https://www.overleaf.com/project/5e4c388313110d0001e8c697https://www.overleaf.com/project/5e4c

République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Electronique et d'Informatique

Département d'Informatique

Projet de fin d'études pour l'obtention du diplôme de

Master

Spécialité

Réseaux et Systèmes Distribués

Gestion du trafic dans les Réseaux SDNs

Encadré par :
Mme N. BOUZIANE
Mme Z. DOUKHA

Présenté par :
ACHAIBOU Radia Amina
HAMENNICHE Amira

Devant le jury composé de :

Mr D.BAHLOUL Président
Mr M.AHMED NACER Membre

Binôme N°105/2020

REMERCIEMENTS

Après avoir rendu grâce à Dieu le tout Puissant et le Miséricordieux, nous tenons à exprimer nos respectueux remerciement à nos encadreurs Mme Z. DOUKHA & Mme N. BOUZIANE pour l'aide qu'elles ont bien voulu nous accorder tout le long de notre travail, pour leurs précieux conseils, orientations bienveillantes, leurs infatigable dévouement.

Nous remercions les membres du jury Mr BAHLOUL et Mr AHMED NACER pour l'honneur d'avoir voulu examiner et évaluer ce travail et nous tenons à vous exprimer tout notre respect et notre estime.

Nous tenons à remercier nos parents pour leurs soutien moral et physique et leurs inéluctable patience et c'est grâce à eux que ce projet a pu voir le jour. Enfin, nous remercions tous ceux qui ont contribué de près ou de loin à la réalisation de ce mémoire.

Amira & Radia

DÉDICACE

A MA TRÈS CHÈRE MAMAN : YOUNÈS LILA

Autant de phrases aussi expressives soient-elles ne sauraient montrer le degré d'amour et d'affection que j'éprouve pour toi. Tu m'as comblé avec ta tendresse et affection tout au long de mon parcours. Tu n'as cessé de me soutenir et de m'encourager durant toutes les années de mes études, tu as toujours été présente à mes cotés pour me consoler quand il fallait. En ce jour mémorable, pour moi ainsi que pour toi, reçoit ce travail en signe de ma vive reconnaissance et mon profond estime. Puisse le tout puissant te donner santé, bonheur et longue vie afin que je puisse te combler à mon tour.

À mon Binôme Amira pour son soutien et ses efforts, je te souhaite un avenir radieux.

Radia

DÉDICACE

C'est avec profonde gratitude et sincères mots, que je dédie ce modeste travail de fin d'étude

À mes chers parents qui ont sacrifié leur vie pour notre réussite et nous ont éclairé le chemin par leurs conseils judicieux. J'espère qu'un jour, je pourrais leurs rendre un peu de ce qu'ils ont fait pour moi. Qu'ALLAH vous accorde santé et langue vie.

À mes frères Abderahim, Nabil et Abd El Djalil pour tout l'amour qu'ils m'apportent.

À mes chères cousines Amel, Asma et Hanen pour leurs soutiens. Je vous souhaite une vie pleine de bonheur et de succès et que Dieu, le tout puissant, vous protège et vous garde.

À mes chères copines Keltoum, Safa, Feriel et Bouchra pour leurs encouragements, je vous souhaite plein de succès.

À mon Binôme Radia pour ses efforts, je te souhaite un avenir plein de bonheur et de succès.

À toutes ma famille et à toutes personnes qui me sont très chères et qui m'ont aidé de près ou de loin.

Amira

RÉSUMÉ

Le SDN (*Software Defined Networking*) est un paradigme de mise en réseau émergent qui sépare le plan de contrôle du réseau du plan de données avec la promesse d'améliorer considérablement l'utilisation des ressources du réseau, de simplifier la gestion du réseau, de réduire les coûts d'exploitation et de promouvoir l'innovation et l'évolution. Bien que les techniques d'ingénierie du trafic aient été largement exploitées dans le passé et dans les réseaux de données actuels, tels que les réseaux ATM et les réseaux IP / MPLS, pour optimiser les performances des réseaux de communication en analysant, prédisant et régulant dynamiquement le comportement des données transmises, les caractéristiques uniques du SDN nécessitent de nouvelles techniques d'ingénierie du trafic qui exploitent la vue du réseau globale, l'état et les caractéristiques de ce dernier et les modèles de flux disponibles pour un meilleur contrôle et une meilleure gestion du trafic.
L'objectif de ce projet est de proposer une stratégie de gestion du trafic dans un réseau SDN en prenant en considération la nature dynamique des caractéristiques réseau et l'optimisation dans l'utilisation des ressources réseau.

Mots clés : SDN, plan de contrôle, plan de données, gestion du trafic.

