à tel point qu'on aurait aujourd'hui beaucoup de peine à imaginer notre quotidien et le monde du travail sans l'existence des réseaux.

En plus de la connexion physique fournie par l'infrastructure, une connexion logique entre les applications est nécessaire afin de transporter les messages échangés. Cette dernière est produite par des protocoles réseau spécifiques comme par exemple le protocole TCP qui se place au-dessus d'IP. Les deux sont tellement associés qu'on parle communément de TCP/IP [1,2].

Dans un réseau IP, la mise en relation entre un client et un serveur s'appuie sur des équipements de routage et de commutation [3]. Le protocole TCP/IP spécifie la manière dont les données sont transmises entre deux machines en fournissant des communications de bout en bout en spécifiant comment les messages doivent être divisés en paquets, adressés, transmis, routés et reçus par la destination [4].

Avant d'être déployés sur le réseau, les routeurs doivent être configurés. La configuration permet au minimum de définir les adresses IP des interfaces physiques et virtuelles du routeur et d'informer le routeur du protocole de routage à appliquer pour construire la table de routage [3].

Les protocoles de routage permettent à chaque routeur de récupérer des informations topologiques des routeurs adjacents afin de structurer localement des informations de routage RIB (Routing Information Base). Ainsi, les informations de routage sont actualisées pour prendre en compte l'état de chaque nœud (saturé, hors ligne, en panne…) de manière dynamique [3]. Par la suite, la table d'acheminement est exploitée par le routeur pour désigner l'interface sur laquelle envoyer le paquet selon l'adresse destination et la classe de service.

L'adaptation des réseaux TCP/IP traditionnels aux besoins des applications modernes est devenue compliquée aujourd'hui, notamment avec le rythme

effréné de l'évolution technologique. Sans parler des pertes de temps, d'efforts et d'argent. Cela est dû à plusieurs désavantages [5].

- L'infrastructure physique nécessite des configurations manuelles. Ceci ne permet tout simplement pas de suivre les nouvelles tendances des applications modernes.
- Le fonctionnement décentralisé du plan de contrôle empêche l'administrateur réseau de contrôler le flux de trafic.
- L'accès à la couche de données doit se faire directement sur le matériel.
- Coût de maintenance et d'administration pour l'ensemble du réseau est très élevé.
- Le propriétaire du réseau est limité aux protocoles de réseau appartenant aux fabricants de matériel.

L'objectif de ce travail de fin d'études, est d'étudier ce nouveau paradigme crucial pour les futurs réseaux. En outre, plusieurs idées et travaux ont été proposés auparavant, notamment la programmation du réseau et la séparation des plans de contrôle et de données. En particulier, la mise en œuvre d'un réseau programmable SDN permet une ouverture aux futures applications de mise en réseau commerciales. En conséquence, il est devenu possible de concevoir un réseau plus intelligent et flexible.

Dans ce travail, nous nous étalerons sur les bases du paradigme SDN et étudierons son architecture en détail. Ensuite, nous présenterons les différentes techniques d'économie d'énergie dans les réseaux SDN. La dernière partie de mémoire est consacrée à une étude expérimentale illustrant l'économie d'énergie implémentée sur SDN.

Plan du mémoire :

Ce travail est réparti sur les chapitres suivants:

- **Chapitre I:** Introduction aux réseaux SDN.
- **Chapitre II:** Etat de l'art sur l'économie d'énergie dans les réseaux SDN.
- **Chapitre III:** Expérimentation avec l'algorithme GreenSDN et la topologie Fat-Tree.

Chapitre I : Introduction aux réseaux SDN

I.1. Introduction

L'idée principale du SDN est de rendre le réseau programmable. C'est-à-dire de permettre aux applications d'interagir directement avec les réseaux, avec une coopération à double sens : l'application informe le réseau du comportement désiré, et inversement le réseau informe les applications de ses capacités. Le tout permettant d'établir un service avec des conditions définies a priori.

Dans ce chapitre, on va définir le fonctionnement typique d'un réseau classique afin d'illustrer ses limites et inconvénients. Le reste du chapitre est consacré à la présentation de SDN et ses avantages et défis. Puis nous traiterons les différents composants d'un réseau SDN.

I.2. Réseaux SDN

Depuis plusieurs années, il est devenu difficile d'innover ou d'apporter des changements au réseau TCP/IP. Avec les nouvelles technologies de l'information comme le Big Data qui nécessite un traitement distribué, le Cloud Computing, ou encore l'internet des objets (IoT), l'adaptation de l'architecture réseau classique aux nouveaux besoins constitue un grand défi, tant pour les opérateurs que pour les administrateurs réseaux [6,7].

La mise à jour des réseaux classiques est réalisée en configurant manuellement chaque équipement ce qui peut produire plusieurs erreurs et incohérences. De plus, ces configurations prennent beaucoup de temps ce qui rend la tâche difficile face à l'expansion rapide du monde des technologies d'information.

C'est dans ce contexte qu'est apparu le paradigme des réseaux définis par logiciels (SDN) qui permet principalement de s'adapter à la nature dynamique des applications susmentionnées [6]. Afin de combler les problèmes de reconfiguration rencontrés dans les réseaux TCP/IP, SDN permet de simplifier la gestion et l'innovation dans le réseau, en séparant la logique de contrôle des équipements d'interconnexions et en promouvant la centralisation du contrôle et la capacité de programmer le réseau [7].

La définition académique, consistait à voir le SDN comme une architecture qui découple les fonctions de contrôle et de transfert des données du réseau afin d'avoir une infrastructure physique complètement exempte de tout service réseau [8]. Dans ce modèle, les équipements réseau se contentent d'implémenter des règles injectées par les applications. Ces règles permettent d'identifier des flux de données ainsi que comment les traiter. Avec un plan de contrôle placé

dans une instance centrale, appelée « contrôleur », l'état global du réseau est complètement visible et accessible. Par conséquence, le contrôleur injecte directement les règles de traitement des données sur chaque équipement pour implémenter une politique souhaitée [8,7].

Plus concrètement, on peut dire qu'une architecture réseau suit le paradigme SDN si, et seulement si, elle vérifie les points suivants [7] :

- ✓ **Séparation du plan de contrôle (Control Plane) et du plan de données (Data Plane) :**

Cela signifie que ces deux plans sont complètement indépendants dans le réseau, tout en maintenant une communication étroite entre eux.

- ❖ **Le plan de données :** Son rôle principal est d'assumer la responsabilité de la mise en mémoire tampon des paquets, de la planification des paquets, de la modification des en-têtes et de la transmission des paquets au prochain saut en fonction de caractéristiques telles que l'adresse MAC, l'adresse IP et l'ID VLAN. Ce plan est installé dans les périphériques réseau. Si un paquet reçu par un Switch SDN correspond à un flux existant, celui-ci peut être soumis à certaines modifications des champs d'en-tête selon les règles installées par le contrôleur. Certains paquets ne peuvent pas être traités de cette manière par contre. Si aucune règle ne correspond à un paquet reçu, ce dernier est communiqué au contrôleur afin d'obtenir la règle appropriée. En outre, certains paquets appartiennent au protocole Openflow et doivent être traités par le Switch afin de mettre à jour sa table de flux [9].
- ❖ **Le plan de contrôle :** Détermine comment les transferts des flux doivent être effectués. Son rôle principal est de mettre à jour les informations dans la table de flux afin que le plan de données puisse livrer les paquets. Ce plan est la pièce maîtresse de cette architecture.
  En pratique, le contrôleur SDN est une plate-forme logicielle de gestion qui s'exécute au-dessus de l'architecture client/serveur dans TCP/IP et facilite l'approvisionnement et la programmation des dispositifs de transfert sur la base d'une visibilité globale, abstraite et centralisée du réseau [10]. La Figure I-1 illustre la différence entre les réseaux traditionnels et SDN.

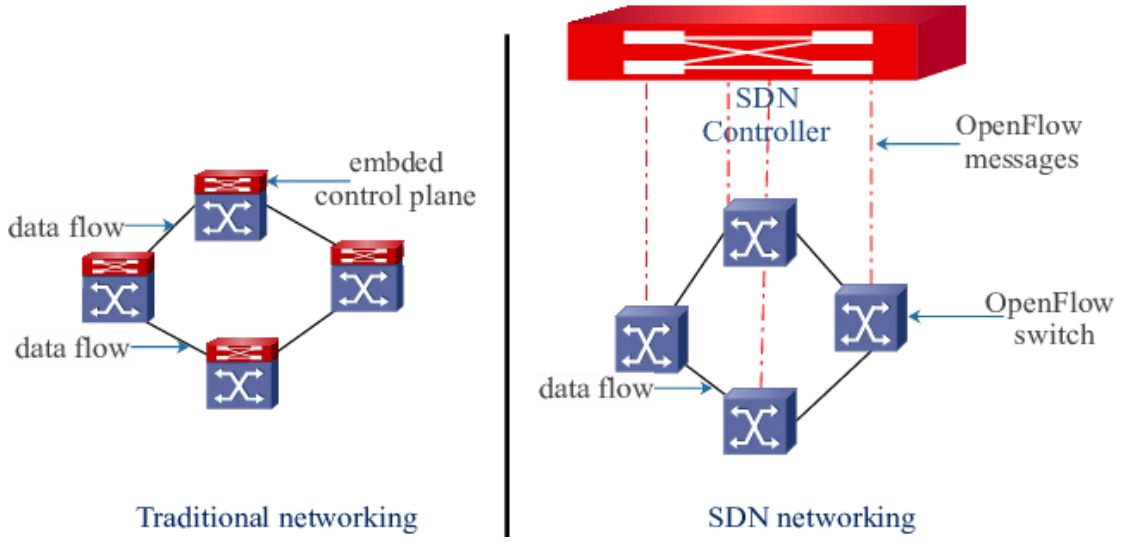

Figure I- 1 : Comparaison entre réseaux traditionnels et SDN [11]

Dans les réseaux traditionnels, le plan de contrôle et le plan de données réside à l'intérieur du périphérique réseau. Chaque périphérique a son plan de contrôle et prend des décisions conformément à la politique/protocole configuré. Une fois que les politiques ont été configurées, il est très difficile de modifier le comportement du réseau en réponse à l'évolution des demandes des services utilisateur. La seule façon est de reconfigurer tous les périphériques [12].

En revanche, une architecture basée sur SDN découple le plan de contrôle du plan de données où le contrôleur joue le rôle du système d'exploitation réseau [11]. L'architecture logiquement centralisée offre plusieurs avantages, tels qu'un meilleur support pour l'ingénierie du trafic et économie d'énergie, l'amélioration de la sécurité, une meilleure surveillance du système qui réagit automatiquement aux changements de l'état du réseau. En outre, il permet au développeur de mettre en œuvre des services et des applications réseau sophistiqués [10].

- ✓ **Basé sur le flux :**

Contrairement au réseau conventionnel où le commutateur/routeur prend des décisions de transfert basées sur la destination, le contrôleur SDN identifie un flux selon plusieurs critères. Un flux est considéré comme l'ensemble de paquets ayant des politiques de service identiques circulant de la source à la destination [10].

I.3. Architecture SDN

Un réseau classique est composé généralement des équipements d'interconnexions tels que des switches et des routeurs. Ces équipements incorporent à la fois le plan de donnée et le plan de contrôle. Par conséquence, il est difficile de développer de nouveaux services en raison du fort couplage qui existe entre le plan de contrôle et le plan de données.

Afin d'ouvrir les équipements réseaux aux innovations, l'architecture SDN a vu le jour [7]. Selon l'ONF, elle est composée de trois couches principales et d'interfaces de communication (Figure I-2). Nous décrivons dans la suite ces couches ainsi que les interfaces de communication :

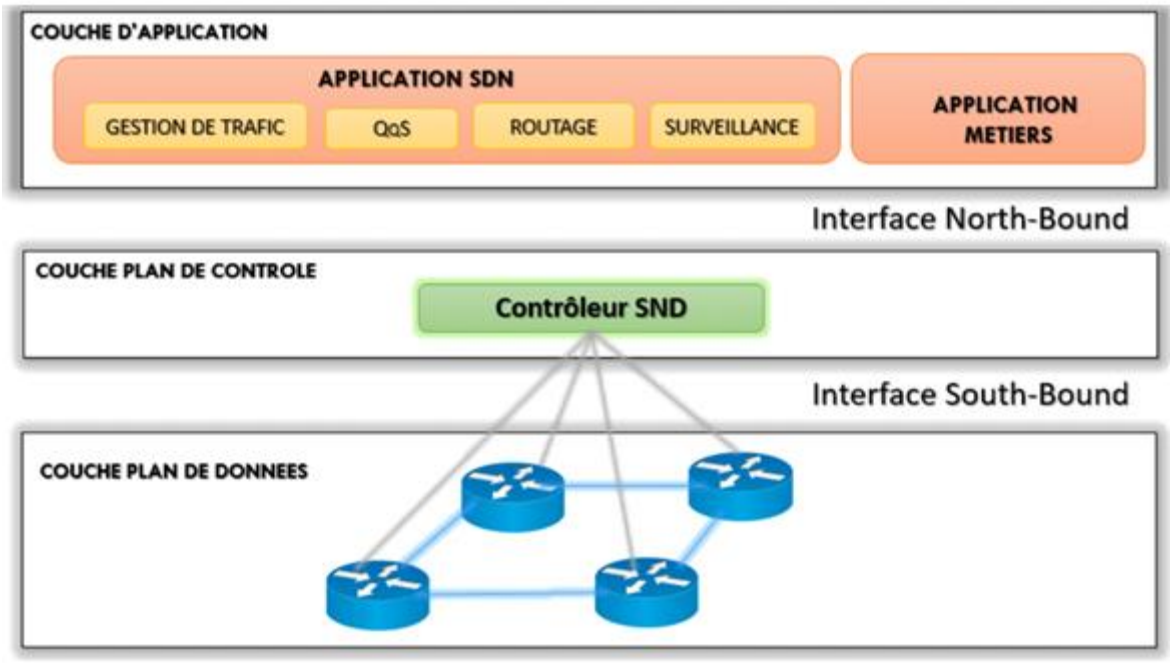

Figure I- 2 : L'architecture du réseau SDN [17]

- **La couche plane de donnée :** où se trouvent les périphériques qui contiennent les plans de données du réseau. Autrement dit, les switches SDN responsables de l'acheminement du trafic [16]. Son rôle principal est de transmettre les données, surveiller les informations locales et collecter les statistiques [13].
- **La couche de contrôle :** est une couche logicielle qui représente le contrôleur SDN. Son rôle est de gérer les équipements de l'infrastructure à travers une interface appelée « south-bound API » [7]. Cette partie est le cerveau du réseau et elle englobe la plupart des opérations de calcul [16].

- **La couche d'application** : représente les applications qui permettent de déployer de nouvelles fonctionnalités réseau, comme l'ingénierie de trafic, QoS, la sécurité, etc. Ces applications sont construites moyennant une interface de programmation appelée « north-bound API » [7].

I.4. Avantages du SDN

Les avantages du SDN sont nombreux pour les organisations. Ci-dessous une liste d'avantages de cette architecture.

- ✓ **Gestion efficace du réseau :** SDN permet aux administrateurs de paramétrer le réseau à partir d'une manière centralisée. Un contrôle de réseau facile et fiable est réalisable en paramétrant le réseau en fonction de la charge de travail [15].
  Gérer le routage via SDN permet de gérer finement l'utilisation des liens et ainsi d'améliorer les performances en augmentant l'utilisation moyenne de chaque lien sans saturer totalement le lien car cela serait contre-productif [14].
- ✓ **Flexibilité :** SDN apporte régulièrement une grande flexibilité sur la gestion du réseau. Avec cette technologie il est devenu plus facile de rediriger le trafic, d'inspecter des flux particuliers, de tester de nouvelles stratégies ou de découvrir des flux inattendus [14].
- ✓ **Protection enrichie :** Dans un environnement virtualisé, le financement de la machine virtuelle est en fait une lutte extrêmement ardue. Pourtant, le SDN offre une surveillance délicate à travers les appareils [15].
  Puisque le contrôleur est responsable de l'ajout de règles dans les commutateurs, il n'y a aucun risque qu'un administrateur de réseau ait oublié un commutateur ou installé des règles incohérentes entre les dispositifs [14].
- ✓ **Simplification matérielle :** SDN a tendance à utiliser des technologies standard et de base pour contrôler les équipements du réseau, tandis que la puissance de calcul n'est requise qu'au niveau du contrôleur. Ainsi, les équipements de réseau deviendront des produits à bas prix offrant des interfaces standard. Ainsi, le réseau devient facilement évolutif [14].

I.5. Les interfaces SDN

Le SDN comporte trois APIs suivantes :

- L'interface en direction sud (South-bound API) : Nous illustrons ces interfaces dans Figure 3 avec deux contrôleurs distribués [18]. Le plan de contrôle et le plan de données communiquent via l'interface en direction sud. Cette interface rend les informations des dispositifs du plan de données accessibles au plan de contrôle et permet la transmission des instructions et règles dans le sens inverse [19].
- L'interface en direction nord (North-bound API) : Le plan de gestion utilise l'interface en direction nord pour communique avec le SDN [19].
- L'interface en direction est-ouest (East-West API) : L'interface orientée est-ouest permet aux contrôleurs de partager une vue commune du réseau et de coordonner l'application des politiques et des protocoles [18]. La Figure I-3 expose les trois types d'interfaces dans l'architecture SDN.

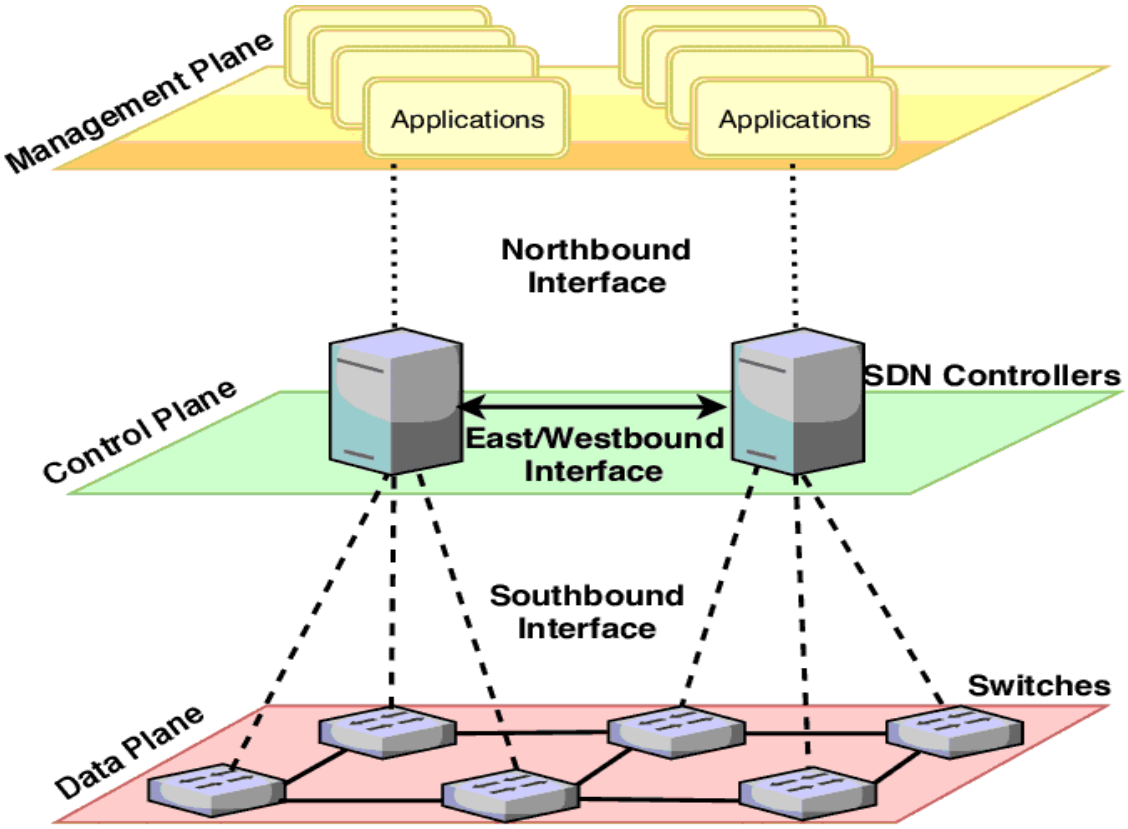

Figure I- 3 : interfaces de SDN [19]

I.5.1. Southbound API

Les interfaces sud est une partie importante du système SDN, qui fournit une interface entre les contrôleurs et les commutateurs pour mettre à jour les entrées de la table de flux [20]. Ceci est accomplie grâce à des protocoles comme Open

Flow, POF [21], OpFlex [22] et Open State [23], et les composants des protocoles supplémentaires pour la gestion des périphériques, physiques ou virtuels, anciens ou récents (tel que : SNMP, BGP et NetConf). En d'autres termes, les protocoles sud définissent les communications de contrôle qui se produisent entre le contrôleur et les dispositifs du plan de données [20]. Actuellement, OpenFlow est le standard adopté pour cette interface.

I.5.2. East/Westbound APIs

Les interfaces Est/Ouest sont des interfaces de communication qui permettent généralement la communication entre les contrôleurs dans une architecture multi-contrôleurs. Ces architectures sont très récentes et aucun standard de communication inter-contrôleur n'est actuellement disponible.