ABSTRACT

Software Defined Networking (SDN) is an emerging networking paradigm that separates the network control plane from the data plane with the promise to dramatically improve network resource utilization, simplify network management, reduce operating cost, and promote innovation and evolution. Although traffic engineering techniques have been widely exploited in the past and current data networks, such as ATM networks and IP/MPLS networks, to optimize the performance of communications networks by dynamically analyzing, predicting, and regulating the behavior of the transmitted data, the unique features of SDN require new traffic engineering techniques that exploit the global network view, the state and characteristics of the latter, and the available flow models for better control and management of traffic.
The objective of this project is to propose a strategy for managing traffic in an SDN network taking into consideration the dynamic nature of network characteristics and optimization in the use of network resources.

Keywords : Software-defined networking, Control plane, Data plane, Traffic management.

Table des matières

Liste des Figures

Liste des tableaux

Liste des abréviations

ADMCF : Adaptive Dynamic Multi-path Computation Framework

API : Application Programming Interface

ARP : Adress Résolution Protocol

AS : Autonoms System

ATM : Asynchronous Transfer Mode

BFD : Bidirectional forwarding detection

Bwm-ng : Bandwidth Monitor NG

CLI : Command Line Interface

ECMP : Equal-Cost Multipath

ETSI : uropean Telecommunications Standards Institute

FPTAS : Fully Polynomial Time Approximation Scheme

HTTP : HyperText Transfer Protocol

ICMP : Internet Control Message Protocol

IDC : International Data Corporation

IETF : Internet Engineering Task Force

IGP : Interior Gateway Protocol

IP : Internet Protocol

IPSEC : Internet Protocol Security

IS – IS : Intermediate System to Intermediate System

LAN : Local Area Network

LLDP : Link Layer Discovery Protocol

LLDPDU : Link Layer Discovery Protocol Data Unit

LSP : Label Switching Path

MPLS : Multi-Protocol Label Switching

NAT : Network Address Translation

NBI : Northbound Interface

Netconf : Network Configuration Protocol

NFV : Network Function Virtualisation

NIB : Network Information Base

ONF : Open Networking Foundation

OVS : Open vSwitch

OVSDB : Open Virtual Switch Data Base

QoS : Quality of Service

RESTful API : Representational state transfer

RSVP : Resource reSerVation Protocol

RTT : Round-trip time

SBI : Southbound Interface

SDN : Software-Defined Network

SDWN : Software Defined Wireless Network

SNMP : Simple Network Management Protocol

TCAM : Ternary Content Addressable Memory

TCP/IP : Transmission Control Protocol/Internet Protocol

TLS : Transport Layer Security

UDP : User Datagram Protocol

VLAN : Virtual Local Area Network

VLAN : Virtual Local Area Network

VM : Virtual Machine

VSDN : Video over Software-Defined Network

WAN : Wide Area Network

YANG : Yet Another Next Generation

Introduction Générale

Le monde des réseaux est en constante évolution de par les technologies de transmissions utilisées (Fibre, 4G, 5G...), les contenus partagés, le volume des données ou encore la maniérée avec laquelle les utilisateurs accèdent aux services réseaux (accès distant, cloud, stockage ...). Ces aspects obligent les infrastructures et les protocoles à s'adapter et à maintenir un niveau décent des prestations.

C'est dans ce contexte que la technologie SDN a vu le jour. Visant à consolider programmation et réseau, elle permet d'agir sur une infrastructure physique comme une entité logique et de faciliter ainsi la gestion des réseaux en permettant un haut degré de mise à l'échelle. Cependant, avec le développement rapide de SDN, le problème de gestion du trafic réseau basé sur le contrôle centralisé offert par cette technologie est devenu un sujet de recherche d'actualité. En effet, il est essentiel de concevoir un mécanisme de gestion du trafic basé sur l'état du réseau en temps réel et les caractéristiques du trafic.

Dans le cadre de notre projet de fin d'études sanctionnant un cursus de deux années de Master en réseaux et systèmes distribués, il nous a été donné de travailler sur la gestion du trafic dans les réseaux SDN *Software-Defined Network*.

Le présent document pésente le travail effectué dans le cadre de ce projet.

Dans l'optique de réaliser cela, nous proposons un mécanisme de gestion du trafic au sein d'une architecture SDN, tirant parti du contrôle centralisé de SDN pour surveiller le comportement et obtenir les statistiques de flux réseau. Par la suite, nous utilisons une stratégie de routage permettant l'acheminement du flux réseau de façon optimale. La gestion du trafic est étroitement intégrée à la planification du trafic pour améliorer et équilibrer la charge des liaisons afin d'éviter la congestion et la perte de données.

Notre mémoire retrace les différentes étapes qui ont donné lieu à cette solution. Des recherches bibliographiques ont été menées sur l'approche SDN et la gestion du trafic. Ainsi, les chapitres 1 et 2 serviront à présenter les résultats de ces recherches et à définir les notions théoriques à connaitre.

Dans le chapitre 3, nous énoncerons les spécificités de la solution proposée ainsi que sa conception en détail. Cette partie sera suivie par un guide d'implémentation visant à détailler le déroulement de la mise en place de la solution dans un environnement virtualisé. Nous concluons le chapitre 4 par une partie dans laquelle nous mettrons en avant la phase d'expérimentation.

Nous terminons ce mémoire avec une conclusion générale dans laquelle nous décrivons les principaux objectifs que nous avons atteints ainsi que les perspectives ouvertes par ce travail, suivis des annexes et des références bibliographiques utilisées.

Chapitre 1

Généralités sur le SDN

1.1 Introduction

Avec le progrès des technologies de communication (cloud, big data, multimedia, IoT ...) la gestion des réseaux à grande échelle devient de plus en plus complexe et couteuse mais aussi la voracité de ces nouvelles technologies en terme de bande passante et leur dynamisme font qu'il est nécessaire d'introduire de nouveaux paradigmes dans le monde des réseaux, de sorte que l'allocation de bande passante, le routage, la mise à l'échelle et la configuration des réseaux deviennent moins contraignants en terme de main d'oeuvre et répondent aux exigences du marché.

C'est dans ce contexte que la technologie SDN est née, développée par l'*Open Networking Foundation*(ONF), créée en 2011. Cette technologie procure un haut niveau d'abstraction en permettant de gérer et configurer de manière centralisé différents services réseau et de voir toute une infrastructure comme une seule entité programmable.

Dans ce chapitre, nous présenterons le concept et l'architecture du SDN ainsi que le protocole OpenFlow et les différents modèles de programmation du réseau afin de mieux interagir avec les applications. Nous expliciterons le rôle du contrôleur au sein de cette architecture ainsi que l'intérêt qu'a suscité SDN auprès des chercheurs et des industriels dans le domaine des réseaux.

1.2 Définition

SDN signifie littéralement *Software-Defined Network*, lancé par L'ONF (consortium d'entreprises à but non lucratif fondé en 2011 pour promouvoir SDN et normaliser ses protocoles) ; c'est un réseau défini par application qui permet l'innovation réseau. Il est basé sur quatre principes fondamentaux [17].

1. Les plans de contrôle et d'acheminement du réseau sont clairement découplés.
2. Les décisions d'acheminement sont basées sur les flux plutôt que sur les destinations.
3. La logique d'acheminement est extraite du matériel vers une couche logicielle programmable.
4. Un élément appelé contrôleur, est introduit pour coordonner les décisions d'acheminement à l'échelle du réseau.

1.3 Architecture

La définition du SDN proposée dans les RFC 7426[14] et RFC 7419[13] définit 3 couches : couche applicative, couche de contrôle et couche de données. La communication entre ses couches se fait via le plan de contrôle qui les relie à l'aide des APIs *Application Programming Interface Sothbound et Northbound*.Figure 1.1.