Actuellement, chaque contrôleur implémente sa propre API l'est/l'ouest. Les fonctionnalités de ces interfaces incluent l'importation/exportation de données entre les contrôleurs, les algorithmes pour les modèles de cohérence des données et les capacités de surveillance/notification (par exemple, vérifier si un contrôleur est opérationnel ou notifier une prise de contrôle sur un ensemble de dispositifs de transfert) [24].

Cette API nécessite des mécanismes avancés de distribution de données tels que le protocole AMQP (Advanced Message Queuing Protocol) [25] utilisé par CISCO [26], des techniques de composition de politiques concurrentes et cohérentes distribuées [27], des bases de données transactionnelles et des DHT [28], ou des algorithmes avancés pour cohérence forte et tolérance aux pannes [29,30] [24].

I.5.3. Northbound API

Les interfaces nord est autant cruciale que l'interface sud dans l'écosystème SDN [24]. L'API nord joue un rôle essentiel pour les développeurs d'applications et fournie une interface commune entre le contrôleur et le plan de gestion. Cette interface aide à fournir les informations des périphériques sous-jacents pour le développement d'applications, ce qui rend le contrôle SDN simple et dynamique. Contrairement à l'interface sud, l'interface nord a connue moins d'efforts de normalisation car il s'agit d'un écosystème logiciel. Ainsi une large gamme de NBI est offerte par les contrôleurs avec les langages de programmation modernes [19].

Ce niveau d'application comprend souvent des applications globales d'automatisation et de gestion des données, ainsi que des fonctionnalités réseau de base telles que le calcul du chemin de données, le routage et la sécurité [31].

Actuellement, les contrôleurs existants tels que Trema, Floodlight, Onix, NOX et OpenDaylight définissent leur propre direction nord API [32,33]. Les langages de programmation tels que Frenetic [34], Nettle [35], NetCore [36], Procera [37], Pyretic [38] et NetKAT [39] résume également les détails internes du contrôleur et le comportement du plan de données [24].

I.6. Le protocole OpenFlow dans l'architecture SDN

OpenFlow sert de lien entre le plan de contrôle et le plan de données. L'échange de messages se fait au cours d'une session TCP établie via le port 6653 du serveur au contrôleur. Bien que la motivation initiale d'OpenFlow soit de développer des réseaux virtuels et programmables, il a également été largement considéré comme un candidat prometteur pour le plan de contrôle et l'interface [40].

OpenFlow a été initié comme un projet à l'université de Stanford lorsqu'un groupe de chercheurs exploraient la manière de tester de nouveaux protocoles dans le réseau IP. L'objectif été de créer un réseau expérimental au-dessus du réseau de production mais sans arrêter le trafic du réseau de production lors des tests [41]. Bien que les formulations des concepts sous-jacents puissent être retracés au début des années 2000, la désignation SDN est récente, précédée de peu par les premiers travaux sur OpenFlow en 2008 [42]. La version 1.0.0 des spécifications du protocole, destinée à la production, était publiée début 2010, où il n'y a que 12 champs de correspondance fixes et une seule table de flux. La dernière version comporte plusieurs tables de flux, plus de 41 champs de correspondance et beaucoup de nouvelles fonctionnalités. Ainsi, la capacité et l'évolutivité du protocole ont été largement étendues [43].

Le pont avec l'industrie a rapidement été franchi, en 2011, avec la création de l'ONF par Deutsche Telekom, Facebook, Google, Microsoft, Verizon et Yahoo. L'ONF est l'organisme principal encourageant à l'adoption des pratiques SDN, et encadre l'évolution du protocole OpenFlow [44]. En 2012, Google a présenté son installation pionnière : une architecture SDN qui effectue désormais le routage de leur backbone utilisateur ainsi que de leur backbone interne, mise en place en deux ans [44].

Le tableau suivant présente les spécifications d'Openflow et discute les additions apportées par chaque version :

Version Openflow	**Additions apportées par la spécification**
	- Les paquets Ethernet et IP sont identifiés selon l'adresse

V 1.0	source et de destination. Port source ou destination UDP et TCP. - Champs Ethernet-type et Vlan pour la couche 2 - Champs protocole, DS et ECN pour la couche 3
V 1.1	- Ajout du pipeline, de la table de groupe et des métadonnées - Ajout des champs d'identification MPLS
V 1.2	- Modèle OXM (Openflow Extensible Match) qui apporte un support IPv6 et une structure de correspondance plus flexible - Un switch peut à présent être connecté à plusieurs contrôleurs en mode Master/Slave.
V 1.3	- Ajout de la table de mesure qui mesure le taux de paquets assignés à une entrée. - Amélioration du support des contrôleurs multiples
V 1.4	- Amélioration d'OXM et de la structure TLV ce qui apporte un gain de synchronisation.
V 1.5	- Notion des tables de sortie qui permet de traiter les paquets sur le port de sortie en même que sur les ports d'entrée.

Tableau I- 1 : L'évolution du protocole OpenFlow [16]

Dans un réseau SDN, Il existe au moins un contrôleur OpenFlow qui communique avec les commutateurs OpenFlow. Le protocole OpenFlow définit les formats des messages échangés entre le contrôleur et commutateurs compatible avec ce protocole. Le comportement OpenFlow spécifie comment l'appareil doit réagir dans diverses situations et comment il doit répondre aux commandes du contrôleur [45]. Un commutateur OpenFlow contient au moins une table de flux et utilise un canal sécurisé pour communiquer avec le contrôleur selon les spécifications du protocole OpenFlow [40]. La Figure I-4 illustre l'architecture d'un réseau OpenFlow.

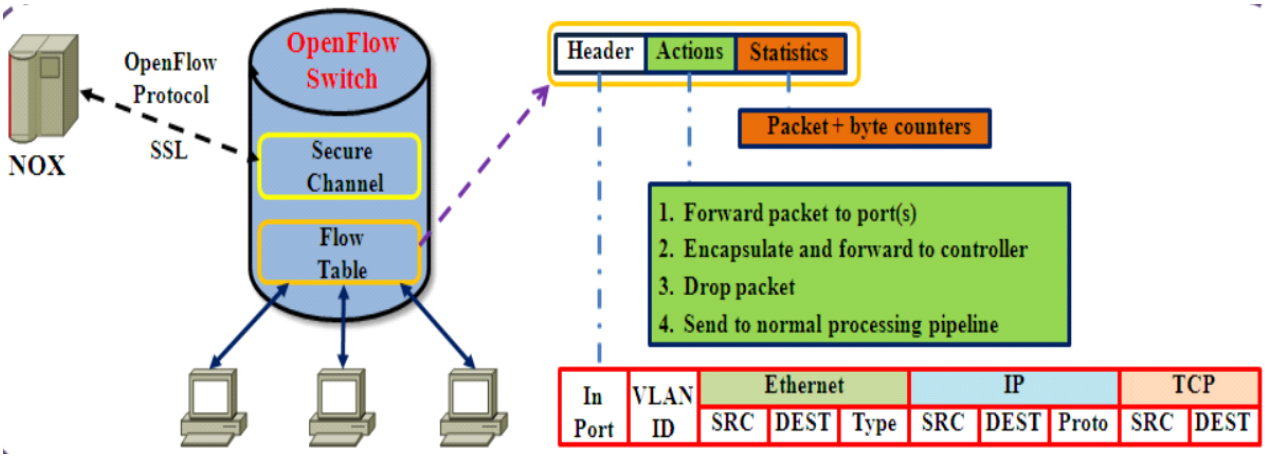

Figure I- 4 : Architecture d'un réseau compatible OpenFlow [40]

I.6.1. Structure d'un commutateur OpenFlow

Le terme commutateur est attribué à tout équipement de transmission qui compare l'entête des paquets aux tables de flux, que la comparaison se fasse à base d'adresses MAC (Couche 2), d'adresses IP (Couche 3) ou une combinaison de plusieurs champs [16].

Un commutateur SDN est composé d'une API pour la communication avec le contrôleur, d'une couche d'abstraction et d'une fonction de traitement des paquets. La couche d'abstraction comprend une ou plusieurs tables de flux. La logique de traitement des paquets comprend les mécanismes permettant de prendre des mesures en fonction des résultats de l'évaluation des paquets entrants et de trouver la correspondance la plus prioritaire [9]. La Figure I-5 expose l'anatomie logique d'un commutateur SDN.

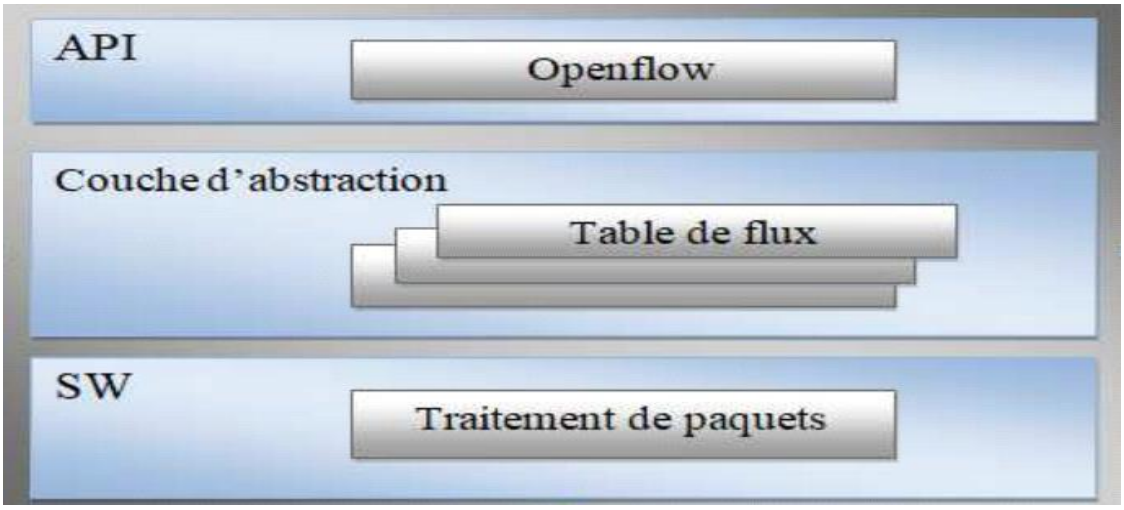

Figure I- 5 : Anatomie logique d'un commutateur SDN [17]

On distingue deux types de commutateur OpenFlow :

- Les commutateurs Pure : prennent en charge uniquement le fonctionnement OpenFlow. Dans ces commutateurs, tous les paquets sont traités par le pipeline OpenFlow et ne peuvent pas être traités autrement [47].
- Les commutateurs hybrides : prennent en charge à la fois le fonctionnement OpenFlow et le fonctionnement d'un commutateur ordinaire, c'est-à-dire la commutation Ethernet L2 traditionnelle, l'isolation VLAN et le routage L3. La plupart des commutateurs actuellement disponibles et commerciaux sont hybrides [46] [47].

I.6.2. Table de flux

Depuis la version 1.1 les commutateurs Openflow consistent en un pipeline de tables de flux implémentes dans la mémoire TCAM qui permettent de gérer le routage des paquets. Ce mécanisme consiste à assembler les actions de chaque table de flux pour les exécuter à la sortie du pipeline. La Figure I-6 ci-dessous décrit ce fonctionnement.

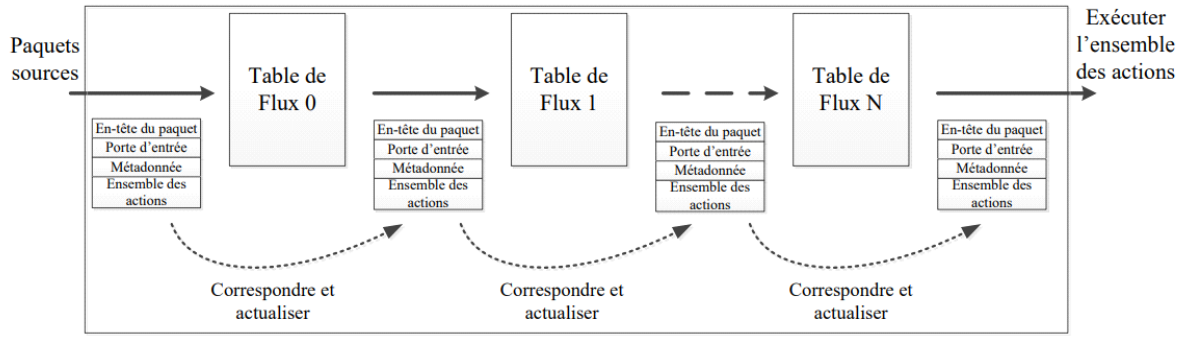

Figure I- 6 : Pipeline Openflow v1.1 [13]

Nous décrirons à présent les champs des entrées qui composent les tables de flux, globalement structurés avec trois sections.

Champs d'en-tête	Compteurs	**Instructions**

Tableau I- 2 : Structure d'une entrée de table de flux [13]

I.6.3.1. Champs d'entête

Utilisé lors d'identification des flux et contient les informations nécessaires pour déterminer les paquets au quels cette entrée sera appliquée, allant de la couche 1 à la couche 4 du modèle ISO. Pour permettre une transmission rapide des paquets avec OpenFlow, le commutateur utilise une mémoire de type TCAM qui permet une recherche efficace des masques. La figure suivante montre quelques-uns des champs selon les spécifications d'OpenFlow, comme Ethernet, IPv4. Depuis les récentes, d'autres possibilités d'identification ont été rajoutés comme les champs IPv6 et les étiquettes MPLS [13].

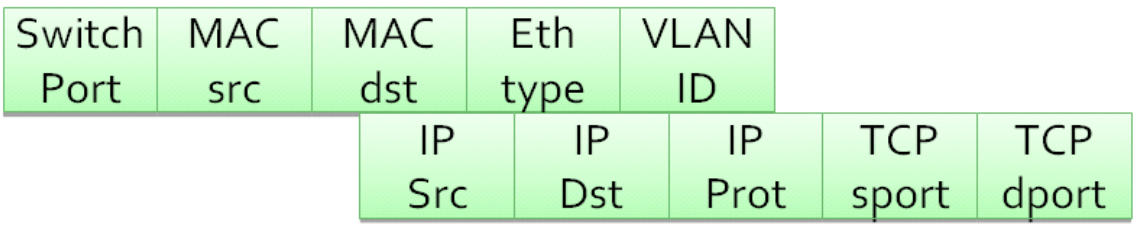

Figure I- 7 : Champs de correspondance Openflow v 1.0 [16]

I.6.3.2. Compteurs (Counters)

Servent essentiellement à garder des statistiques sur les flux, telles que le nombre de paquets et d'octets reçus, la durée des flux dans la mémoire temporaire et la gestion des entrées de table de flux, pour ensuite décider si une entrée de flux est active ou non. Pour chaque table, chaque flux, chaque port, des compteurs de statistiques sont maintenus [16].

I.6.3.3. Instructions (Actions)

À chaque entrée de la table de flux est associé un ensemble d'actions à exécuter sur les paquets, avant de les envoyer vers un port de sortie. Les actions doivent être exécutées dans le même ordre dans lequel elles sont présentées dans la table. Un paquet peut satisfaire les Champs d'en-tête de plusieurs entrées. Dans ce cas, les actions de l'entrée avec les valeurs de règles les plus spécifiques (minimum de valeurs) sont appliquées. Par contre, si une entrée ne contient aucune action, le paquet qui lui correspond est supprimé. Une action peut être l'une des suivantes [16] :

- Transférer le paquet à travers un ou plusieurs ports de sortie vers le nœud suivant dans le réseau.
- Encapsuler et envoyer le paquet vers un contrôleur à travers le canal sécurisé. Cette action est principalement utilisée lorsqu'un paquet d'un flux est reçu pour la première fois. Le contrôleur peut décider si une nouvelle entrée doit être ajoutée dans la table, ou si le paquet doit être supprimé.
- Modifier le champ d'entête d'un paquet.
- Supprimer tous les paquets appartenant à un flux est aussi une action qui peut être utilisée pour une raison de sécurité pour mettre fin à une attaque réseau par exemple.

Lorsqu'une correspondance est trouvée, le paquet est traité localement dans le commutateur, à moins qu'il soit explicitement envoyé au contrôleur. Les instructions d'une entrée de flux peuvent explicitement diriger le paquet vers une autre "Flow Table" dans un pipeline via l'instruction "Goto". Si l'entrée de flux ne dirige pas le paquet vers une autre table, le "pipeline processing" s'arrête à cette table. Dans ce cas le paquet subira les actions accumulées qui lui sont associées durant le pipeline [16] [49].

Instruction	Argument	Sémantique
Apply-Actions	Action(s)	Appliquer les actions immédiatement sans les ajouter à l'ensemble des actions.
Write-Actions	Action(s)	Configurer une action spécifique dans le champ des actions.
Clear-Actions	_	Supprimer les actions.
Write-Metadata	Métadonnées	Actualiser le champ de la Métadonnée
Goto-Table	ID de la table	Faire correspondre à la table suivante

Tableau I- 3 : Liste des instructions pour la version 1.1 d'OpenFlow [13]

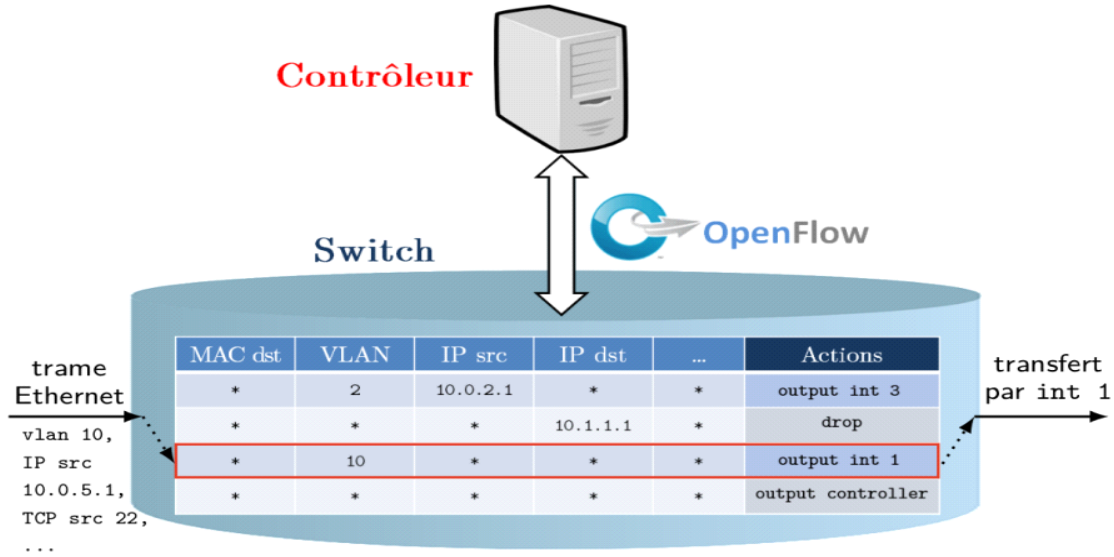

Figure I- 8 : Schématisation de la table de flux[44]

La figure ci-dessus illustre le traitement d'un paquet arrivant à un commutateur OpenFlow contenant une seule table de flux : parmi les différents champs d'en-têtes de la trame, l'appartenance au VLAN 10 fait correspondre cette trame à un

des flux du commutateur, dont l'action associée consiste en un transfert par l'interface 1 [44].

I.6.3. Messages Openflow

La communication entre les commutateurs SDN et le contrôleur s'effectue grâce au protocole OpenFlow qui définit la longueur, l'identificateur de transaction et le type des messages échangés entre ces deux entités [16]. Après la configuration des commutateurs avec l'adresse IP du contrôleur, ces derniers établissent une connexion sécurisée et assurée au contrôleur avec les protocoles cryptographiques SSL ou TLS qui fonctionne avec TCP, où le commutateur et le contrôleur sont mutuellement authentifiés par l'échange des certificats signés par les deux côtés de la clé privée avant que les données ne soient envoyées [48].

La logique du traitement des paquets consiste en un nombre de mécanismes qui se mettent en actions en fonction du résultat de l'évaluation de l'entête du paquet et la correspondance de priorité la plus haute. Si le paquet ne correspond à aucune entrée de flux, le comportement du commutateur vis-à-vis ce paquet dépendra de la configuration de la table. Le comportement par défaut est d'envoyer le paquet par un message PACKET_IN au contrôleur. Par la suite, ce dernier répondra par un PACKET_OUT donnant l'instruction à suivre, et éventuellement mettant à jour la table de flux pour introduire une nouvelle entrée. Une autre option consiste à supprimer le paquet. La table peut aussi spécifier que dans ce cas le paquet doit continuer son chemin dans le pipeline vers la table suivante [16] [49].