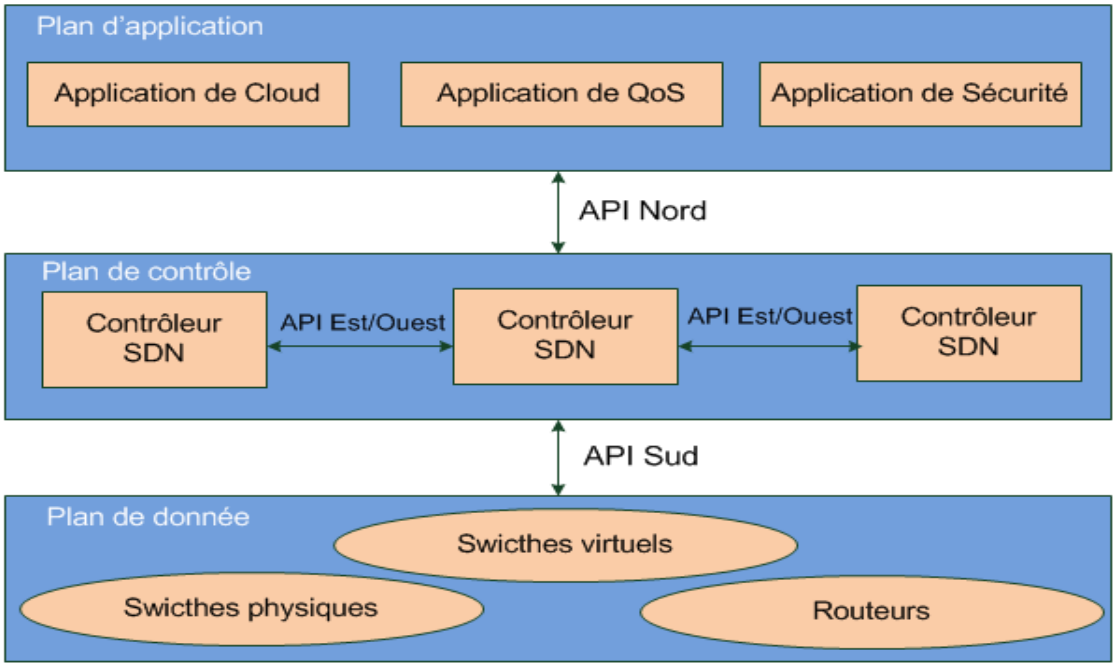


FIGURE 1.1 – Architecture du SDN [30]

1.3.1 La couche applicative

C'est la couche la plus haute dans l'architecture, représente les programmes qui communiquent les comportements et les ressources nécessaires au contrôleur et définissent la logique de contrôle du réseau tel que la définition d'une politique de routage, la façons de gérer la qualité de service (QoS) ...etc. Ces programmes sont construits moyennant une interface de programmation *Northbound* et offerte par le controleur.

1.3.2 La couche de contrôle

Située entre le plan de données et le plan d'application, elle est responsable de la gestion et de la configuration des ressources de la couche de données via des interfaces *Sorthbound*. La couche de contrôle est constituée principalement d'un contrôleur SDN centralisé qui permet d'héberger la logique de contrôle et d'administration du réseau.
Ce contrôleur met en œuvre cette logique en accédant au plan de données à travers une interface *Sorthbound*.

Le plan de contrôle fournit également des services aux applications qui utilisent le plan de données et les données collectées par le plan de contrôle pour fournir d'autres fonctions au sein du réseau. Parmi les applications, citons le provisionnement du réseau, la découverte avancée de topologie de réseau, la réservation de chemin d'accès et le pare-feu. Le rôle de cette couche est

le point de contrôle pour relayer les informations au matériel ci-dessous via les API *Southbound* et aux applications ci-dessus via les API *Northbound*.

1.3.3 Interface *Southbound* (SBI)

Ce sont des interfaces de communication permettant au contrôleur d'interagir avec les switches/ routeurs du plan de données. Le protocole le plus utilisé et le plus déployé comme interface *Southbound* est OpenFlow. Ce protocole est traité plus loin dans ce chapitre [37].

1.3.4 Interface *Northbound* (NBI)

Les interfaces Nord servent à programmer les éléments de transmission en exploitant l'abstraction du réseau fournie par le plan de contrôle. Elles sont considérées davantage comme des APIs que comme protocole de programmation et de gestion du réseau. Ces APIs sont prises en charge par le contrôleur afin de communiquer avec les applications de la couche applicative par le biais de requête-réponse. Parmi les APIs les plus connues, on cite l'API REST, qui est une interface de programmation d'application qui fait appel à des requêtes HTTP (GET, POST, DELETE...) pour établir l'échange de requête [44].

1.3.5 La couche de données

C'est la couche la plus basse, elle contient tous les équipements de transmission tels que les switches et les routeurs. Son rôle est de transporter le trafic généré selon les directives et les décisions prises par le plan de contrôle.

1.3.6 Interface *Est/ouest*

Les interfaces côté Est/Ouest sont des interfaces de communication qui permettent généralement la communication entre les contrôleurs dans une architecture multi-contrôleurs pour synchroniser les états du réseau. Ces architectures sont très récentes et aucun standard de communication intercontrôleur n'est actuellement disponible [47].

1.4 Le protocole OpenFlow

OpenFlow est un protocole de communication entre le contrôleur et le commutateur dans un réseau SDN. Publié par l'*Open Networking Foundation* (ONF) à Stanford. Il s'agit d'un standard ouvert utilisé par le contrôleur pour transmettre aux commutateurs des instructions qui permettent de programmer leur plan de données et d'obtenir des informations de ces commutateurs afin que le contrôleur puisse disposer d'une vue globale logique (abstraction) du réseau physique [18]. En utilisant le protocole OpenFlow, un contrôleur peut non seulement définir des règles de transfert en envoyant des entrées de flux, mais aussi apprendre les statistiques réseau, les détails matériels, les ports, l'état de connectivité et la topologie réseau des commutateurs Ethernet.
Comme le montre la Figure 1.2, un commutateur OpenFlow comprend au moins trois parties :

1. Une ou plusieurs tables de flux : avec une action associée à chaque entrée de flux (supprimer, transmettre, modifier) pour indiquer au commutateur comment traiter les flux.

2. Un canal sécurisé : est l'interface utilisée pour la connexion entre chaque commutateur OpenFlow et un contrôleur. Grâce à cette interface, le contrôleur peut recevoir des

événements générés par le commutateur, envoyer des paquets hors du commutateur, configurer et gérer le commutateur. La connexion entre le commutateur et le contrôleur est cryptée avec le protocole TLS [18] .

3. Le protocole OpenFlow : qui fournit un moyen ouvert et standard de communication avec un commutateur.

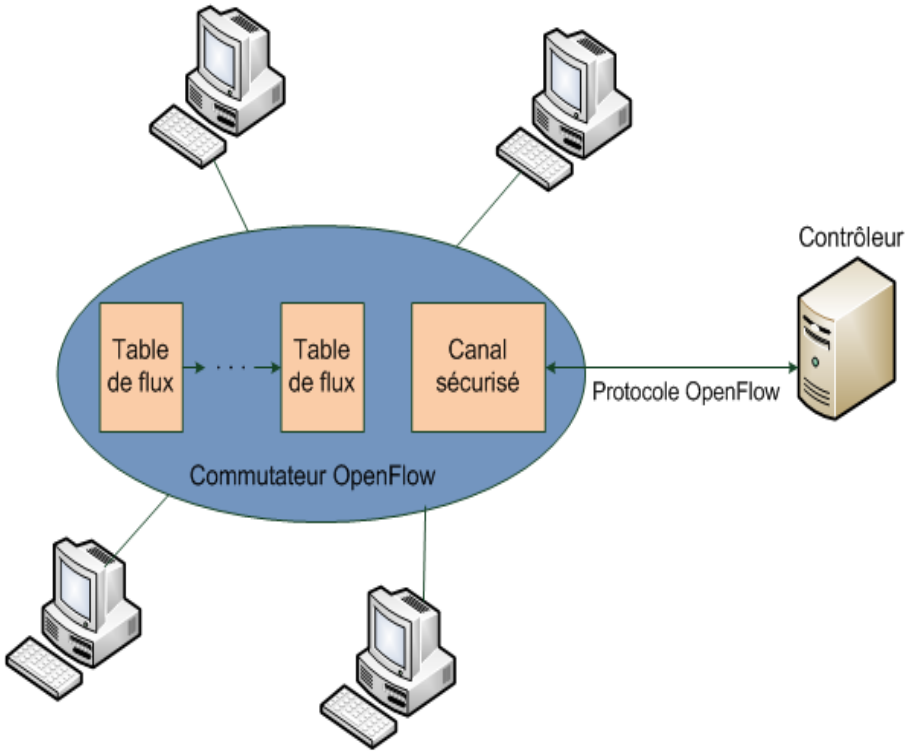


FIGURE 1.2 – Le commutateur OpenFlow et ses différentes parties [18]

1.4.1 Etablissement d'une connexion commutateur-contrôleur

Nous allons citer ci-dessous trois cas d'établissement d'une connexion entre le commutateur OpenFlow et le contrôleur :

1.4.1.1 Cas 1 : Connexion établie

Tout d'abord, il faut renseigner l'adresse IP du contrôleur au niveau du commutateur. Lors du démarrage du commutateur OpenFlow, ce dernier envoie un paquet *OFPT_HELLO* avec le numéro de version d'OpenFlow supportée. Le contrôleur vérifie la version d'OpenFlow supportée par le commutateur et lui répond par un message *OFPT_HELLO* en indiquant la version d'OpenFlow avec laquelle ils communiqueront. La connexion est ainsi établie.

1.4.1.2 Cas 2 : Échec de la connexion

Comme dans le cas précédent le commutateur OpenFlow envoie un paquet *OFPT_HELLO* avec la version du protocole utilisée, le contrôleur s'aperçoit qu'il ne supporte pas la version OpenFlow du commutateur. Il lui retourne donc un paquet *OFPT_ERROR* en indiquant que c'est un problème de compatibilité.

1.4.1.3 Cas 3 : Mode d'urgence

Comme dans les cas précédents le commutateur envoie un paquet *OFPT_HELLO* au contrôleur, si celui-ci ne répond pas, il se met alors en mode d'urgence *EMERGENCY_MODE*. Le commutateur utilise sa table de flux par défaut, si toutefois un paquet ne correspond à aucun enregistrement dans la table il le supprime.