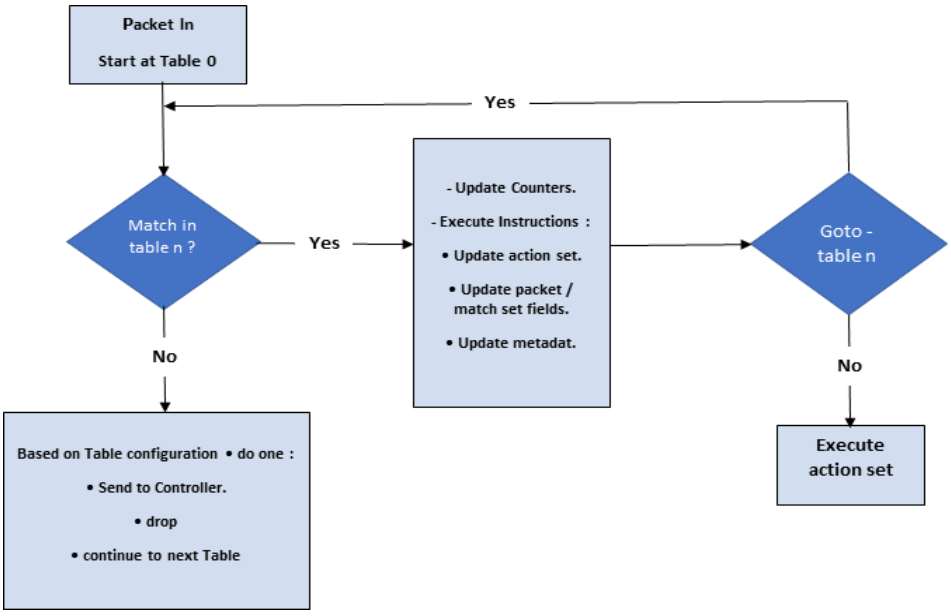

Figure I- 9: Traitement des paquets dans un réseau OpenFlow [49]

Le protocole OpenFlow définit trois catégories de messages, chacun ayant plusieurs sous-types :

- ✓ Contrôleur-à-commutateur.
- ✓ Symétrique.
- ✓ Asynchrone.

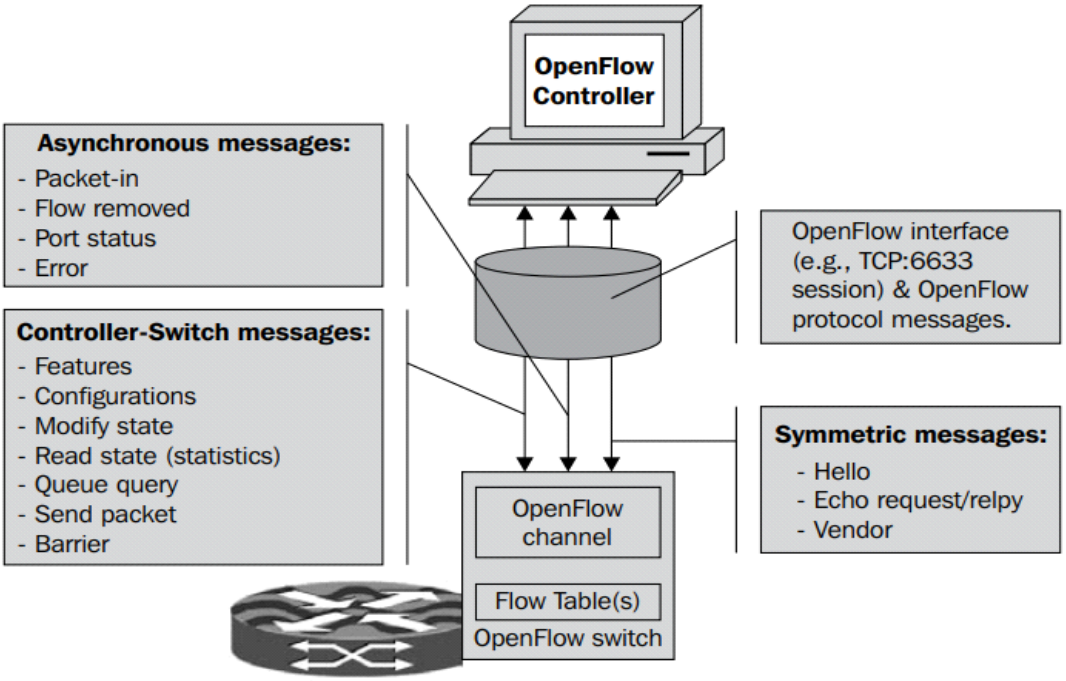

Figure I- 10 : Types de messages OpenFlow [46]

I.6.3.1. Les messages Contrôleur-à-commutateur

Les messages contrôleur/commutateur sont initiés par le contrôleur et peuvent nécessiter une réponse du commutateur, [47] et sont utilisés pour gérer ou inspecter directement l'état du commutateur [46]. Les messages appartenant à cette catégorie :

- ➢ **Features :** Le contrôleur peut demander l'identité et les capacités de base d'un commutateur en envoyant une demande de fonctionnalités. Le commutateur doit répondre par une liste de fonctionnalités qui spécifie l'identité et les capacités supportées par le commutateur. Ceci est généralement effectué lors de l'établissement du canal OpenFlow [47].

Les messages implémentés comprennent les OFPT_Features_Request et OFPT_Features_Reply qui sont requis pour l'initialisation du canal OpenFlow entre le commutateur et le contrôleur. Le message OFPT_Packet_In est utilisé par le commutateur pour informer le contrôleur d'un paquet entrant sans correspondance ou pour envoyer un paquet entrant au contrôleur s'il s'agit de l'action associée d'une correspondance [50].

- **Configuration :** Le contrôleur peut définir et interroger les paramètres de configuration dans le commutateur [47] avec les messages OFPT_SET_CONFIG et OFPT_GET_CONFIG_REPLY, respectivement [46]. Le commutateur ne répond qu'à une demande de configuration par un message OFPT_GET_CONFIG_REPLY [46,47]
- **Modify-State :** Les messages de modification d'état sont envoyés par le contrôleur pour gérer l'état des commutateurs. Leur objectif principal est d'ajouter, supprimer ou de modifier les entrées de flux/groupe dans les tables OpenFlow et de définir les propriétés du port du commutateur [47].

  Les modifications apportées à la table de flux depuis le contrôleur sont effectuées avec le message OFPT_FLOW_MOD. Avec ce type de messages, le contrôleur modifie le comportement des ports physiques [46].

- **Read-State :** Les messages Read-State sont utilisés par le contrôleur pour collecter diverses informations du commutateur, telles que la configuration actuelle, les statistiques et les capacités [47]. Le contrôleur peut demander l'état du commutateur à l'aide du message OFPT_STAT_REQUES, et le commutateur répond par un ou plusieurs messages OFPT_STATS_REPLY [46].
- **Packet-out :** Ceux-ci sont utilisés par le contrôleur pour envoyer des paquets depuis un port spécifié sur le commutateur et pour transférer les paquets reçus via des messages Packet-in. Les messages de sortie de paquet doivent contenir un paquet complet ou un ID de tampon faisant référence à un paquet stocké dans le commutateur [47]. Ils doivent contenir aussi une liste d'actions à appliquer, s'il n'y a pas d'action définie le paquet sera détruit.

- **Barrier :** Le message BARRIER_REQUEST est utilisé par le contrôleur pour s'assurer que tous les messages OpenFlow émis par le contrôleur et qui ont précédé cette requête ont été reçus et traités par le commutateur. La confirmation est retournée par le commutateur via la réponse BARRRIER_REPLY [48].
- **Role-Request :** Les messages de demande de rôle sont utilisés par le contrôleur pour définir le rôle de son canal OpenFlow. Ceci est surtout utile lorsque le commutateur se connecte à plusieurs contrôleurs [47].

I.6.3.2. Messages symétriques

Les messages symétriques peuvent être émis indifféremment par le contrôleur ou le commutateur sans avoir été sollicité par l'autre entité [51]. Il existe trois sous-types de messages symétriques dans le protocole OpenFlow :

- ✓ **Hello :** Les messages Hello sont échangés une fois que le canal sécurisé a été établi entre le contrôleur et le commutateur [43] [51].
- ✓ **Echo :** Les messages ECHO sont utilisés dans les deux sens pendant le fonctionnement du canal pour s'assurer de la vivacité d'une connexion contrôleur-commutateur et, afin de mesurer la latence la liaison. Chaque message ECHO_REQUEST doit être acquitté par un message ECHO_REPLY [43] [51].

  En cas où la connexion est rompue entre le commutateur / Contrôleur le commutateur commence à utiliser une Table de Flux alternative appelée Emergency Table et supprime les entrées installées par le Contrôleur avant la coupure ce fonctionnement est appelée Mode d'Urgence. Le commutateur continue à tenter de connecter au Contrôleur et s'il succède, le mode de fonctionnement normal restitué [52].

- ✓ **Vendor :** Ces messages fournissent un moyen standard pour les commutateurs OpenFlow d'offrir des fonctionnalités supplémentaires dans les futures versions [43].

I.6.3.3. Les messages asynchrones

Les messages asynchrones sont envoyés sans qu'un contrôleur ne les sollicite auprès d'un commutateur. Les commutateurs envoient des messages asynchrones aux contrôleurs pour indiquer l'arrivée d'un paquet, un changement

d'état du commutateur ou une erreur. Les quatre principaux types de messages asynchrones sont décrits ci-dessous [47] :

- ✓ **Packet-in :** Le message PACKET_IN est utilisé par le commutateur pour passer les paquets de données au contrôleur lorsqu'aucune entrée de flux ne correspond au paquet entrant ou lorsque l'action de l'entrée correspondante spécifie que le paquet doit être relayé au contrôleur. Le trafic du plan de contrôle (paquet de routage OSPF) sera généralement relayé au contrôleur via ce message PACKET_IN. Si le commutateur dispose d'une mémoire suffisante pour mémoriser les paquets qui sont envoyés au contrôleur, les messages PACKET-IN contiennent une partie de l'en-tête (par défaut 128 octets) et un buffer ID qui peut être utilisé par le contrôleur. Les commutateurs ne supportant pas la mémorisation interne (ou ne disposant plus de mémoire) émettent le paquet entier au contrôleur dans le message PACKET-IN [41].
- ✓ **Flow-Removed :** Le commutateur peut informer le contrôleur qu'une entrée de flux a été supprimée de la table de flux via le message FLOW_REMOVED.
- ✓ **Port-status :** Utilisé pour informer le contrôleur d'un changement sur un port. Le commutateur est censé envoyer des messages d'état du port aux contrôleurs lorsque la configuration du port ou l'état du port change. Ces événements incluent les événements de changement de configuration du port, par exemple si un port a été éteint directement par un utilisateur, ou si le lien est tombé en panne [47].
- ✓ **Error :** Le commutateur est capable d'informer les contrôleurs des problèmes à l'aide de messages d'erreur [47], par exemple, l'ajout d'une entrée de flux par le contrôleur contenant des actions non supportées par le commutateur [41]. La Figure I-11 montre les échanges Openflow entre le contrôleur/commutateur.

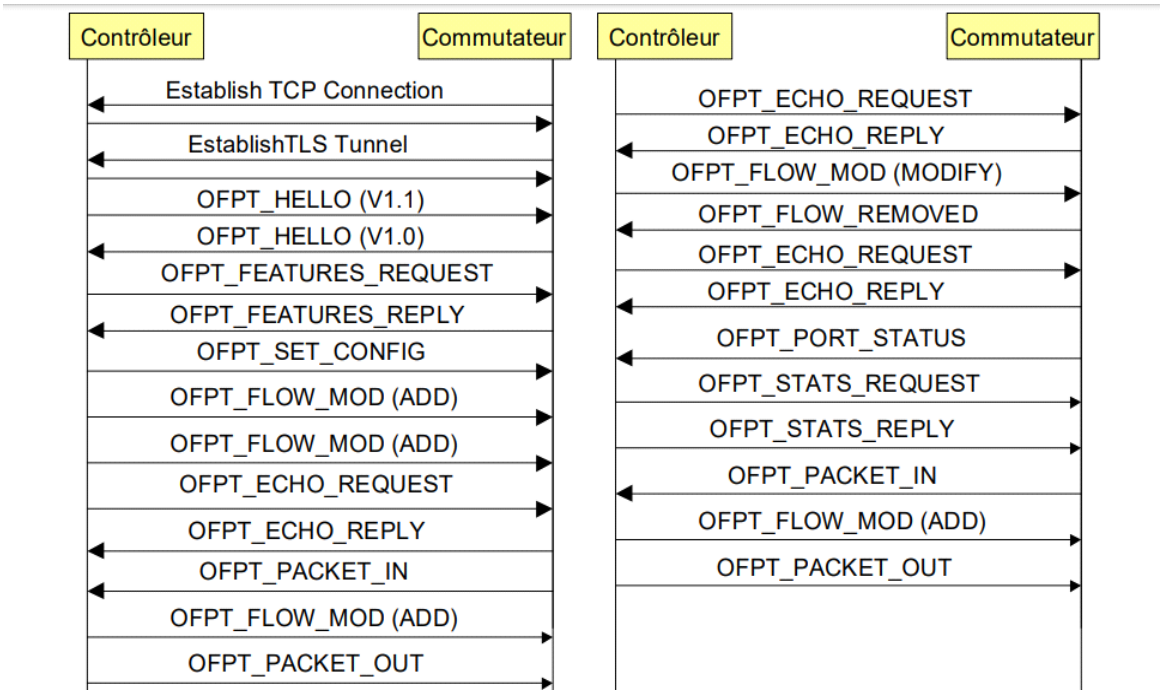

Figure I- 11 : Échanges Openflow entre le contrôleur/commutateur [41]

I.7. Contrôleur

Le contrôleur est le noyau et la partie principale du système d'exploitation réseau dans les réseaux SDN [48]. Car celui-ci reçoit des instructions ou des exigences de la couche application SDN et les relais aux composants réseau, extrait également des informations sur le réseau à partir des équipements et les communique aux applications SDN. Ainsi, il facilite la gestion automatisée du réseau et facilite l'intégration.

Afin de pouvoir interagir avec le réseau, le contrôleur a besoin d'une vue précise de ce dernier. C'est ainsi que le concept de NIB (Network Information Base) a vu le jour. Cette NIB est construite au niveau du contrôleur et permet à ce dernier de savoir comment implémenter chaque ordre abstrait, trouver les équipements qui doivent être reconfigurés, s'assurer de la capacité de ces derniers à implémenter une directive et les API supportées par l'équipement [8].

Le contrôleur se base sur deux modes opérationnels : réactif et proactif.

- ✓ Dans l'approche réactive, les paquets de chaque nouveau flux venant au commutateur sont transmis au contrôleur pour que ce dernier décide du comportement à adopter vis-à-vis du flux. Cette approche prend un temps

considérable lors de l'installation des règles. Latence peut être affectée par les ressources d'un contrôleur, les performances et la distance contrôleur-commutateur [48].

- ✓ Dans l'approche proactive, des règles sont déjà installées dans les commutateurs par le contrôleur. Par conséquent, le nombre de paquets qui envoient au contrôleur est réduit. Dans cette approche, la performance et le surcoût de l'architecture s'améliorent [48].

Plusieurs contrôleurs SDN existent actuellement, commerciales ou Open source. Ces derniers sont différents selon leur architecture langages de programmation, les API qu'ils supportent, les techniques utilisés et performances [16].

Le tableau ci-dessous résume certains des contrôleurs les plus en vigueur actuellement :

Contrôleur	Langage	Architecture	Editeur	Plateforme	Tolérance	API Nord	Aperçu
NOX	Python/C++	Centralisé	Nicira	Linux	NON	Ad-hoc API	Le premier contrôleur Openflow écrit en Python et C++.
POX	Python	Centralisé	Nicira	Linux	NON	Ad-hoc API	Améliorer les performances de NOX .
BEACON		Centralisé multi-thread	Standford	Linux/ Windows	NON	Ad-hoc API	Multiplateforme, modulaire, qui prend en charge les opérations basées sur les événements et les unités d'exécution.
OPEN DAYLIGH	Java	Distribué	Linux Fundation	Linux/ Windows	NON	REST/RES TCONF	Support le framework OSGi et le REST API.

FLOODLIGHT	Java/Python	Centralisé multi-thread	Big Switch	Linux/ Windows/ Mac	NON	RESTful API	Basé sur l'implémentation Beacon. Testé avec des commutateurs OpenFlow physiques et virtuels.
RYU		Centralisé	NTT, OSRG group	Linux	NON	Ad-hoc API	Un NOS qui vise à fournir un contrôle et des API logiquement centralisés pour créer de nouvelles applications de gestion et de contrôle de réseau. Ryu supporte la majorité des versions d'OpenFlow.
HPE VAN	JAVA-based /java	Distribué	Hewelt Packar	HPE VAN			Le contrôleur HPE VAN fournit un point de contrôle unifié dans un réseau compatible OpenFlow, simplifiant la gestion, l'approvisionnement, l'orchestration et permettant la livraison d'une nouvelle génération de services réseau basés sur des applications.

Tableau I- 4 : Comparaison entre les propriétés des différents contrôleurs SDN

I.8. Défis SDN

En dissociant les plans de données et de contrôle des réseaux traditionnels, le SDN promet une nouvelle génération de matériel à faible coût autour duquel les communautés de développement de logiciel peuvent se regrouper pour créer des services réseau rentables et robustes adaptés aux besoins individuels. Cependant, les entreprises matures et les organisations de mise en réseau doivent surmonter plusieurs défis pour profiter pleinement des avantages du SDN [53].

Voici quelques défis courants dans le SDN créés par le changement de paradigme des services définis par logiciel par rapport aux réseaux traditionnels basés sur le matériel [53] :

- **Sécurité :** Étant donné que le plan de contrôle joue une fonction centrale dans une architecture SDN, les stratégies de sécurité doivent se concentrer sur la protection du contrôleur et l'authentification de l'accès d'une application au plan de contrôle [53].

  De nouveaux services peuvent introduire des menaces de sécurité car les programmeurs et les administrateurs réseau peuvent introduire involontairement du code à risque et exposer le réseau à des menaces. Par

conséquent, des pratiques de sécurité appropriées doivent être mises en place pour prévenir de telles attaques [53] [48].

- **Evolutivité (Scalabilité) :** Étant donné que l'architecture SDN comprend des contrôleurs centralisés ou partiellement distribués s'interfaçant avec des plans de données sur plusieurs appareils, il est possible que les contrôleurs deviennent un goulot d'étranglement du réseau [53]. Si trop de paquets atteignent le contrôleur, des problèmes de performances peuvent survenir [54].

  Des solutions ont été proposés par les auteurs, afin d'améliorer l'évolutivité des contrôleurs SDN [7] avec une architecture de contrôle décentralisée ou une solution similaire, telle que des plans de contrôle divisés ou entièrement distribués. Mais de telles solutions peuvent introduire de nouveaux obstacles tels que la convergence et d'innombrables instances de contrôle à configurer et à gérer [53].
- **Interopérabilité :** Pour les nouveaux réseaux, la mise en œuvre du SDN est assez simple. La transition d'un réseau hérité vers le SDN est une autre histoire, car le réseau hérité prend probablement en charge des systèmes commerciaux et de mise en réseau actifs. Les entreprises et la plupart des environnements réseau doivent passer au SDN, ce qui nécessite l'interopérabilité avec une infrastructure hybride héritée-SDN [53].
- **Performance :** est le plus grand problème pour tous les réseaux et constituent un domaine important que les chercheurs essayent toujours à améliorer [7]. Quel que soit le degré de robustesse, de sécurité, d'évolutivité ou d'interopérabilité d'un réseau, il est inutilisable s'il manque de performances [53], et puisque le SDN est une technique basée sur les flux, ses performances sont mesurées en fonction de deux métriques : le temps nécessaire pour instaurer un nouveau flux dans les commutateurs et le nombre de flux que le contrôleur peut traiter par seconde [7] [53].

  La solution à de nombreux problèmes de performances dans les grands réseaux en croissance consiste à pousser plus d'intelligence vers le plan de données ou à passer à une architecture de plan de contrôle [53].

- **Disponibilité :** La disponibilité du contrôleur est le principal aspect à prendre en compte. La dépendance étroite entre les commutateurs et le contrôleur chaque fois qu'un changement de règle est nécessaire pourrait

devenir un problème. De plus, si la conception du réseau ne prend en compte qu'un seul contrôleur centralisé, il pourrait devenir un « point de défaillance unique ». Une approche distribuée pourrait être mise en œuvre pour mieux garantir la disponibilité et éviter les défaillances potentielles. De plus, une solution de redondance ou de sauvegarde pourrait être utilisée pour améliorer la robustesse [54].

I.9. Conclusion

Dans ce premier chapitre, nous avons présenté le SDN et ses avantages. Particulièrement, ce paradigme est compatible aux nouvelles tendances telles-que l'économie d'énergie et l'ingénierie des trafics. Dans le chapitre suivant, on présente les travaux relatifs à l'économie d'énergie avec un résumé des résultats obtenus.

Chapitre II : Etat de l’art sur l’économie d’énergie dans les réseaux SDN

II.1. Introduction

La consommation d'énergie constitue une partie importante des coûts globaux des technologies de l'information et de la communication. L'optimisation énergétique peut être appliquée sur différents composants de l'architecture SDN au niveau applicatif et/ou matériel. Ce chapitre présentera les approches écoénergétiques classées en deux catégories principales : a) stratégies logicielles, b) stratégies matérielles, où chacune sera développée en termes de techniques d'optimisation d'énergie utilisées.