1.4.2 Comportement d'un switch OpenFlow

Quand un switch reçoit un paquet, il compare son en-tête avec les règles de ces tables de flux. Si le masque configuré dans le champ d'en-tête d'une règle correspond à l'en-tête du paquet, la règle avec la plus haute prioritée est sélectionnée, puis le switch actualise son compteur pour cette règle comme montionné sur la figure 1.3.
Si l'entrée du flux correspondante au paquet n'existe pas dans toutes les tables de flux du commutateur, dans ce cas il s'agit d'une table manquante [18]. L'entrée de cette table dans la table de flux permet de spécifier le traitement de ce paquet en se basant sur une des instructions suivantes (selon la configuration de base du switch Openflow) :

1. Supprimer le paquet.
2. Transmettre le Paquet à une autre table de flux.
3. Envoyer le paquet au contrôleur via le canal sécurisé.

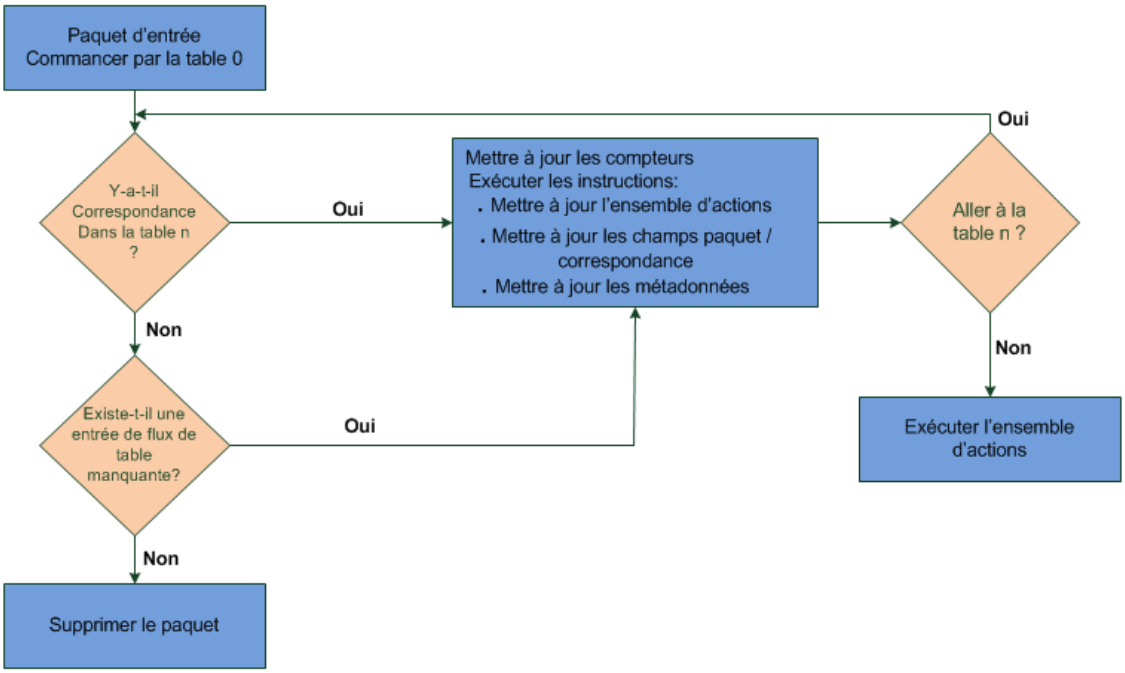


FIGURE 1.3 – Cycle de vie d'un paquet [18]

Les entrées de flux dans les tables de flux sont structurées comme mentionné sur la Figure 1.4.

- **Match Fields** : C'est la partie sur laquelle le contrôleur se base pour faire correspondre les paquets, cela consiste à vérifier le port d'entrée ou l'en-tête du paquet afin d'y appliquer une action X.

- **Actions** : Utilisé pour modifier l'ensemble des actions à appliquer sur le paquet.
- **Stats** : Il est possible de disposer d'un certain nombre de statistiques dont on se sert pour la gestion des entrées dans les tables de flux, pour ensuite décider si une entrée de flux est active ou non.