II.2. Techniques d'économie d'énergie dans les réseaux SDN

Les équipements réseaux et les protocoles de communications sont inconscients de l'énergie qu'ils dissipent. Vue l'importance de la consommation énergétique, des travaux de recherche ont été consacrés à étudier et proposer des techniques scientifiques qui permettent d'incorporer la conscience énergétique dans les réseaux SDN. Selon ces études on peut classer les stratégies vertes dans deux catégories : [55]

- **Stratégies logicielles :** peuvent être appliquées sur le contrôleur. Ces stratégies peuvent être classées dans trois catégories : a) la connaissance du trafic, b) la connaissance du système final, c) le placement des règles. Où chacune d'elles se compose de quatre propriétés.
- **Stratégies matérielles :** sont appliquées à un équipement ou sous équipement réseau.

II.3. Conscience du trafic

Le trafic réseau est défini comme l'ensemble des paquets transmis via un réseau à un moment donné. La gestion du trafic consiste à analyser, prévoir et réguler de manière dynamique le comportement des périphériques réseau afin d'optimiser les performances et les exigences de qualité de service. Le contrôleur recueille périodiquement des informations statistiques sur le trafic, l'état des commutateurs, l'état des liaisons et la topologie du réseau. Par la suite, la programmabilité du contrôleur permet de s'adapter à l'évolution de l'environnement réseau [55].

Par contre, les composants du réseau fonctionnent à pleine capacité à tout moment, quelle que soit la demande de trafic. Ainsi, la consommation énergétique du réseau n'est pas proportionnelle à la charge du trafic. En limitant la consommation selon la charge, les approches d'efficacité énergétique tenant

compte du trafic ont le potentiel d'améliorer considérablement l'efficacité énergétique dans le SDN [55].

II.3.1. Elastic Tree

Dans [56], les auteurs proposent le framework «Elastic Tree» pour la gestion de l'énergie des réseaux de centres de données qui est mise en œuvre sur un banc d'essai composé de commutateurs OpenFlow illustrées dans la Figure II-1. L'idée est de désactiver les liens et les commutateurs en fonction de la charge de trafic. Par conséquent, la consommation d'énergie du réseau est rendue proportionnelle à l'évolution dynamique du trafic. La méthode proposée se compose de trois modules logiques : a) modèle formel, b) Bin-Packing gourmand, c) une heuristique sensible à la topologie. Chaque stratégie prend en entrée la topologie du réseau comme un graphe, les contraintes de routage, un modèle d'énergie, la matrice de trafic, et génère un sous-ensemble du réseau (liens, commutateurs) essentiel pour acheminer les flux en cours.

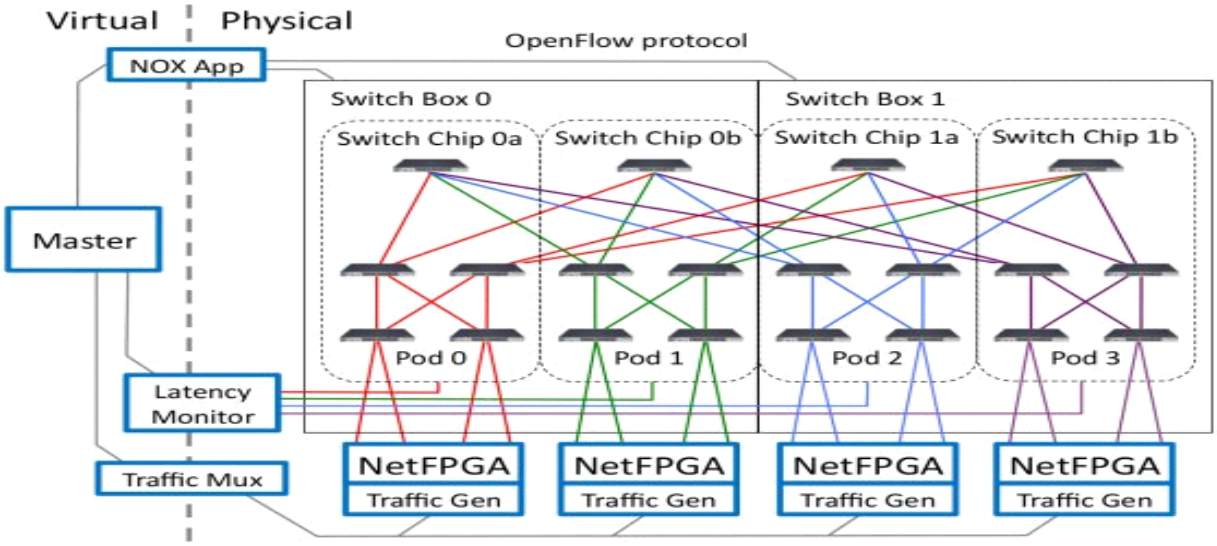

Figure II- 1 : Banc d'essai Elastic Tree [56]

Le modèle formel fournit un outil pour évaluer la qualité de la solution d'autres optimiseurs, en raison de lourdes exigences de calcul. Le but de cette approche est de minimiser l'énergie totale du réseau, tout en satisfaisant toutes les contraintes de trafic. Par la suite, la fonction objective limite le trafic aux liens et commutateurs actifs (sous tension). L'avantage du modèle formel est qu'il garantit une solution flexible dans un optimum configurable. Cependant, le modèle ne peut évoluer que pour 1000 hôtes ou moins.

L'optimiseur gourmand de bin-packing évalue les chemins possibles pour chaque flux et l'affecte au chemin le moins chargé. Ainsi, cet optimiseur

améliore l'évolutivité du modèle formel. Cette approche ne trouvera pas une affectation satisfaisante pour tous les flux, il s'agit d'un problème inhérent à toute stratégie gourmande. Cependant, les solutions peuvent être calculées de manière incrémentielle et efficace en terme de temps de calcule afin de prendre en charge l'utilisation en ligne de l'algorithme.

L'optimiseur heuristique sensible à la topologie, d'autre part, divise les flux tout en cherchant un sous-ensemble de liens minimums pour acheminer tous les flux actifs. Cette heuristique est aussi plus efficace du point de vue du temps de calcul. Mais présente un inconvénient relatif à la dégradation des performances en raison de l'activation et la désactivation des éléments du réseau. Les auteurs ont examiné les compromis entre l'efficacité énergétique, la performance et la robustesse pour chaque stratégie. Ainsi, en matière d'économie d'énergie, les résultats démontrent que ElasticTree peut économiser jusqu'à 50% de l'énergie du réseau.

II.3.2. CARPO

Dans [57], les auteurs proposent un algorithme d'optimisation basé sur la corrélation des flux. Cette solution est appliquée aux réseaux de centre de données. Comme le montre la Figure II-2, L'algorithme « CARPO » suit trois étapes : a) l'analyse de corrélation, b) la consolidation du trafic, c) l'adaptation du débit de liaison pour optimiser la consommation d'énergie.

Dans un premier temps, CARPO prend en entrée les débits des flux et analyse la corrélation entre les différents flux. Dans la deuxième étape, sur la base des coefficients de corrélation de l'analyse précédente, CARPO utilise le débit de données optimal (90%) de chaque liaison au cours de la période précédente pour consolider dynamiquement les flux de trafic sur un petit ensemble de liens et de commutateurs sous la contrainte de capacité de liaison. Après la consolidation, les commutateurs et les ports inutilisés sont désactivés pour économiser de l'énergie. Dans la dernière étape, CARPO adapte le débit de données de chaque lien actif en tenant compte des flux de trafic consolidés sur ce lien. Les résultats obtenus démontrent que l'algorithme CARPO peut économiser jusqu'à 46% de l'énergie par rapport à la consommation ordinaire.

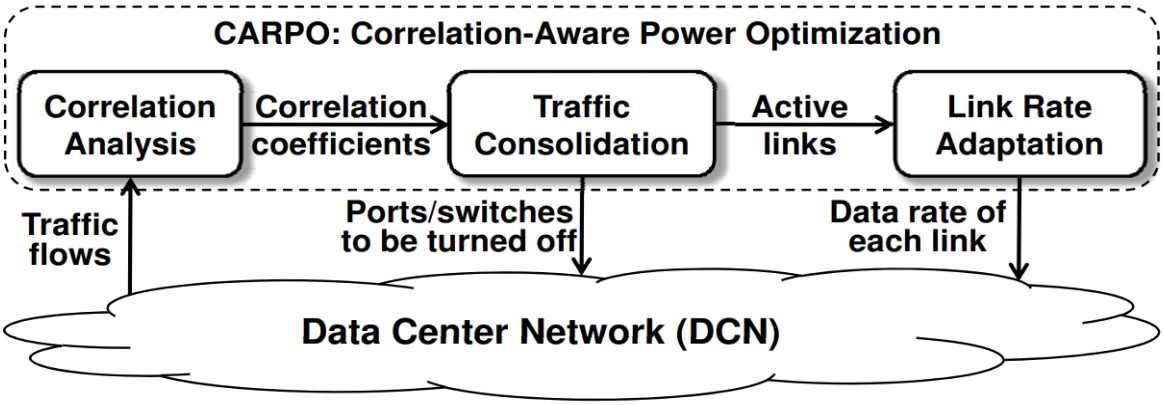

Figure II- 2 : La technique CARPO [57]

II.3.3. FLOWP

Dans le but de réduire la consommation d'énergie dans les centres de données, les auteurs optent pour une autre technique « FLOWP » [58], un algorithme de planification adaptative dynamique basé sur la préemption de flux et le routage sensible à l'énergie, qui économise de l'énergie en diminuant les appareils à faible utilisation, puis en les mettant en mode en veille. Cette technique combine le chemin le plus économe en énergie et se compose de quatre modules : a) le module de gestion des flux réseau (NFM), b) le module d'algorithme de planification des flux (FSA), c) le module de configuration réseau (NC) et d) le module de contrôle de l'alimentation des périphériques (DPC). La relation entre ces derniers est illustrée dans Figure II-3.

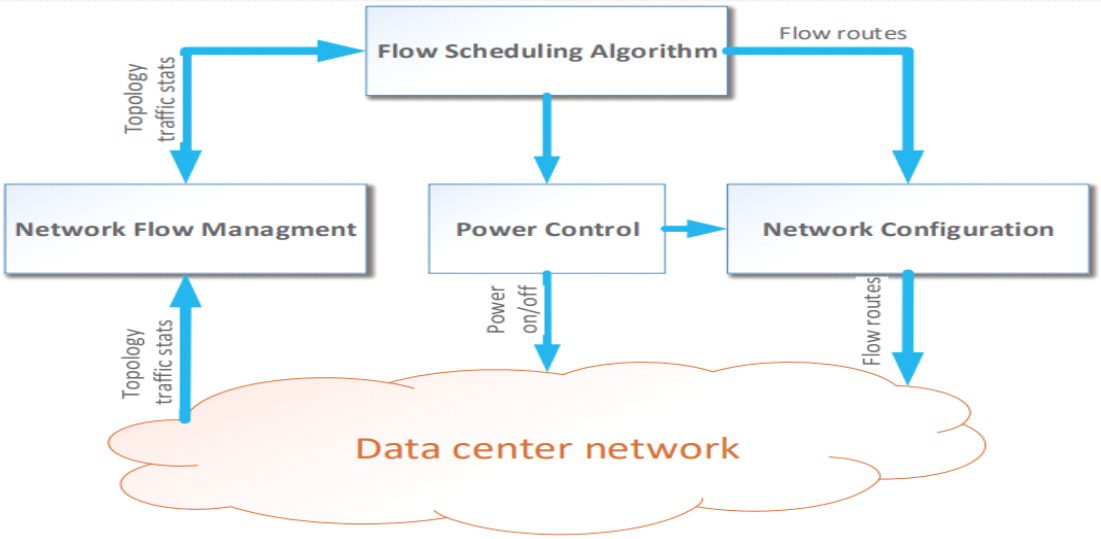

Figure II- 3 : La technique FLOWP [58]

Le module NFM est chargé de collecter en permanence les informations sur la topologie du réseau et les informations pertinentes des flux de donnée, puis transmet ces informations au module FSA en temps réel. Par la suite, la responsabilité de FSA est de trouver un chemin plus économe en énergie en fonction des informations reçues du module NFM. Dans l'algorithme de planification FLOWP, un flux de données doit être envoyé à partir du serveur source. Sur la base du coût de l'énergie, l'algorithme de Dijkstra est adapté pour trouver le chemin approprié. Pour éviter un impact excessif sur le débit du réseau, le chemin sélectionné doit être inactif ou utilisé par un seul flux de données.

Le module NC est responsable des sorties de la stratégie de routage du réseau et de planification des flux vers les commutateurs et serveurs. Et en même temps, le module DPC met les commutateurs et les ports inactifs en mode en veille. Les résultats expérimentaux montrent que sans affecter la QoS, FLOWP pourrait réduire la consommation d'énergie d'environ 30 %.

II.4. Système final

Les solutions d'économie d'énergie sensibles au système final utilisent la pratique consistant à désactiver les serveurs physiques sous-utilisés et à exécuter leurs tâches sur un nombre réduit de serveurs dans des centres de données basés sur SDN. Un centre de données se compose de serveurs interconnectés (hôtes physiques) organisés en racks, dans lesquels le modèle SDN est utilisé pour former une superposition connectant des machines virtuelles, Cette superposition est connue sous le nom de liste de commutateurs et de liens compatibles SDN. Le système final connecté peut être une machine virtuelle ou une machine physique, cette machine hébergeant les machines virtuelles avec la connexion réelle [55].

II.4.1. Honeyguide

Dans [59], les auteurs proposent une recherche sur la topologie « Honeyguide » pour réduire la consommation d'énergie des réseaux en limitant le nombre de commutateurs actifs. La solution "Honeyguide", est une topologie de migration de machines virtuelles pour l'efficacité énergétique dans les réseaux de centres de données. En outre, c'est un optimiseur d'énergie à l'échelle du réseau qui surveille en permanence les conditions de trafic du centre de données et les charges de travail des machines virtuelles. Ce dernier détermine le placement des machines virtuelles qui satisfait aux exigences de redondance et aux limites de capacité des machines physiques. Ensuite, ce mécanisme choisit l'ensemble

des éléments réseau qui doivent rester actifs et éteint autant de liens/commutateurs inutiles que possible.

Pour démontrer l'effet d'économie d'énergie de Honeyguide, des simulations ont été conduites. Pour illustrer comment le Honeyguide réduit la consommation d'énergie des éléments du réseau, celui-ci est appliqué à la topologie «fat-tree». Honeyguide utilise un moteur de surveillance d'hyperviseur qui recueille des statistiques sur le processeur, le réseau et la mémoire des commutateurs.

À l'aide de ces statistiques, Honeyguide détermine quelle machine virtuelle migrer vers quelle machine physique. Tout d'abord, Honeyguide choisit une machine physique à partir de laquelle les machines virtuelles sont déchargées, puis sélectionne une autre machine physique vers laquelle les machines virtuelles sont migrées. L'algorithme de premier ajustement d'origine sélectionne la machine physique la plus chargée comme source et la machine physique là moins chargée comme destination. Ensuite, la machine virtuelle la plus chargée dans la source est migrée vers la destination, si la destination possède la capacité suffisante pour héberger la machine virtuelle entrante. Ce processus est répété jusqu'à ce qu'il n'y ait plus de machines physiques surchargées. Des expériences basées sur des simulations ont montré que Honeyguide peut réduire la consommation d'énergie du réseau mieux que le schéma de pagination de machine virtuelle traditionnel, et ses économies peuvent atteindre 7,8 %.

II.4.2. EQVMP

Dans [60], La solution EQVMP propose un mécanisme de placement de machine virtuelle amélioré, appelé placement de VM conscient de l'efficacité énergétique et de la qualité de service, pour surmonter le problème de la charge de trafic déséquilibrée lors de l'activation et de la désactivation des VM dans le but d'économiser de l'énergie. EQVMP combine trois principes clés : a) réduction des sauts, b) économie d'énergie, c) équilibrage de charge.

- **Réduction des sauts :** est un mécanisme proposé pour partitionner les machines virtuelles en groupes afin que le nombre de machines virtuelles dans chaque groupe soit équilibré et que les coûts entre les différents groupes soient minimisés. La division déséquilibrée des machines virtuelles peut entraîner une congestion dans le réseau.
- **Economie d'énergie :** cette technique recherche un placement multi-ressources efficace en terme énergétique pour garantir que le fonctionnement de chaque machine virtuelle puisse répondre aux exigences. Le module d'économie d'énergie peut décider du nombre

minimum de serveurs dans le centre de données et offre une bande passante suffisante.

- **Equilibrage de charge :** peut améliorer considérablement les performances du réseau. Les machines virtuelles hôtes devraient être soumises aux contraintes d'efficacité énergétique et de réduction des sauts. Le SDN est utilisé pour l'équilibrage de charge, tente de réaliser une transmission de flux dans des réseaux sans congestion.

Les expériences prouvent qu'EQVMP peut fournir un meilleur taux de croissance du système par rapport à d'autres stratégies. EQVMP surpasse les autres schémas de placement, bien que les performances en matière de puissance et de délai soient les meilleures, et atteint un débit 10 fois supérieur à celui du placement sensible à la puissance et au délai.

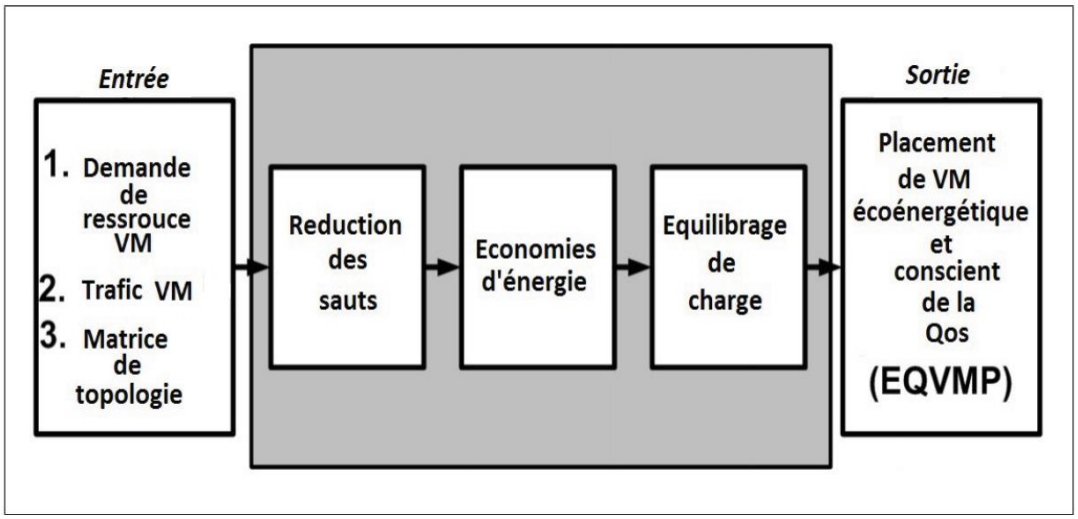

Figure II- 4 : Energy-efficient with QoS-aware VM Placement algorithme [60]

II.4.3. PowerNets

Dans [61], les auteurs proposent une technique PowerNetS, un framework d'optimisation de l'alimentation qui coordonne les serveurs ainsi que le refroidissement pour minimiser la consommation d'énergie d'un centre de données. Sa conception est basée sur les principales observations selon lesquelles les charges de travail des différents serveurs et les flux ne culminent pas exactement au même moment.

La Figure II-5 montre le framework de PowerNetS, qui est composé d'un module de consolidation de machines virtuelles et d'un module de consolidation

de trafic. Les deux modules analysent périodiquement les charges de travail du serveur et les flux, respectivement, et prennent les décisions de consolidation de manière coordonnée. La caractéristique principale du module de consolidation de VM est sa prise en compte de la topologie réseau.

Tout d'abord, le module essaie de placer les machines virtuelles qui sont liées via un flux réseau sur le même serveur, ou sur des serveurs aussi proches que possible, de sorte que le nombre requis de sauts est minimisé. Deuxièmement, le module de consolidation du trafic est impliqué pour vérifier s'il existe un chemin physique disponible pour le flux à affecter et si le flux peut être consolidé avec des flux existants pour économiser l'énergie DCN.

En tirant parti de la consolidation des serveurs et de la corrélation des charges de travail, PowerNets permet d'économiser jusqu'à 51 % d'énergie.

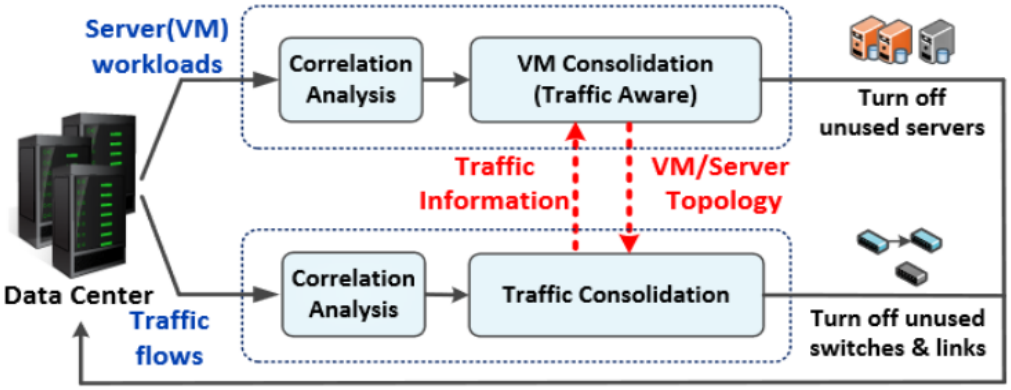

Figure II- 5 : Le framework proposé de PowerNetS [61]

II.5. Placement des règles

Le routage dans le SDN est déterminé par le plan de contrôle et appliqué par le transfert dans le plan de données. En outre, les techniques de placement de règles se concentrent sur la façon de placer les règles dans les commutateurs. Ensuite, Le contrôleur fournit un mécanisme pour convertir les politiques de haut niveau en règles compréhensibles par le commutateur, en tenant compte des politiques de réseau et des politiques de point d'extrémité qui sont une exigence de service réseau qui décide de la liaison sortante que chaque paquet doit être transmis. Ainsi, les règles sont placées sur les commutateurs respectifs répartis sur le réseau est un problème NP-difficile et nécessite donc des solutions heuristiques.

II.5.1. Pallete

Dans [62], les auteurs proposent une méthode, nommée Palette, distribuée appliquée aux tables des commutateurs. Étant donné que la table du contrôleur SDN ne peut gérer que des centaines d'entrées et que la mémoire est coûteuse et gourmande en énergie, Palette décompose les grandes tables SDN en petites tables, puis les distribue sur le réseau, tout en préservant la sémantique globale de la politique SDN.

Cette méthode est particulièrement efficace car la taille des tables de commutation peut devenir un goulot d'étranglement lors du passage à l'échelle des SDN. De plus, ceci facilite la gestion de l'hétérogénéité des commutateurs du réseau et des changements d'équipements à travers le réseau.

II.5.2. FTRS

Afin de réduire le nombre d'entrées de flux, il est intuitif de combiner plusieurs entrées. Cependant, la plupart des fonctions SDN sont réalisées sur la base d'entrées de flux, telles que la gestion de la transmission, la reconnaissance de flux, la collecte de données, etc. Par conséquent, chaque entrée de flux est également importante pour la gestion SDN. Au contraire, si la table de flux est pleine, le commutateur n'est pas en mesure de traiter de nouveaux flux. La conséquence peut être aussi légère qu'une panne de transmission et aussi grave qu'un effondrement du réseau. Pour résumer, un compromis doit être fait entre la fonctionnalité SDN complète et la robustesse du service réseau.

Dans [63] les auteurs présentent un schéma appelé Flow Table Reduction Scheme (FTRS) pour réduire le nombre d'entrées de flux sortantes en combinant les entrées qui sont moins importantes pour traiter le problème de congestion de la table de flux (FTCP) dans le SDN.

Les performances de FTRS sont évaluées à l'aide de simulations Mininet. Dans ces simulations, plusieurs topologies ont été examinées. FTRS produit des performances parfaites dans les topologies en étoile et en arbre avec une réduction des nombre d'entrées de flux de 99 %.

II.6. Approches matérielles

Cette approche tente de minimiser la mémoire utilisée pour stocker les informations dans les commutateurs. Dans l’architecture SDN, les commutateurs utilisent la mémoire TCAM, qui est un type spécialisé de mémoire à grande vitesse. Cette dernière effectue une recherche sur toute la mémoire dans un seul cycle d'horloge. Cependant, la mémoire TCAM est très coûteuse et énergivore. Une stratégie pour utiliser efficacement la TCAM en termes de coût et d’énergie

peut être obtenue en compactant la TCAM elle-même où en compressant les informations qui y sont stockées [55].

Les types de compression qui peuvent être appliqués dans la TCAM sont la compression des règles et la compression du contenu. Dans une liste de contrôle d'accès traditionnel, une règle comporte cinq composantes : plage source, plage de destination, protocole, ports et action [55].

II.6.1. Bit Weaving

Dans [64], les auteurs proposent le framework « Bit Weaving », le premier schéma de compression sans préfixe. Ce dernier est basé sur l'observation des entrées TCAM qui ont la même décision, mais dont les prédicats diffèrent d'un seul bit peuvent être fusionnés en une seule entrée. Le « bit Weaving » consiste en deux nouvelles stratégies, l'échange de bits et la fusion de bits. D'abord, il échange des bits pour les rendre similaires, puis fusionne ces règles ensemble. Les principaux avantages de cette approche sont la rapidité d'exécution, l'efficacité est peut-être complémentaire à d'autres méthodes d'optimisation TCAM en tant que routine de pré / post-traitement.

II.6.2. Campact TCAM

Une approche « Compact TCAM » qui réduit la taille des entrées de flux est proposée dans [65], qui exploite l'interface de programmation et la détermination dynamique des actions pour chaque flux au niveau des commutateurs SDN. Cette dernière utilise des balises plus courtes pour identifier les flux par rapport au nombre initial de bits utilisés. Pour ce faire, le contrôleur attribue un Flow-ID numérique à chaque flux qui peut identifier de manière unique les paquets dans un flux en question. Les en-têtes de paquets sont modifiés au niveau des commutateurs d'entrée pour transporter le Flow-ID qui est utilisé par d'autres commutateurs sur le chemin afin de classer les paquets puis effectuer les actions qui y sont associées. En outre, cette technique permet d'économiser environ 80% d'énergie et d'optimiser les coûts en augmentant le nombre de flux qui peuvent désormais être stockés dans les mêmes tables de flux.

Catégorie	*Approches*	*Description*
Système final	Honeyguide	• Combine la consolidation du trafic et des machines virtuelles pour atteindre l'efficacité énergétique maximale des réseaux de centres de données. • Tolérance aux pannes. • Déploiement facile. • Cette approche ajoute des liaisons de contournement entre les commutateurs de niveau supérieur et les machines physiques en tant qu'extension de la topologie fat-tree existante. • Les résultats expérimentaux montrent une augmentation relative des économies d'énergie atteint jusqu'à 7,8 %.
	EQVMP	• Dans EQVMP, les économies d'énergie sont principalement réalisées par le placement de VM. • L'algorithme dépend sur l'équilibrage de charge qui permet la transmission de flux dans les réseaux sans congestion. • Contrairement à ElasticTree, la mise sous tension et hors tension est appliquée aux serveurs eux-mêmes au lieu des commutateurs. • Sensible à la qualité de service pour les centres de données basés sur SDN. • Les résultats expérimentaux montrent une économie d'énergie de 25 %.

	PowerNets	• Une technique d'optimisation conjointe pour la consolidation des serveurs et l'optimisation du réseau. • Basé sur la topologie fat-tree. • Permet d'économiser jusqu'à 51 % d'énergie.
Emplacement de la règle	FTRS	• Ce mécanisme traite le problème de congestion de la table de flux (FTCP) dans le SDN. • L'objectif de cette technique est de réduire le nombre d'entrées de flux de plus de 95% .
	Pallete	• C'est une approche distribuée appliquée aux tables SDN. • Facilite la gestion de l'hétérogénéité des commutateurs du réseau et des changements d'équipements. • L'implémentation du framework Palette est basée sur des algorithmes de formulation de la théorie des graphes et des heuristiques.

Conscient du trafic	Elastic Tree	• Un système d'adaptation dynamique à l'évolution du trafic basé sur trois algorithmes : a) la résolution par Cplex du modèle formel, b) l'algorithme glouton et c) une heuristique sensible à la topologie. • Les traces de trafic utilisées pour le test sont des traces du monde réel (R). • La topologie Fat Tree est utilisée dans les évaluations de l'algorithme . • Le contrôleur utilisé dans l'expérience est NOX. • L'objectif d'optimisation est de minimiser le nombre de liens/ports inactifs et économiser jusqu'à 50% d'énergie.
	CARPO	• Une approche basée sur la consolidation dynamique du trafic sensible à la corrélation entre les flux avec l'adaptation du débit de liaison. • Les traces de trafic utilisées pour le test sont des traces du monde réel (R) • Basé sur la topologie fat-tree • L'objectif d'optimisation est de diminuer le nombre de liens/ports inactifs et économiser jusqu'à 46% de l'énergie du réseau.

	FLOWP	• Un algorithme de planification adaptative dynamique basé sur Dijkstra pour router les paquets avec un chemin plus économe en énergie. • Les traces de trafic utilisées pour le test sont des traces générées artificiellement (A). • Basé sur la Topologie fat-tree. • L'objectif est de mettre les commutateurs inactifs en veille et économiser 30 % d'énergie. • Meilleure qualité de service par rapport à ElasticTree et CARPO**.**
Matérielles	Bit Weaving	• Compression préfixe-free. • Une technique basée sur la fusion d'entrées TCAM adjacentes qui ont la même décision et ont une distance de Hamming d'un seul bit en remplaçant ce bit par $*$. • Le but est de réduire considérablement la taille d'informations stockées dans le TCAM .
	Campact TCAM	• Une méthodologie basée sur l'utilisation des balises courtes pour stocker les entrées de flux. • Le Flow-ID est attribuée par le contrôleur. • La modification des en-têtes des paquets arrivés se fait au niveau de commutateur d'entrée. • L'objectif est d'optimiser l'espace TCAM et économiser environ 80% d'énergie.

Tableau II- 1 : Comparaison entre les méthodes d'économie d'énergie dans les réseaux SDN

II.7. Conclusion

La programmabilité d'un réseau, une propriété puissante du SDN, permet d'apporter plus de flexibilité, une meilleure gestion des ressources réseau et représente une grande opportunité d'améliorer les performances du réseau en général et l'efficacité énergétique en particulier.

La sensibilisation à l'énergie est devenue une exigence de conception importante pour les mécanismes de mise en œuvre des réseaux modernes, et la conception de solutions économes en énergie n'est pas une tâche facile, car elle doit tenir compte du compromis entre l'efficacité énergétique et les performances du réseau. Dans ce chapitre, nous avions présenté les politiques d'efficacité énergétique conçues pour les réseaux SDN. Nous avons classifié les solutions éco-énergétiques dans le SDN en sous-catégories de gestion du trafic, de gestion du système final et de placement de règles. Puis, nous avons présenté les principaux concepts des solutions dans chaque catégorie.

Chapitre III : Expérimentation avec l’algorithme GreenSDN et la topologie Fat-Tree

III.1. Introduction

Ces dernières années, afin de répondre au besoin de réduction de la consommation énergétique des infrastructures réseaux, un grand nombre de politiques réseaux verts et économes en énergie ont été inventés. Au début de ce chapitre, nous présenterons les outils que nous avons utilisés pour cette expérimentation et leur utilité. Ainsi, on va présenter Mininet en détaille. En outre, nous introduisons les paramètres de simulations pour quantifier la différence des performances avec et sans prise en charge de l'économie d'énergie. La méthode d'économie d'énergie qu'on a utilisée pour mettre en évidence l'efficacité de la technologie SDN a été publiée avec son code source dans [66].

III.2. Outils

- **Open vSwitch :** Open vSwitch est un commutateur logiciel multicouche sous licence open source Apache 2. Celui-ci peut être contrôlé via OpenFlow et peut fonctionner à la fois comme un serveur virtuel et comme une pile de contrôle pour les commutateurs matériels. Ce composant est une plate-forme de qualité de production conçu pour permettre une automatisation efficace du réseau via des extensions programmatiques, tout en prenant en charge les interfaces et protocoles de gestion standard. Alors que le comportement de transfert peut être contrôlé par OpenFlow, OVS a de nombreux autres aspects qui sortent du cadre d'OpenFlow. La configuration et les états d'OVS sont conservés dans la base de données Open vSwitch (OVSDB), qui peut être interrogée et manipulée via le protocole OVSDB basé sur JSON. POX prend en charge le protocole OVSDB [68] [69].
- **Oracle VM VirtualBox :** Oracle VM VirtualBox est une application gratuite, open source et multiplateforme pour crée, gérer et exécuter des machines virtuelles (VM). Ce dernier permet de configurer une ou plusieurs machines virtuelles (VM) sur une seule machine physique et de les utiliser simultanément, avec la machine hôte/physique. Chaque machine virtuelle peut exécuter son propre système d'exploitation, y compris les versions de Microsoft Windows, Linux, BSD et MS-DOS [69].
- **MobaXterm :** MobaXterm fournit tous les outils d'accès à distance importants (SSH, Telnet, Rlogin, ...) sur le bureau Windows. MobaXterm est activement développé et fréquemment mis à jour par Mobatek [70].

- **POX :** POX [71] est un contrôleur OpenFlow/SDN open source basée sur Python. Celui-ci est utilisé pour accélérer le développement et le prototypage de nouvelles applications réseau. Ce contrôleur peut être lancé avec différents paramètres selon des topologies réelles ou expérimentales, ainsi il permet d'exécuter des expériences sur du matériel réel, des bancs d'essai ou avec l'émulateur Mininet. De plus, POX permet d'exécuter facilement des expériences OpenFlow/SDN.

  POX s'appuie sur un modèle basé sur des composants dans lequel l'ensemble des éléments du réseau ainsi que les activités sont reconnues comme des composants distincts pouvant être isolés et utilisés là où ils sont nécessaires [72]. Les modules POX sont en fait des programmes python supplémentaires qui peuvent être utilisés dans le cas où POX est lancé à partir de l'invite de ligne de commande, ces modules exécutent les fonctionnalités réseau dans SDN. [72]
- **ONOS :** ONOS est un contrôleur conçu pour aider les fournisseurs de services réseau à créer des réseaux SDN de classe opérateur conçus pour une évolutivité, une disponibilité et des performances élevées. Bien que spécifiquement conçu pour répondre aux besoins des fournisseurs de services, ONOS peut également servir de plan de contrôle de réseau SDN pour les réseaux locaux (LAN) de campus d'entreprise et les réseaux de centres de données.

  Le cœur d'ONOS est basé sur une architecture modulaire, par opposition à un système intégré qui brouille la division entre ses composants. Étant donné que les fournisseurs de services ont besoin de pouvoir faire évoluer leurs réseaux, le contrôleur ONOS peut évoluer pour s'adapter à un système d'appareils physiquement distribué. Cela permet aux fournisseurs de services d'ajouter de nouveaux commutateurs ou composants sans perturber le reste du système [73].
- **Python :** Le langage Python est un langage de programmation open source multi-plateformes et orienté objet. Grâce à des bibliothèques spécialisées, Python est utilisé actuellement pour de nombreuses situations comme le développement logiciel, l'analyse de données, automatisation des systèmes ou la gestion d'infrastructures [74]. En effet, parmi ses qualités, Python permet notamment aux développeurs de se concentrer sur ce qu'ils font plutôt que sur la manière dont ils le font. Ainsi, développer du code avec Python est plus rapide qu'avec d'autres langages. [75]

❖ **Mininet :** Mininet est un émulateur permettant de déployer de grands réseaux sur les ressources limitées d'un simple ordinateur. Il permet à l'utilisateur de créer, personnaliser, partager et tester facilement des réseaux SDN [66]. Ce dernier aussi prend en charge la recherche, le développement, l'apprentissage, le prototypage, les tests, le débogage et toute autre tâche qui pourrait bénéficier d'un réseau expérimental complet [76]. En outre, l'émulateur Mininet permet d'exécuter de la simulation de manière interactive [77].

Mininet permet de créer des topologies de très grande taille jusqu'à des milliers de nœuds et d'effectuer des tests facilement. Il dispose d'outils de ligne de commande et d'une API très simples [66]. Un réseau Mininet se compose de :

- ✓ **Hôtes isolés :** un groupe de processus déplacé dans un espace de noms de réseau qui fournit la propriété exclusive des interfaces, des ports et des tables de routage. [77]
- ✓ **Liens émulés :** Linux Traffic Control (TC) applique le débit de données de chaque lien pour façonner le trafic à un débit configuré [77].
- ✓ **Commutateurs émulés :** le pont Linux par défaut ou l'Open vSwitch exécuté en mode noyau est utilisé pour commuter les paquets entre les interfaces [77].

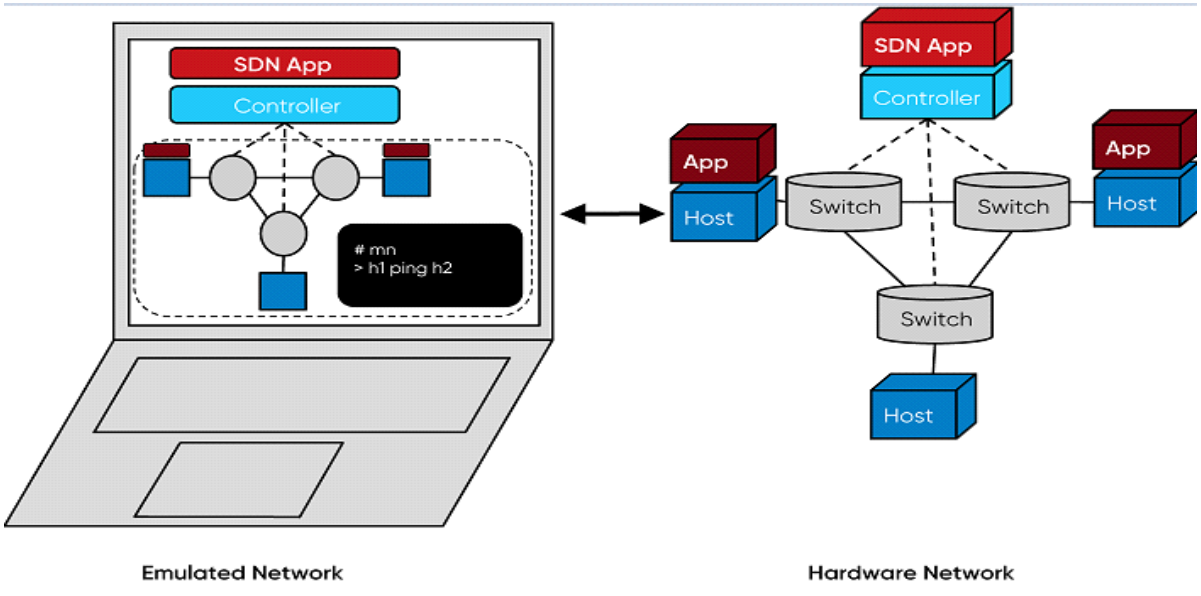

Figure III- 1 : Émulation de réseaux avec Mininet [77]

Mininet contient un certain nombre de topologies par défaut telles que minimale,

simple, inversée, linéaire et arborescente. Comprendre la méthode de dénomination des interfaces, des hôtes et des commutateurs est essentiel pour prospérer avec Mininet [66]. Nous allons donner une explication simplifiée de ces topologies.