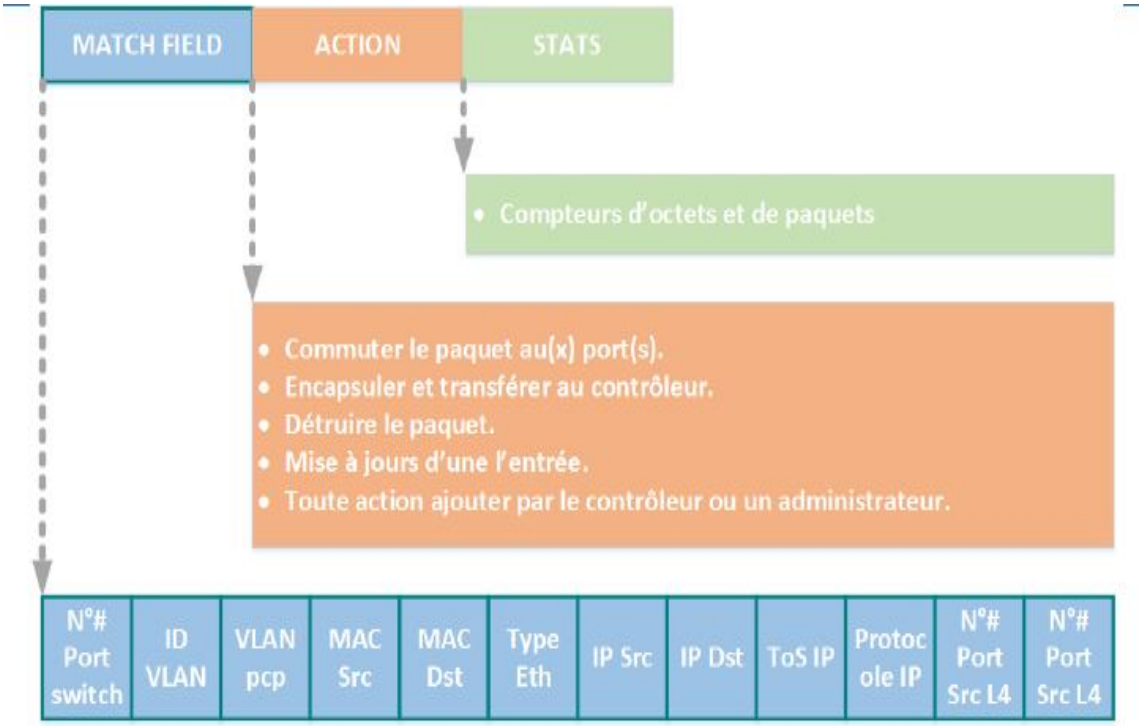


FIGURE 1.4 – Structure d'une entrée OpenFlow [18]

1.4.3 Messages OpenFlow

Le protocole de commutateur OpenFlow prend en charge trois types de message : symétrique, asynchrone et les messages de commandes [18].

1. **Les messages symétriques** : sont initiés par le commutateur ou le contrôleur et envoyés sans sollicitation. Par exemple, on trouve les messages HELLO qui sont échangés une fois que le canal sécurisé a été établi.

   - **Hello** : les messages Hello sont échangés entre le commutateur et le contrôleur au démarrage de la connexion.
   - **Echo** : les messages de demande/réponse d'écho peuvent être envoyés à partir du commutateur ou du contrôleur, et renvoient une réponse d'écho. Ils servent principalement à vérifier la qualité de vie d'une connexion contrôleur-commutateur.
   - **Erreur** : le commutateur utilise les messages d'erreur pour signaler les problèmes à l'autre côté de la connexion. Ils sont principalement utilisés par le commutateur pour indiquer l'échec d'une requête initiée par le contrôleur.

2. **Les messages asynchrones** : initiées par le commutateur OpenFlow et sont utilisés pour informer le contrôleur du passage des paquets ou du changement de l'état du commutateur.

   - **Packet-in** : pour tous les paquets qui ne disposent pas d'une entrée de flux correspondante ou dans le cas où l'action correspondante est l'envoi au contrôleur.
   - **Flow-Removed** : informe le contrôleur de la suppression d'une entrée de flux d'une table de flux.
   - **Port-status** : informe le contrôleur d'une modification sur un port.
   - **Role-status** : informe le contrôleur d'un changement de son rôle. Lorsqu'un nouveau contrôleur choisit lui-même le maître, le commutateur est censé envoyer des messages d'état de rôle à l'ancien contrôleur maître.
   - **Flow-monitor** : informe le contrôleur d'une modification dans une table de flux.

3. **Les messages de commande** : ils sont envoyés par le contrôleur afin de récolter des informations et configurer les commutateurs à travers leurs tables de flux. Les trois principaux types de messages de commande sont :

   - **Read-state** : ils permettent au contrôleur de récolter des informations sur le plan de données telles que les statistiques des flux et la configuration du commutateur avec lequel il interagit.
   - **Modify-state** : ce sont ces messages qui rendent possible au contrôleur la modification, la création et la suppression d'entrées (règles) dans les tables de flux des commutateurs Openflow.
   - **Packet-out** : ces messages viennent en réponse aux messages Packet-in envoyés par le commutateur. Ils permettent de montrer au commutateur vers quel port de sortie le paquet concerné par le message Packet-in doit être acheminé.