- ✓ **Minimale :** est une topologie qui contient seulement un commutateur OpenFlow et 2 Hôtes. Les liens entre le commutateur et deux hôtes sont également créés. [66] La commande pour crée cette topologie est la suivante : *$ sudo mn –topo minimal.*
- ✓ **Simple :** Il s'agit d'une topologie simple avec un commutateur Openflow et k hôtes. Les liens entre le commutateur et k hôtes sont aussi créés. [66] Pour crée cette topologie, la commande suivante est utilisée : *$ sudo mn –topo single, K.*
- ✓ **Inversée :** Elle est similaire à la topologie simple mais l'ordre de connexion est inversé [66]. La commande suivante crée cette topologie *$ sudo mn –topo reversed, K* (k le nombre des hôtes).
- ✓ **Linière :** Il s'agit d'une topologie de type bus composée par n commutateurs Openflow et k hôtes. [66] Cette commande crée la topologie linière : *$ sudo mn –topo linear,n,K* .
- ✓ **Arborescente :** La topologie arborescente contient k niveaux et 2 hôtes sont connectés à chaque commutateur [66]. Cette commande pour crée la topologie : *$ sudo mn –topo tree, k.*

III.3. Simulation avec Mininet

III.3.1. Création de topologies personnalisée

Mininet permet de créer et simuler une topologie personnalisée. Cette méthode est la plus utilisée parce qu'elle permet de construire d'autres topologies que bus, tree, ou ring. En dessous un exemple d'une topologie personnalisée :

lab1.py

```
from mininet.topo import Topo
class MyTopo( Topo ):
     def __init__( self ):
        "Create custom topo." # Initialize topology
        Topo.__init__( self )
        # Add hosts and switches
        leftHost = self.addHost( 'h1' )
        rightHost = self.addHost( 'h2' )
        leftSwitch = self.addSwitch( 's3' )
        rightSwitch = self.addSwitch( 's4' )
        # Add links
        self.addLink( leftHost, leftSwitch )
        self.addLink( leftSwitch, rightSwitch )
        self.addLink( rightSwitch, rightHost )

topos = { 'mytopo': ( lambda: MyTopo() ) }
```

Figure III- 2 : Création d'une topologie personnalisée

Voici quelques explications sur le code précédant [29] :

- **Mininet :** utilisé comme classe principale pour créer et gérer un réseau.
- **Topo :** une classe de base pour les topologies Mininet.
- **AddHost (nom, cpu = f) :** ajoute un hôte à la topologie avec un paramètre obligatoire et un deuxième optionnel. Le premier paramètre spécifie le nom de l'hôte et le deuxième spécifie la fraction de la capicité CPU globale du système qui doit être allouée à l'hôte virtuel, par exemple : *addHost ('h1', cpu=.5)* spécifie que h1 obtient 50 % du processeur.
- **AddSwitch () :** ajoute un commutateur à la topologie et renvoie le nom du commutateur, par exemple s1.
- **AddLink (nœud1, nœud2, **options de lien) :** ajoute un lien bidirectionnel qui contient trois paramètres. Le premier et le deuxième paramètre spécifient respectivement le nom de l'hôte et du commutateur et le troisième paramètre optionnel spécifie les caractéristiques telles que la bande passante, retard et perte de paquets ainsi que la taille de la file d'attente. Par example: *addLink (nœud*1, nœud2, bw=*10, delay='10ms', loss=0, max_queue_size=1000).*

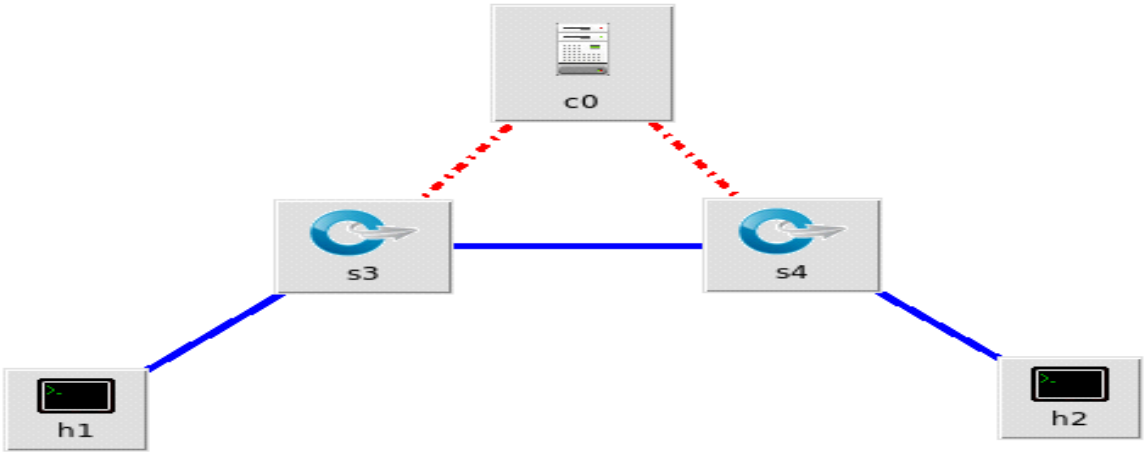

Figure III- 3 : Diagramme de topologie customisée

```
mininet@mininet-vm:~/mininet/custom$ sudo mn --custom lab1.py --topo mytopo --te
st pingall
[sudo] password for mininet:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s3 s4
*** Adding links:
(h1, s3) (s3, s4) (s4, h2)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 2 switches
s3 s4 ...
*** Waiting for switches to connect
s3 s4
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 1 controllers
c0
*** Stopping 3 links
...
*** Stopping 2 switches
s3 s4
*** Stopping 2 hosts
h1 h2
*** Done
```

Figure III- 4 : Topologie du fichier lab1.py

La commande à exécuter pour créer cette topologie à partir du scripte Python et la suivante :

$ sudo mn --custom lab1.py --topo mytopo --test pingall

- L'option *--custom lab1.py --topo mytopo* permet d'utiliser la topologie personnalisée définie dans lab1.py.
- **PingAll :** Ceci est utilisé pour vérifier la connectivité entre tous les nœuds dans la topologie.

III.3.2. Implémentation d'un contrôleur dans une topologie personnalisée

CustomTopo.py

```
from mininet.node import CPULimitedHost
from mininet.link import TCLink
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel
from mininet.node import Controller, RemoteController
import os
class POXcontroller1( Controller):
        def start(self):
                self.pox='/home/mininet/pox/pox.py' #%os.environ['HOME'] /root
                self.cmd(self.pox, "lab3_1_controller &")
        def stop(self):
                self.cmd('kill %' +self.pox)
controllers = { 'poxcontroller1': POXcontroller1}
class SingleSwitchTopo(Topo):
        "Single switch connected to n hosts."
        def __init__(self, n=2, **opts):
                Topo.__init__(self, **opts)
                switch = self.addSwitch('s1')
                # Each host gets 50%/n of system CPU
                h1=self.addHost('h1', cpu=.5/n)
                h2=self.addHost('h2', cpu=.5/n)
                # 10 Mbps, 10ms delay, 0% loss, 100packet queue
                self.addLink('h1', switch, bw=10,
                delay='10ms', loss=0,
                max_queue_size=1000, use_htb=True)
                self.addLink('h2', switch, bw=10,
                delay='10ms', loss=0,
                max_queue_size=1000, use_htb=True)

topos={'mytopo' : (lambda :SingleSwitchTopo(3))}
```

Figure III- 5 : Définition du contrôleur et de la topologie

```
def perfTest():
        "Create network and run simple performance test"
        topo = SingleSwitchTopo(n=2)
        net = Mininet(topo=topo,host=CPULimitedHost, link=TCLink,controller=RemoteController)
        c1=net.addController('c1',controller=RemoteController,ip="127.0.0.1",port=6633)
        net.start()
        c1.start()
        print("Dumping host connections")
        dumpNodeConnections(net.hosts)
        print("Testing network connectivity")
        net.pingAll()
        print("Testing bandwidth between h1 and h2")
        h1, h2 = net.get('h1', 'h2')
        net.iperf((h1, h2))
        net.stop()
if __name__ == '__main__':
        setLogLevel('info')
        perfTest()
```

Figure III- 6 : Fichier CustomTopo.py partie 2: procédure de lancement de la topologie

lab3_1_controller.py

```
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpidToStr
log = core.getLogger()
def _handle_ConnectionUp (event):
        msg = of.ofp_flow_mod()
        msg.priority =1
        msg.idle_timeout = 0
        msg.hard_timeout = 0
        msg.match.in_port =1
        msg.actions.append(of.ofp_action_output(port = 2))
        event.connection.send(msg)
        msg = of.ofp_flow_mod()
        msg.priority =1
        msg.idle_timeout = 0
        msg.hard_timeout = 0
        msg.match.in_port =2
        msg.actions.append(of.ofp_action_output(port = 1))
        event.connection.send(msg)
def launch ():
        core.openflow.addListenerByName("ConnectionUp",
        _handle_ConnectionUp)
```

Figure III- 7 : Définition de la gestion d'événements de la topologie

Ce script python commence par la déclaration des bibliothèques utilisées, puis déclare une fonction 'main' appelée perfTest qui sera exécutée à la fin du script. Avec ce code, nous avons créé une topologie composée par un commutateur attaché à deux hôtes. Ce réseau est géré par un contrôleur accessible à l'adresse 127.0.0.1 et sur le port 6633. Voici quelques significations du code CustomTopo.py :

- **Start () :** démarre l'émulation réseau.
- **Stop () :** arrête l'émulation réseau.
- **Net.hosts :** liste de tous les hôtes du réseau.
- **AddController () :** ajouter un contrôleur à la topologie.
- **DumpNodeConnections () :** Affiche les connexions entre les nœuds.
- **Net.iperf () :** permet de générer un traffic entre deux hôtes
- **Net.get(nom) :** permet de récupérer un nœud correspondant au nom.
- **Cmd () :** utilisé pour exécuter une commande depuis une machine.

La partie cadrée dans Figure 22 peut être remplacée par la commande suivante :

$ sudo mn --custom CustomTopo.py --topo mytopo --controller=remote,ip=172.0.0.1,port=6633

L'option --controller=remote permet à Mininet de connecter les commutateurs avec un contrôleur externe.

Par défaut, l'adresse et le numéro de port du contrôleur prennent les valeurs {172.0.0.1 :6633}. Le code dans le fichier lab3_1_controller.py permet d'ajouter deux règles dans la table de flux uniquement lorsque l'événement se produit, c'est-à-dire lorsque le commutateur se connecte au contrôleur. La première règle dicte que lorsque les paquets arrivent du port 1, le commutateur les retransmet sur le port 2. Et vice versa dans la deuxième règle.

- **idle_timeout :** est le nombre de secondes après lesquelles une entrée de flux est supprimée de la table des flux si aucun paquet correspondant à ce flux n'est reçu au cours de cette durée.
- **hard_timeout :** est le nombre de secondes après lesquelles l'entrée de flux est supprimée de la table de flux.
- **_handle_ConnectionUp (event) :** définit la procédure de traitement l'événement associé à la connexion d'un commutateur au contrôleur.

- **addListenerByName() :** associe un événement à une procédure qui gère cette événement. Le premier paramètre de cette procédure est le nom de l'événement et le deuxième représente la fonction à exécuter à chaque fois que cet événement est déclenché.

Pour exécuter cette topologie, il faut d'abord lancer POX comme suit :

```
mininet@mininet-vm:~$ cd /home/mininet/pox
mininet@mininet-vm:~/pox$ sudo python pox.py lab3_1_controller
[sudo] password for mininet:
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
clear
```

Figure III- 8 : Lancement du contrôleur POX

Par la suite, la commande suivante est exécutée pour émuler la topologie : *$ sudo python CustomTopo.py*

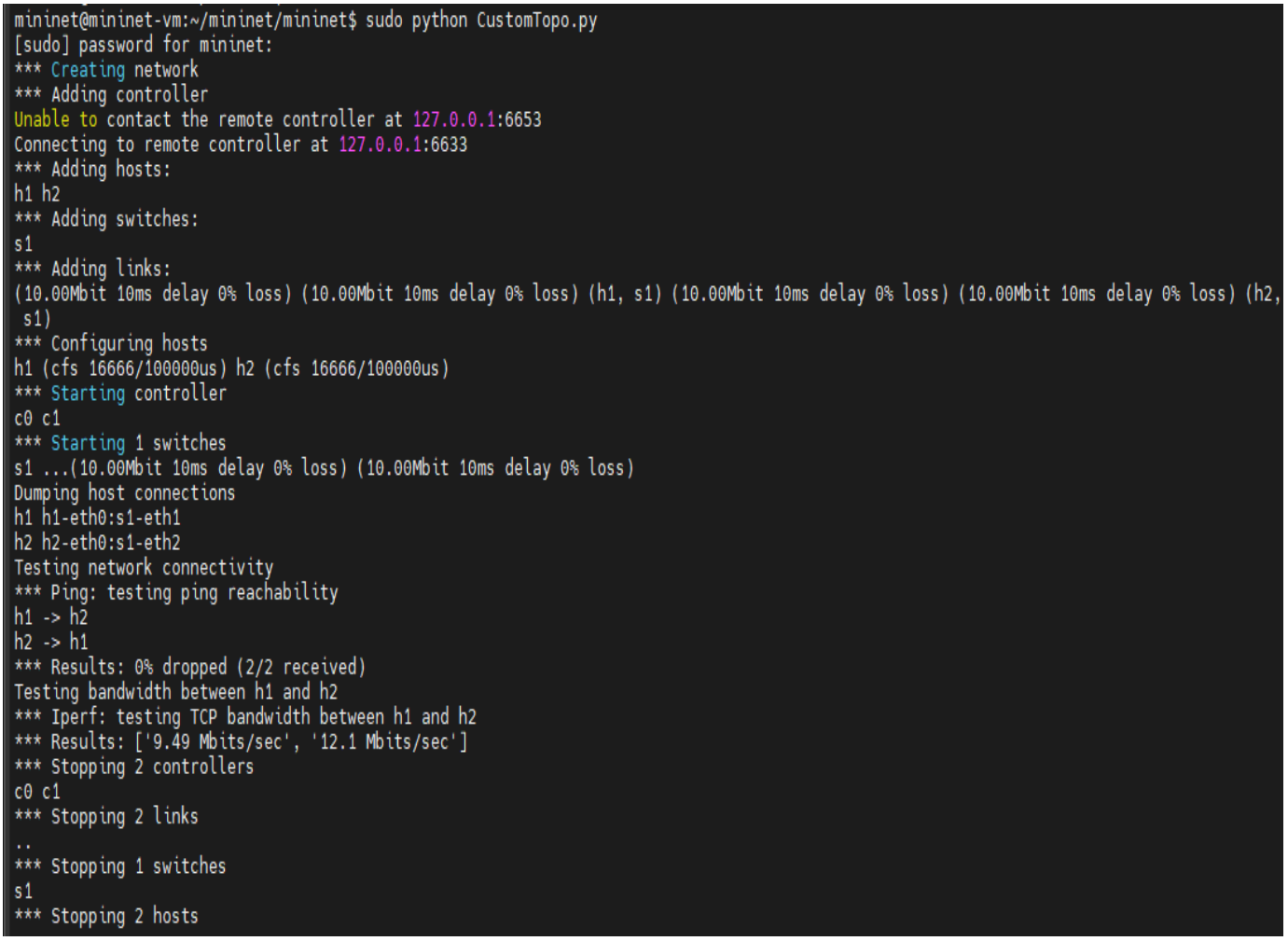

Figure III- 9 : Emulation d'un topologie personnalisée

III.4. Expérimentation avec une technique d'économie d'énergie dans le SDN

Dans cette section, nous étudions une approche de routage éco-conscient qui nous a permet de limiter le nombre des commutateurs d'agrégation et les commutateurs core afin d'économiser de l'énergie. La simulation topologie a été faite grâce à l'émulateur Mininet. En outre, la gestion de la topologie a été réalisée grâce au contrôleur ONOS. La figure ci-dessus montre la topologie Fat-tree de 4 degrés sur laquelle nous avons appliqué cette technique.

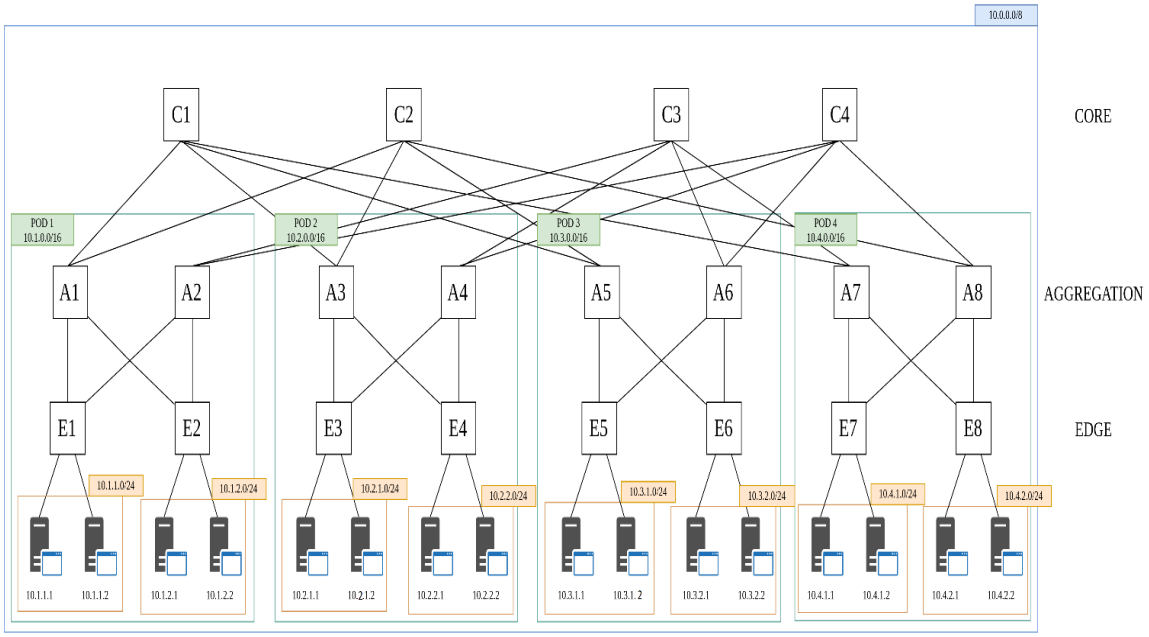

Figure III- 10 : Fat-tree 4 degrés

Les étapes suivantes illustrent la simulation de la politique d'économie d'énergie que nous avons choisie. Dans le premier terminal on a démarré le controlleur ONOS avec la commande suivante :

- ~$ cd onos
- ~/onos$ bazel run onos-local

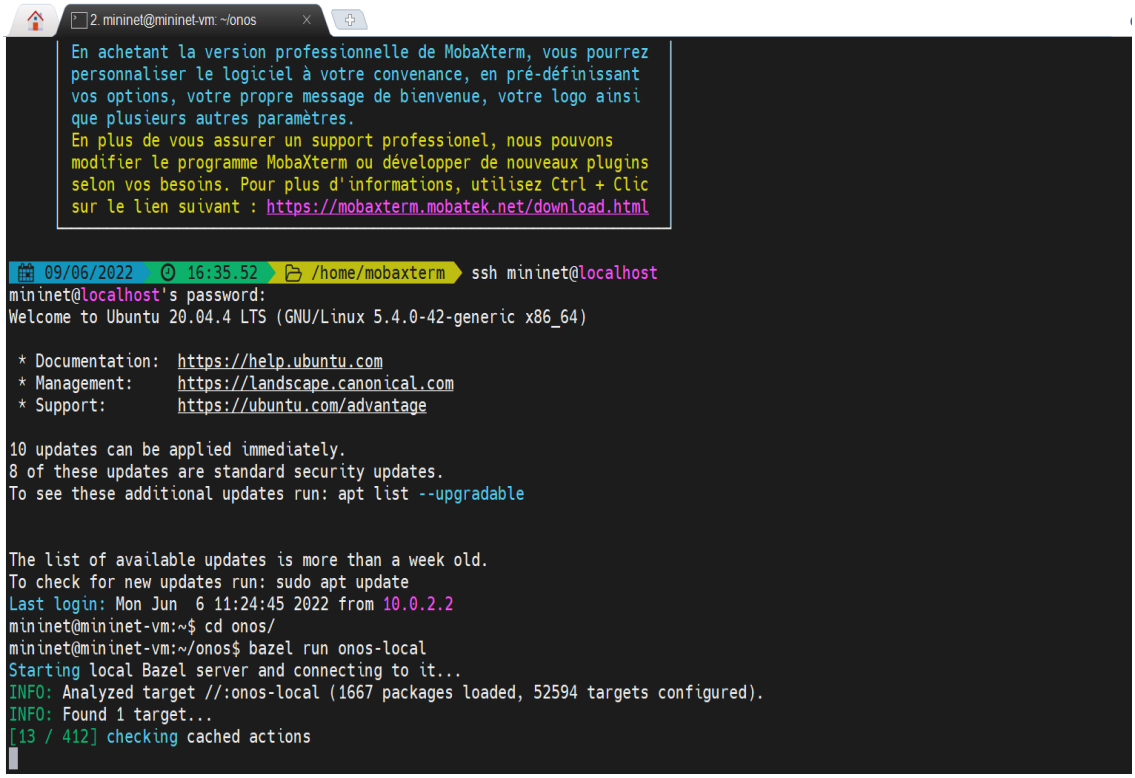

Figure III- 11 : Lancement controlleur ONOS

Dans un autre terminal on a activélé les applications ONOS par les commandes suivantes :

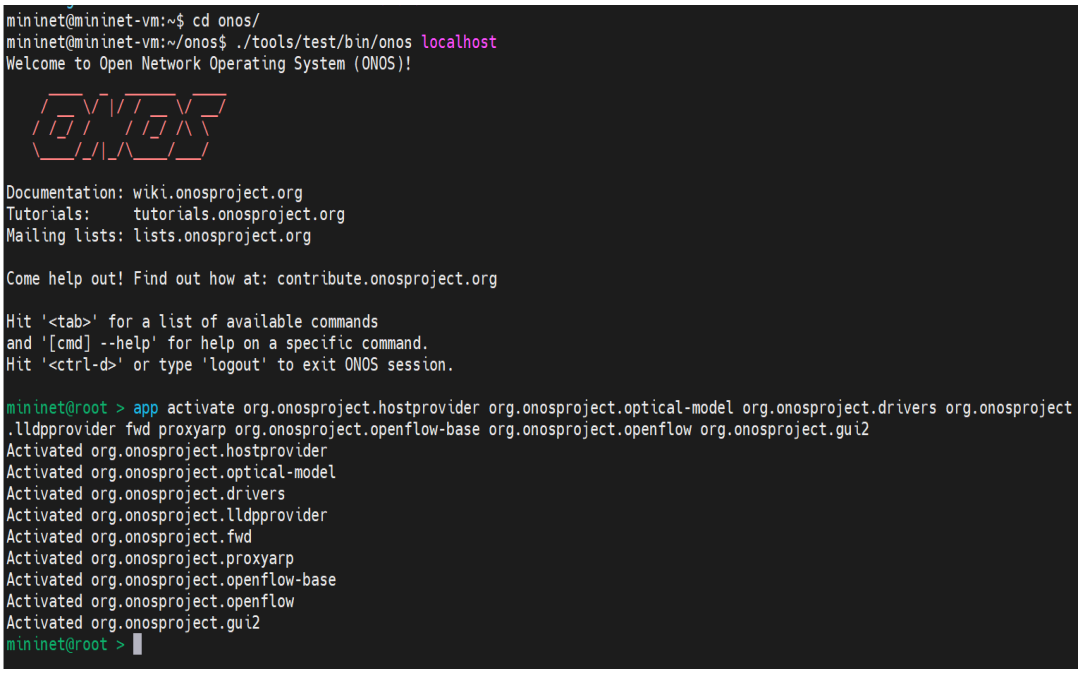

Figure III- 12 : Activation des fonctionnalités ONOS

On ouvre un troisième terminal pour créer une topologie FatTree 4 degrés et on a testé le pingall avec les commandes suivantes :

- ~$ cd GreenSDN/mininet-simulation/
- ~/GreenSDN/mininet-simulation$ sudo python fattree.py <k><traffic|notraffic>
- mininet > pingall

Le fichier python fattree.py permet de créer une topologie fat-tree connecté au contrôleur ONOS. La topologie est k-fattree (4 ou 8 degrés). La figure suivante illustre la génération de topologie avec Mininet :

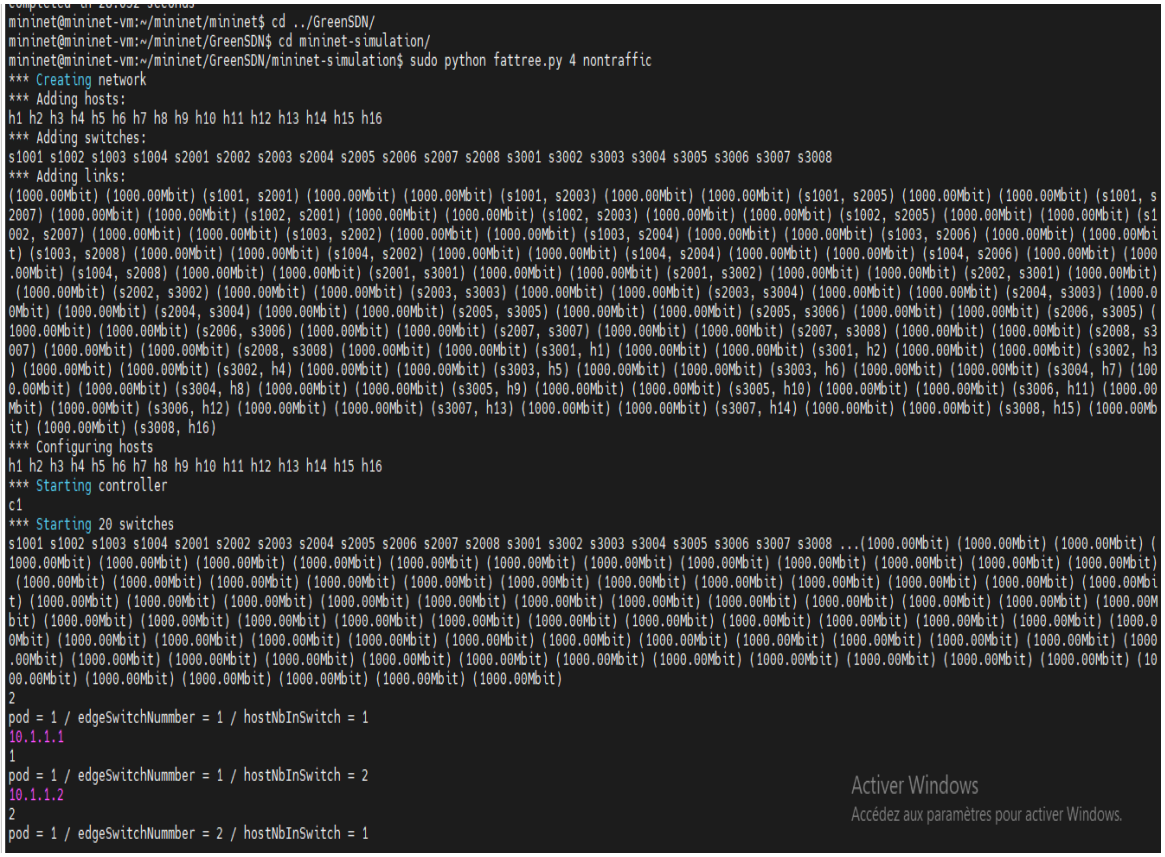

Figure III- 13 : La génération de topologie avec Mininet

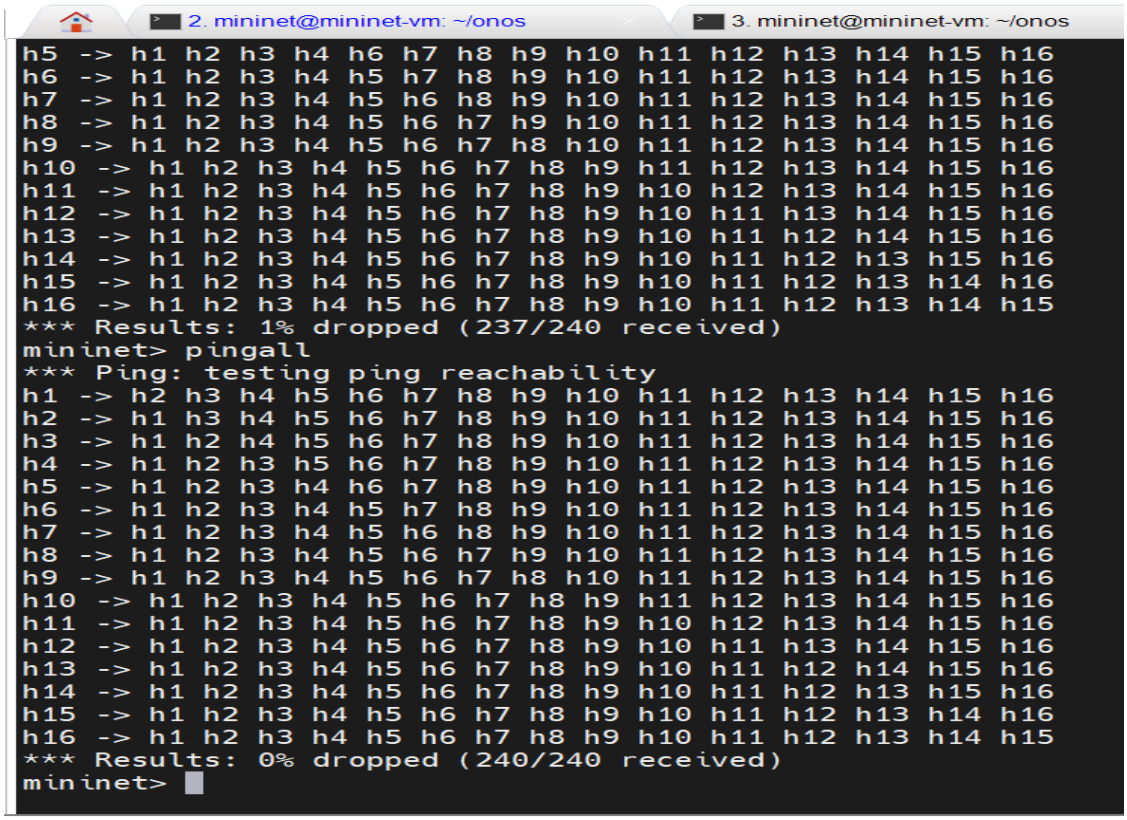

Figure III- 14 : Test pingall avec Mininet

Cette figure illustre les commutateurs détectés par le controleur ONOS :

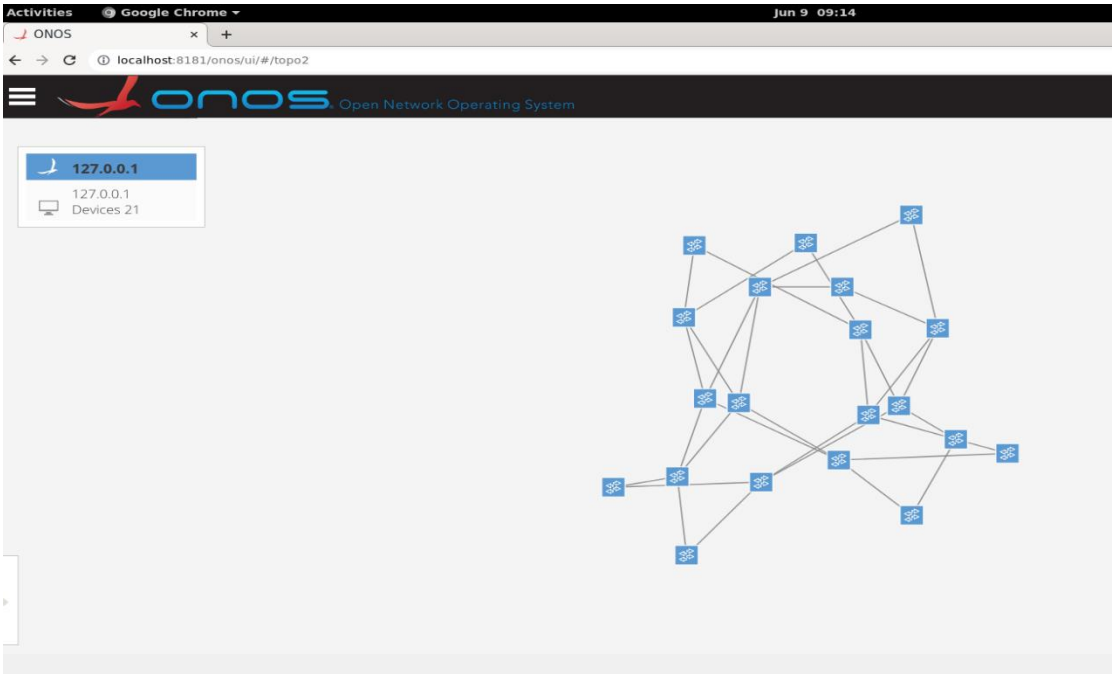

Figure III- 15 : Affichage de la topologie avec ONOS

Ensuite, nous revenons au deuxième terminal pour désactiver l'application ONOS d'acheminement (fwd) à l'aide de la CLI ONOS avec cette commande :

- mininet@root> app deactivate fwd

Après on a créé un chemin par défaut par cette commande :

- ~$ cd ../mininet/GreenSDN/mininet-simulation/app/
- ~$ sudo python defaultpath.py <k>

Le script defaultpath.py contient un algorithme pour créer un chemin par défaut entre chaque hôte du réseau. Par la suite, on a lancée l'algorithme de routage GreenSDN avec la commande :

- ~$ sudo python runElasticTree.py 4

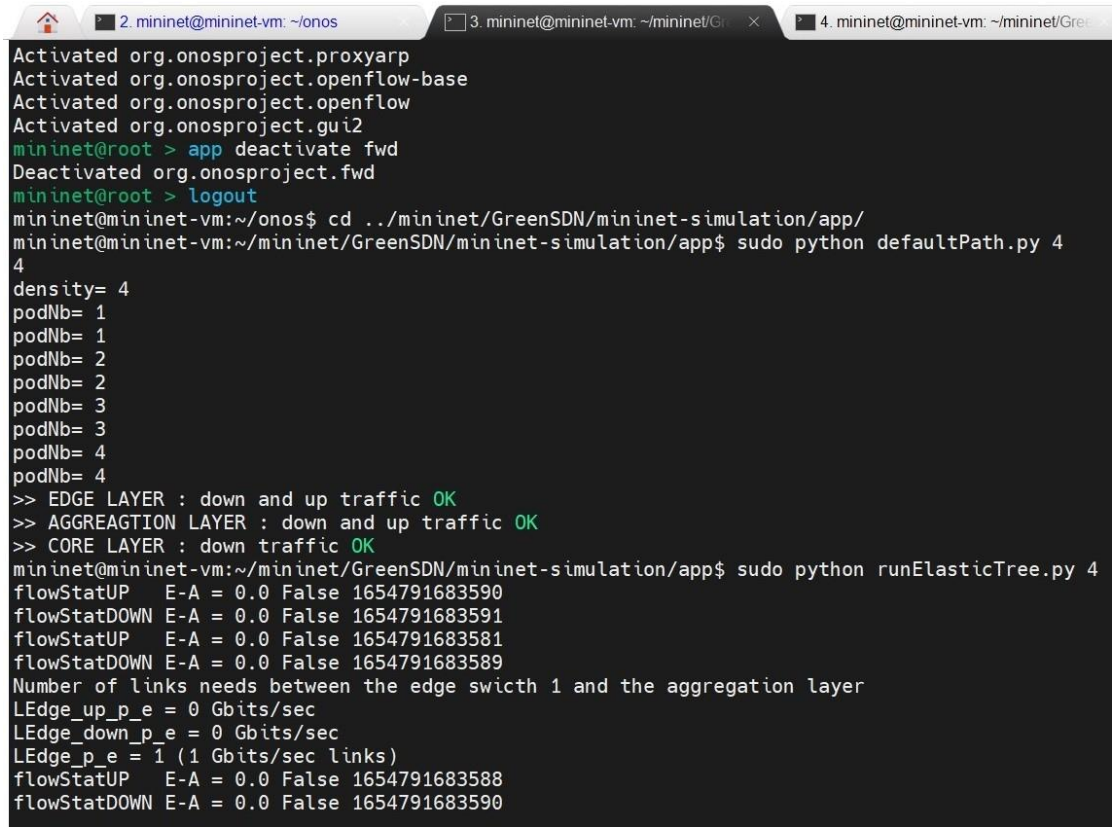

Figure III- 16 : Lancement de l'algorithme de routage GreenSDN

Dans le troisième terminal, on a lancé le trafic généré après lancement de l'algorithme de routage. La figure suivante liste les transmissions actives lors de l'expérimentation :

Figure III- 17 : Lancement des trafics avec ipref

Avant de mettre en œuvre la technologie d'économie d'énergie, nous avions 20 commutateurs allumés. Après l'exécution de l'algorithme d'optimisation, l'algorithme cherche à minimiser le nombre de commutateurs nécessaires pour accommoder le trafic. Nous avons obtenu pour chaque POD, un seul commutateur d'agrégation et un seul commutateur dans la couche core sont nécessaires. Ainsi, le reste est peut-être mise en veille. De plus, le nombre des liens/ports actives est réduit aussi. La figure suivante montre le résultat affiché lors de l'exécution de l'algorithme.

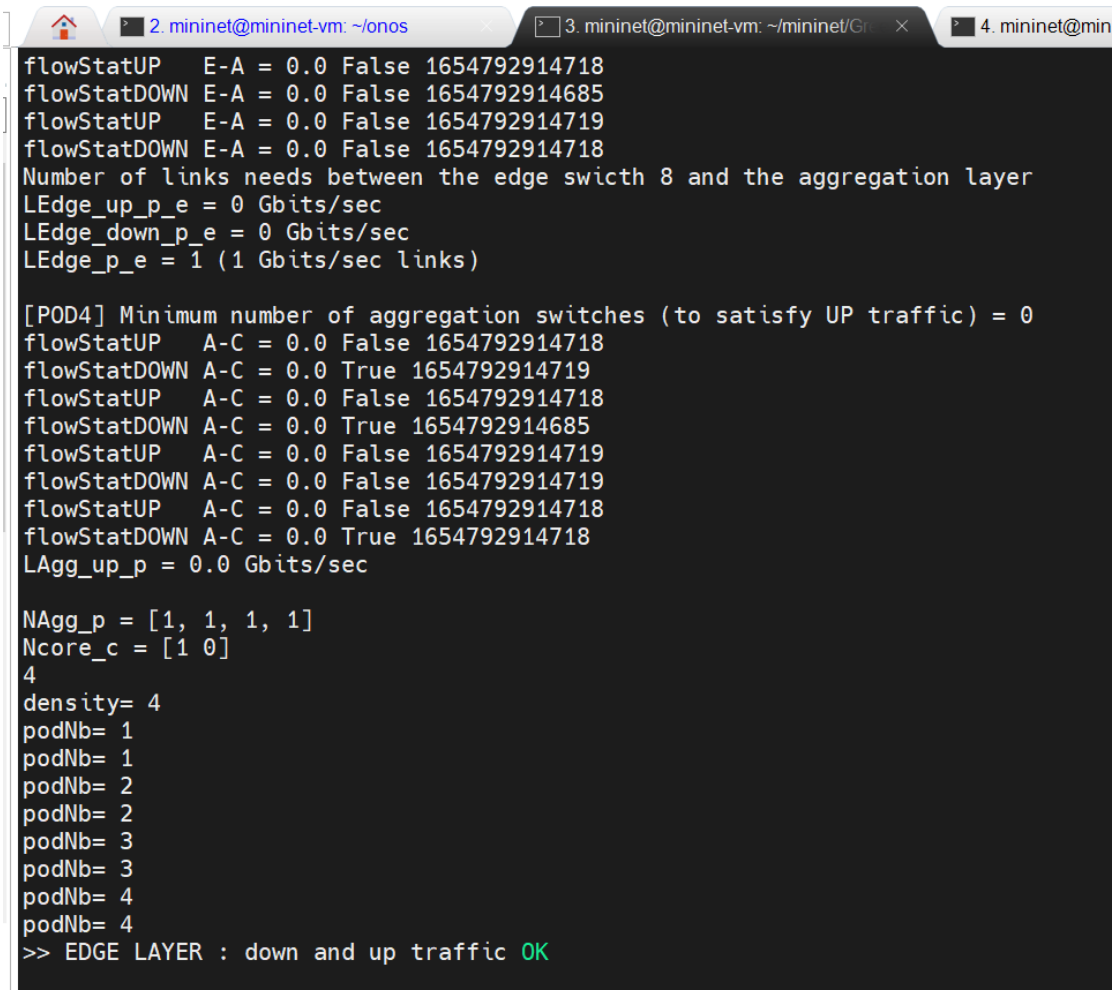

Figure III- 18 : Nombre de commutateurs d'agrégation est core nécessaires après l'exécution de l'algorithme d'optimisation

III.5. Conclusion

Au cours de ce dernier chapitre, nous avons fourni une explication détaillée sur Mininet avec des exemples simples sur les topologies, et on a présenté les outils que nous avons utilisés pour cette étude. Ensuite, nous avons présenté la topologie réseau simulée ainsi qu'une implémentation d'un contrôleur avec une topologie personnalisée.

Les résultats de simulation illustre l'efficacité du routage avec l'économie d'énergie. La technique examinée offre l'avantage d'être basée sur la technologie SDN qui rend l'approche plus flexible et adaptable.

Conclusion générale

Au cours de la dernière décennie, une grande attention a été accordée à l'efficacité énergétique au niveau des réseaux de télécommunication en raison de ses impacts économiques et environnementaux. Cependant, la mise en œuvre de stratégies d'efficacité énergétique sur les réseaux traditionnels est très compliquée par l'intégration verticale du plan de contrôle et de données à chaque dispositif de mise en réseau.

L'émergence du modèle SDN est venue simplifier cette complexité et permettre l'innovation au niveau des réseaux de communication. En effet, la technologie SDN qui découple le plan de contrôle du plan de données permet de programmer le comportement du réseau de manière centralisée à travers des API ouvertes. Grâce à cette nouvelle architecture, les administrateurs peuvent gérer le réseau d'une manière simplifié à partir du plan de contrôle, et peuvent introduire ou éliminer n'importe quel service à travers le plan d'application sans accéder à l'infrastructure physique. C'est bien là que réside le point fort du réseau SDN, désormais il n'évolue plus avec la lourdeur du matériel mais avec la flexibilité logicielle.

Dans ce mémoire, nous avons constaté que la mise en œuvre d'économie d'énergie dans le SDN est plus facile et moins coûteuse que les réseaux traditionnels. Par conséquence, nous avons tenté de répondre au problème d'optimisation de la consommation énergétique des réseaux SDN. Pour y parvenir, nous avons commencé par étudier et analyser les différentes stratégies et techniques présentes dans ce domaine.

Dans ce travail de fin d'étude, nous avons examiné une stratégie spécifique. La stratégie choisie est le routage éco-conscient appliqué aux réseaux SDN. Cette stratégie exploite la vue globale du réseau fournie par le contrôleur SDN afin de réaliser un routage des flux en tenant compte des contraintes d'économie d'énergie. Nous avons pu illustrer son avantage ainsi que son efficacité. Dans des futurs travaux, nous souhaitons explorer les perspectives suivantes :

- Étudier la sécurité au sein du SDN.
- Mener une comparaison empirique entre les différents contrôleurs SDN actuellement disponibles tel que RYU.
- Proposer notre propre contribution dans ce domaine.

Références

[1] Définition TCP/IP (Transmission Control Protocol/Internet Protocol) [Internet]. Disponible sur, https://actualiteinformatique.fr/cloud/definition-tcp-ip-transmission-control-protocol-internet-protocol?fbclid=IwAR0qx_Lhqn01VYGeNb2c6jVUKrxfsaQ8DO2pGf2KITOJJcXSILCTeKSzSy4

[2] Principe du SDN dans une architecture réseau classique [Internet]. [cité 15 janvier 2018]. Disponible sur :https://blogs.univ-poitiers.fr/f-launay/2018/01/15/principe-du-sdn-dans-une-architecture-reseau-classique/?fbclid=IwAR0s674Mm851hb63aDLfqK3VFEfxzOGxMNWgrn0qmZ3ddkTg5-2478H5C1k

[3] Qu'est-ce qu'un réseau informatique ? Définition et exemples [Internet]. [cité 05 Octobre 2020]. Disponible sur : https://www.ionos.fr/digitalguide/serveur/know-how/reseau-informatique-definition/

[4] Michèle Germain. « INTRODUCTION AUX RÉSEAUX ». Disponible sur : https://www.electronique-mixte.fr/wp-content/uploads/2018/08/Formation-Interface-communication-89.pdf

[5] Software Defined Networking : qu'est-ce que le SDN ? [Internet]. [cité 12 juin 2019] Disponible sur : https://www.ionos.fr/digitalguide/serveur/know-how/software-defined-networking/#c192037

[6] Chahed B. Mise en oeuvre des aspects de gestion des réseaux définis par logiciels (réseaux SDN) [Internet]. Université de Montréal ; 2015. Disponible sur : https://publications.polymtl.ca/1924/.

[7] Choukri, Ihssane, Mohammed Ouzzif and KB. Software Defined Networking (SDN) : Etat de L'art. 2019 ; Disponible sur : https://hal.archives-ouvertes.fr/hal-02298874

[8] Jérôme Durand, « Le SDN pour les nuls », Montpellier, JRES 2015. Disponible sur : https://conf-ng.jres.org/2015/document_revision_2132.html?download

[9] Göransson P, Black C. Software Defined Networks A Comprehensive Approach. Mark Roger. Elliot S, éditeur. 2014. Page 12: https://ridhanegara.staff.telkomuniversity.ac.id/files/2017/04/Paul-Goransson-and-Chuck-Black-Auth.-Software-Defined-Networks.-A-Comprehensive-Approach.pdf

[10] Babu R. Dawadi , Danda B. Rawat SRJ. Software Defined IPv6 Network: A New Paradigm for Future Networking. J Inst Eng. 2019;15(2):1-13.

[11] Energy-Aware Routing in Carrier-Grade Ethernet using SDN Approach Rihab Maaloul , Raouia Taktak∗ Lamia Chaari∗ and Bernard Cousin† ∗ Laboratory of Technology and Smart Systems (LT2S/CRNS), University of Sfax, Tunisia †University of Rennes 1, IRISA, France. Disponible sur: https://www.researchgate.net/figure/Traditional-networking-versus-SDN-networking_fig1_324941281

[12] Pradeep Kumar Sharma, S. S. Tyagi. « Improving Security through Software Defined Networking (SDN): AN SDN based Model ». Disponible sur : https://www.ijrte.org/wp-content/uploads/papers/v8i4/D6814118419.pdf?fbclid=IwAR0KeismV1tlPIVdgX4jDP9lE05WmUemyzCveirQ1tmLls_nHEp4fzGi6Kg
[13] Fouad BENAMRANE. « Etude des Performances des Architectures du Plan de Contrôle des Réseaux 'Software-Defined Networks ». Thèse de doctorat en Informatique. Faculté des Sciences, 4 Avenue Ibn Battouta B.P. 1014 RP, Rabat – Maroc 2017. Disponible sur le site : https://www.researchgate.net/profile/Fouad-Benamrane/publication/316620880_Etude_des_Performances_des_Architectures_du_Plan_de_Controle_des_Reseaux_%27Software Defined_Networks%27/links/59084797aca272116d3d0324/Etude-des-Performances-des-Architectures-du-Plan-de-Controle-des-Reseaux-Software-Defined-Networks.pdf.
[14] Thomas Paradis, « Software-Defined Networkin », mémoire de Master, School of Information and Communication Technology KTH Royal Institute of Technology Stockholm, Sweden, le 20 Janvier 2014.
[15] V. Thirupathi, Ch. Sandeep, S. Naresh Kumar, P. Pramod Kumar. « A COMPREHENSIVE REVIEW ON SDN ARCHITECTURE, APPLICATIONS AND MAJOR BENIFITS OF SDN ». Disponible sur : https://www.researchgate.net/profile/Thirupathi-Vadluri-2/publication/339771419_A_COMPREHENSIVE_REVIEW_ON_SDN_ARCHITECTURE_APPLICATIONS_AND_MAJOR_BENIFITS_OF_SDN/links/5e69e18d92851c20f3220ad4/A-COMPREHENSIVE-REVIEW-ON-SDN-ARCHITECTURE-APPLICATIONS-AND-MAJOR-BENIFITS-OF-SDN.pdf
[16] Mr AICHAOUI Anis, Mr AITBELKACEM Yanis. « Etude et implémentation d'une architecture SDN LAN ». PFE de Master en Informatique. UNIVERSITE Mouloud MAMMERI DE TIZI- OUZOU 2018. Disponible sur :https://www.ummto.dz/dspace/bitstream/handle/ummto/6707/AichaouiAnis_AitbelkacemYanis.pdf
[17] Current Trends of Topology Discovery in OpenFlow-based Software Defined Networks Leonardo Ochoa-Aday, Cristina Cervello-Pastor, ´ Member, IEEE, and Adriana Fernandez-Fern ´ andez disponible sur : https://www.researchgate.net/figure/Layered-architecture-of-the-SDN_fig1_311577105
[18] Cox JH, Chung J, Donovan S, Ivey J, Clark RJ, Riley G, et al. Advancing softwaredefined networks: A survey. IEEE Access. 2017;5:25487-526.
[19] Zohaib Latif, Kashif Sharif, Fan Li, Md Monjurul Karim, and Yu Wang « A Comprehensive Survey of Interface Protocols for Software Defined Networks ». 21 Feb 2019. Disponible sur : https://www.researchgate.net/figure/Layered-view-of-SDN-Architecture_fig1_331273642
[20] Advancing Software-Defined Networks: A Survey JACOB H. COX , JR.1 , JOAQUIN CHUNG2 , SEAN DONOVAN2 , JARED IVEY2 , RUSSELL J. CLARK3 , (Member, IEEE), GEORGE RILEY2 , AND HENRY L. OWEN, III2 , (Senior Member, IEEE) , 12 October, 2017.Disponible sur : https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8066287
[21] H. Song. Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane. In Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, ACM, 2013.

[22] M Smith, M Dvorkin YL et al. OpFlex control protocol. IETF. 2014;
[23] G. Bianchi, M. Bonola, A. Capone, and C. Cascone. Openstate: programming platform-independent stateful openflow applications inside the switch. ACM SIGCOMM Computer Communication Review, 44(2):44–51, 2014.
[24] Kreutz D, Ramos FMV, Verissimo PE, Rothenberg CE, Azodolmolky S, Uhlig S. Software-defined networking : A comprehensive survey. Proc IEEE. 2014 ; p 19,17.
[25] S. Vinoski, « Advanced message queuing protocol », IEEE Internet Computing, vol. 10, no. 6, pp. 87–89, Nov. 2006.
[26] K. Phemius, M. Bouet, and J. Leguay, « DISCO : Distributed Multi-domain SDN Controllers », ArXiv e-prints, Aug. 2013.
[27] M. Canini, P. Kuznetsov, D. Levin, and S. Schmid, « Software transactional networking : concurrent and consistent policy composition », in Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, ser. HotSDN '13. New York, NY, USA : ACM, 2013, pp. 1–6
[28] A. Ghodsi, "Distributed k-ary system : Algorithms for distributed hash tables," Ph.D. dissertation, KTH-Royal Institute of Technology, 2006.
[29] F. A. Botelho, F. M. V. Ramos, D. Kreutz, and A. N. Bessani, « On the feasibility of a consistent and fault-tolerant data store for SDNs », in Proceedings of the 2013 Second European Workshop on Software Defined Networks, ser. EWSDN '13. Washington, DC, USA : IEEE Computer Society, 2013, pp. 38–43.
[30] F. Botelho, A. Bessani, F. M. V. Ramos, and P. Ferreira, « On the design of practical fault-tolerant SDN controllers », in Third European Workshop on Software Defined Networks, 2014, pp.
[31] Shaghaghi, Arash MAK et al. « Software-Defined Network (SDN) Data Plane Security: Issues, Solutions, and Future Directions». Clust Comput J. 2018;24.
[32] B. Salisbury. « The northbound API- a big little problem 2012».
[33] R. Chua, «OpenFlow northbound API : A new olympic sport». 2012. [Online]. Available: http://www.sdncentral.com/sdn-blog/openflow-northbound-api-olympics/2012/07/
[34] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford,A. Story, and D. Walker, « Frenetic: a network programming language » SIGPLAN Not., 2011.
[35] A. Voellmy and P. Hudak, « Nettle: taking the sting out of programming network routers », in Proceedings of the 13th international conference on Practical aspects of declarative languages, ser. PADL'11. Berlin, Heidelberg : Springer-Verlag, 2011, pp. 235–249.
[36] C. Monsanto, N. Foster, R. Harrison, and D. Walker, « A compiler and run-time system for network programming languages," SIGPLAN Not., vol. 47, no. 1, pp. 217–230, Jan. 2012.
[37] A. Voellmy, H. Kim, and N. Feamster, « Procera : a language for high-level reactive network control », in Proceedings of the first workshop on Hot topics in software defined networks, ser. HotSDN '12. New York, NY, USA : ACM, 2012, pp. 43–48.
[38] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, « Modular SDN Programming with Pyretic ». USENIX ; login, vol. 38, no. 5, October 2013.
[39] C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen,C . Schlesinger, and D. Walker, « NetKAT: Semantic foundations for networks », SIGPLAN Not., vol. 49, no. 1, pp. 113–126, Jan. 2014.

[40] Lei Liu, Takehiro Tsuritani, Itsuro Morita, Hongxiang Guo, and Jian Wu. « Experimental validation and performance evaluation of OpenFlow-based wavelength path control in transparent optical networks ». Décembre 2011. Disponible sur : https://www.researchgate.net/profile/Jian-Wu-26/publication/221774501_Experimental_validation_and_performance_evaluation_of_OpenFlow-based_wavelength_path_control_in_transparent_optical_networks/links/54d3286d0cf28e069727a0ad/Experimental-validation-and-performance-evaluation-of-OpenFlow-based-wavelength-path-control-in-transparent-optical-networks.pdf?origin=publication_detail .
[41] « Le Protocole OpenFlow dans l'Architecture SDN (Software Defined Network) ». Disponible sur: https://docplayer.fr/42864749-Le-protocole-openflow-dans-l-architecture-sdn-software-defined-network.html.
[42] « OpenFlow Switch Specification ». dispobile sur : https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf, October 2013. Rev. 1.4.0.
[43] Ching-Hao, Chang and Dr. Ying-Dar Lin. « OpenFlow Version Roadmap ». September 11, 2015. Disponible sur : http://speed.cis.nctu.edu.tw/~ydlin/miscpub/indep_frank.pdf.
[44] Maxence Tury. « Les risques d'OpenFlow et du SDN ». disponible sur : https://www.ssi.gouv.fr/uploads/2015/06/SSTIC2015-Article-risques_openflow_et_sdn-tury.pdf.
[45] FEI HU. « Network Innovation through OpenFlow and SDN ». 2014.
[46] Azodolmolky S. Software Defined Networking with OpenFlow [Internet]. Packt Publishing Ltd. 2013. 153 p. Disponible sur: https://speetis.fei.tuke.sk/KomunikacnaTechnika1/prednasky/7_11_2016/kniha_sietovanie.pdf
[47] Open Networking Foundation, (2014). OpenFlow Switch Specification version 1.5.0. Consulté sur : http://www.opennetworking.org/wpcontent/uploads/2014/10/openflow-switch-v1.5.0.noipr.pdf
[48] PFE « SAFSAF AHMED NAZIM & CHAIERE REDHOUANE ».« COMPARAISON DES PERFORMANCES DES CONTROLEURS OPENFLOW ». SPÉCIALITÉ RÉSEAU & TÉLÉCOMMUNICATION. 2016 – 2017. Disponible sur : http://di.univ-blida.dz:8080/jspui/bitstream/123456789/2616/1/SAFSAF%20ahmed%20nazim%20et%20CHAIERE%20Redhouane.pdf.
[49] Karim Idoudi, «IMPLÉMENTATION D'UN PLAN DE CONTRÔLE UNIFIÉ POUR LES RÉSEAUX MULTICOUCHES IP / DWDM», Université du québec Montréal, 2014. disponible sur: https://archipel.uqam.ca/7182/1/M13669.pdf
[50] Dominik Klein, Michael Jarschel. « An OpenFlow Extension for the OMNeT++ INET Framework ». 23 December 2013. Disponible sur : https://www.researchgate.net/profile/Michael-Jarschel/publication/256471159_An_OpenFlow_extension_for_the_OMNeT_INET_framework/links/0046352b84447b3c34000000/An-OpenFlow-extension-for-the-OMNeT-INET-framework.pdf?origin=publication_detail.

[51] MUKADI NTUMBA Chaco « MISE EN PLACE D'UNE SOLUTION SDN POUR LA GESTION DU RESEAU : « Application de Gestion de Load-Balancing ». Disponible sur : https://docplayer.fr/162916975-Universite-de-kinshasa.html
[52] Ilyas Bambrik « Chapitre X: Software Defined Network ». Disponible sur: https://elearn.univ-tlemcen.dz/pluginfile.php/134106/mod_resource/content/2/Chapitre%20X%20SDN.pdf
[53] « Challenges Lying in the Wait of SDN ». April 15, 2015. Disponible sur : https://www.nojitter.com/4-challenges-lying-wait-sdn.
[54] Ongaro F. « ENHANCING QUALITY OF SERVICE IN SOFTWARE-DEFINED NETWORKS ». Diss Alma Mater Stud Bol. 2014. Disponible sur : https://amslaurea.unibo.it/7356/1/francesco_ongaro_tesi.pdf.
[55] Beakal Gizachew Assefa, Öznur Özkasap . « A Survey of Energy Efficiency in SDN: Software-based Methods and Optimization Models ». Department of Computer Engineering, Koç University, Istanbul, Turkey.
[56] Brandon, Heller, Srinivasan Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, PuneetSharma, Sujata Banerjee et Nick McKeown. 2010. « ElasticTree : Saving Energy inData Center Networks ». In NSDI, vol. 10, p. 249-264.
[57] Xiaodong, Wang, Yanjun Yao, Xiaorui Wang, Kefa Lu et Qing Cao. 2012. « Carpo :Correlation-aware power optimization in data center networks ». In INFOCOM, p. 1125-1133.
[58] Luo, J., Zhang, S., yin, L., Guo, Y., 2017. Dynamic flow scheduling for power optimization of data center networks. In : Fifth International Conference on Advanced Cloud and Big Data (CBD), pp. 57–62.
[59] Hiroki, Shirayanagi, Hiroshi Yamada et Kono Kenji. 2013. « Honeyguide : A vm migrationaware network topology for saving energy consumption in data center networks ». IEICE TRANSACTIONS on Information and Systems, vol. 96, n◦ 9, p. 2055-2064.
[60] Shao-Heng, Wang, Patrick P-W. Huang, Charles H-P. Wen et Li-Chun Wang. 2014. « EQVMP : Energy-efficient with QoS-aware VM Placement algorithm ». In The International Conference on Information Networking(ICOIN), p. 220-225.
[61] Zheng, K., Zheng, W., Li, L., & Wang, X. (2017). PowerNetS: Coordinating Data Center Network With Servers and Cooling for Power Optimization. IEEE Transactions on Network and Service Management, 14, 661-675.
[62] Yossi, CKanizo, David Hay, et Isaac Keslassy. 2013. « Palette : Distributing tables in softwaredefined networks ». In INFOCOM, 2013 Proceedings IEEE, p. 545-549.
[63] Leng, B., Huang, L., Qiao, C., Xu, H., Wang, X., 2017. Ftrs: a mechanism for reducing flow table entries in software defined networks. Comput. Network. 122, 1–15.
[64] Meiners, Chad R., Alex X. Liu, et Eric Torng. 2012. « Bit weaving : A non-prefix approachto compressing packet classifiers in TCAMs ». IEEE/ACM Transactions on Networking (ToN), vol. 20, n◦ 2, p. 488-500.
[65] Kannan, K., Banerjee, S., 2012. Compact TCAM: flow entry compaction in TCAM for power aware SDN. IEEE/ACM Trans. Netw. 20 (2), 488–500.
[66] Karamjeet Kaur, Japinder Singh and Navtej Singh Ghumman. « Mininet as Software Defined NetworkingTesting Platform ». Disponible sur :https://www.researchgate.net/profile/Japinder-Singh/publication/287216738_Mininet_as_Software_Defined_Networking_Testing_Platform/

links/5674461808aebcdda0de21cc/Mininet-as-Software-Defined-Networking-Testing-Platform.pdf?origin=publication_detail.
[67] What Is Open vSwitch? [Internet]. Disponible sur : https://docs.openvswitch.org/en/latest/intro/what-is-ovs/.
[68] POX Manual Current documentation [Internet]. [cité 18 août 2020]. Disponible sur: https://noxrepo.github.io/pox-doc/html/.
[69] What is Oracle VM VirtualBox? [Internet]. Disponible sur : https://geek-university.com/what-is-oracle-vm-virtualbox/.
[70] What is MobaXterm? [Internet]. Disponible sur :https://mobaxterm.mobatek.net/.
[71] Sukhveer Kaur, Japinder Singhand Navtej Singh Ghumman. « Network Programmability UsingPOX Controller ». Disponible sur : https://www.researchgate.net/profile/Japinder-Singh/publication/287216671_Network_Programmability_Using_POX_Controller/links/567447b508aebcdda0de21e6/Network-Programmability-Using-POX-Controller.pdf?origin=publication_detail.
[72] HaeederMunther Noman, Mahdi Nsaif Jasim. « POX Controller and Open Flow PerformanceEvaluation in Software Defined Networks (SDN)Using Mininet Emulator ». Disponible sur :https://iopscience.iop.org/article/10.1088/1757-899X/881/1/012102/pdf.
[73] What is ONOS (Open Network Operating System) [Internet]. Disponible sur: https://www.techtarget.com/searchnetworking/definition/ONOS-Open-Network-Operating-System
[74] Python : qu'est-ce que c'est ? [Internet].Disponible sur : https://www.futura-sciences.com/tech/definitions/informatique-python-19349/.
[75] Python : définition et utilisation de ce langage informatique. [Internet]. Disponible sur : https://www.journaldunet.fr/web-tech/dictionnaire-du-webmastering/1445304-python-definition-et-utilisation-de-ce-langage-informatique/.
[76] MininetOverview [Internet]. [cité 18 août 2020] . Disponible sur: http://mininet.org/overview/.
[77] MININET [Internet]. [cité 18 août 2020]. Disponible sur: https://www.opennetworking.org/mininet/.

Résumé

La croissance exponentielle des utilisateurs du réseau et leurs besoins de communication ont conduit à une augmentation substantielle de la consommation d'énergie dans l'infrastructure du réseau. Un nouveau paradigme de réseautage appelé Software Defined Networking (SDN) a récemment vu le jour, dans lequel le plan de données, responsable de la transmission des paquets est séparé du plan de contrôle responsable de la prise de décision de routage. La technologie SDN a déclenché un changement radical à long terme dans la conception des réseaux en offrant la programmabilité des équipements réseau et permet la redirection rapide des flux.

Dans ce mémoire, nous étudions l'effet d'une approche de routage basée sur l'économie d'énergie sur la performance réseau à l'aide d'Openflow et le SDN. Cette méthode prend en considération le nombre de commutateurs actifs redondants et leur mise en veille, afin de réduire la consommation d'énergie dans le réseau. L'environnement de simulation de cette approche est testé grâce à l'émulateur Mininet.

Mots clefs : SDN, Contrôleur, OpenFlow, Optimisation, L'économie d'énergie.

Abstract

The exponential growth of network users and their communication needs has led to a substantial increase of energy consumption in the network infrastructure. Consequently, a new networking paradigm called Software Defined Networking (SDN) has recently emerged, in which the data plane, responsible for packet forwarding is separated from the control plane responsible for routing decision making.SDN technology has triggered a radical change over the long term in network design by offering the programmability of network equipment and allowing the rapid redirection of flows.

In this thesis, we study the effect of a power-saving routing approach on network performance using Openflow and SDN. This method takes into consideration the number of redundant active switches and puts them on standby, in order to reduce the energy consumption of the network. The simulation environment of this approach is tested using the Mininet emulator.

Keywords: SDN, Controller, OpenFlow, Optimization, Energy saving.

ملخص

أدى النمو المتسارع لمستخدمي الشبكة واحتياجات اتصالاتهم إلى زيادة كبيرة في استهلاك الطاقة في البنية التحتية للشبكة.
ظهر نموذج جديد للشبكات يسمى الشبكات المعرفة بالبرمجيات مؤخرًا، حيث يتم فصل مستوى البيانات المسؤول عن إعادة توجيه الحزمة عن مستوى التحكم المسؤول عن توجيه اتخاذ القرار.

أحدثت تقنية " الشبكات المعرفة بالبرمجيات " تغييرًا جذريًا طويل المدى في تصميم الشبكة من خلال توفير إمكانية برمجة معدات الشبكة والسماح بإعادة التوجيه السريع للتدفقات. سرعان ما تبنى السوق " الشبكات المعرفة بالبرمجيات " كمجموعة من الحلول / البنى التي تسمح بإزالة الحدود القائمة بين عوالم التطبيقات والشبكة، وعلى وجه الخصوص لأنها تفتح فرصًا جديدة واعدة لكفاءة الطاقة

في هذه الأطروحة، ندرس تأثير نهج التوجيه الموفر للطاقة على أداء الشبكة باستخدام " تدفق مفتوح " و " الشبكات المعرفة بالبرمجيات ". تأخذ هذه الطريقة في عين الاعتبار عدد المفاتيح النشطة الزائدة عن الحاجة وتضعها في وضع الاستعداد، من أجل تقليل استهلاك الطاقة للشبكة. يتم تنفيذ بيئة المحاكاة الخاصة بهذا النهج بفضل المحاكي " مينينات".

كلمات مفتاحية: الشبكات المعرفة بالبرمجيات، تحكم، تدفق مفتوح، التحسين، توفير الطاقة.