1.5 Programmabilité dans le SDN

1.5.1 Les contrôleurs

Les solutions SDN reposent sur la mise en place de contrôleurs. Leurs mission est de fournir une couche d'abstraction du réseau et de présenter ce dernier comme un système. Le contrôleur SDN permet d'implémenter rapidement un changement sur le réseau en traduisant une demande globale en une suite d'opérations sur les équipements réseau (requêtes Netconf , ajout d'états Openflow, configuration en CLI...). Les ordres sont donnés au contrôleur par une application via une API dite *Northbound*. Les éditeurs logiciels de contrôleurs publient la documentation de l'API afin de permettre d'interfacer des applications. Le contrôleur communique avec les équipements via une ou plusieurs API dites *Southbound* . Afin de pouvoir interagir avec le

réseau, le contrôleur a besoin d'une vue précise du réseau. 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 équipements à implémenter une directive ainsi que les APIs supportées par chaque équipement.

1.5.2 Les modèles de réseaux programmables

Le SDN englobe toutes les solutions permettant une programmation du réseau, afin de mieux interagir avec les applications. Diverses solutions coexistent, elles sont adaptées selon les besoins des utilisateurs. La Figure 1.5 illustre les différents modèles de programmabilité.

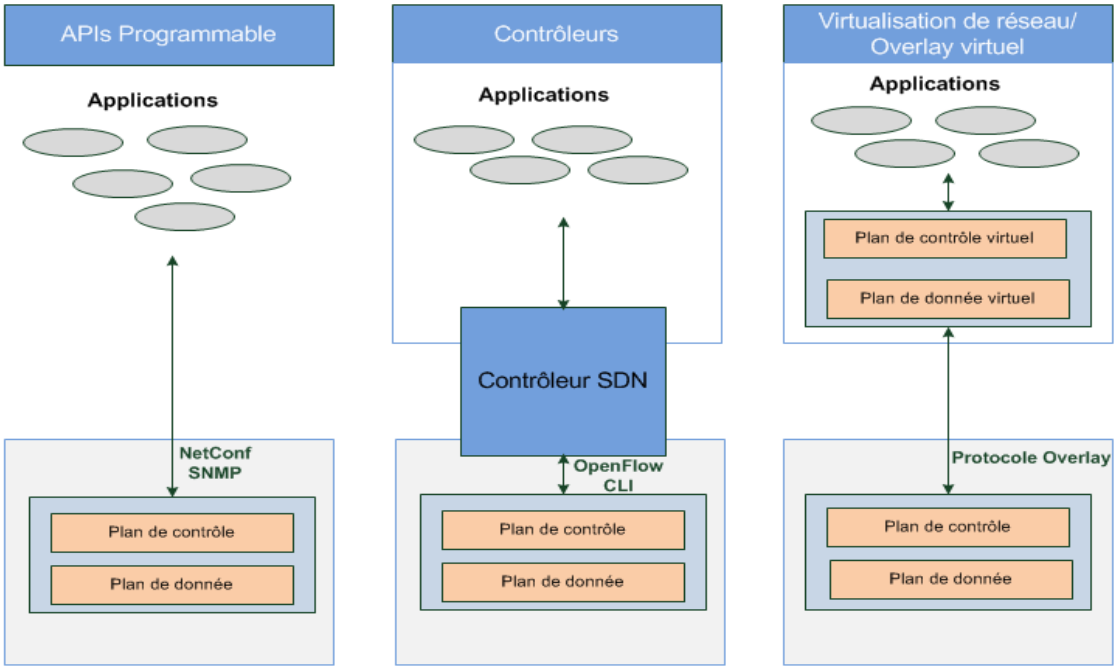


FIGURE 1.5 – Modèles de programmabilité en SDN [2]

- Création d'un réseau virtuel : Dans ce modèle, les applications s'affranchissant des contraintes du réseau physique en créant leurs propre réseau *overlay* (se trouve au dessus du réseau physique). Ce dernier a pour mission une simple connectivité entre les noeuds d'extrémité des tunnels et le réseau d'overlay qui assure l'intégralité des services [41].
- Programmabilité individuelle de chaque équipement : Dans ce modèle, une application qui peut être incluse dans les ressources ou centralisée interagit directement dans chaque équipement pour effectuer une tâche spécifique [41].
- Programmabilité via un contrôleur : Dans ce modèle, un contrôleur reçoit de l'application une requête qui est sous forme d'un ordre global et abstrait pour le transformer en un ensemble d'instructions auprès des équipements du réseau concerné [41].

1.5.3 La virtualisation des fonctions réseaux NFV

Les fournisseurs de services se sont tournés vers les technologies de virtualisation informatique standard, afin d'accélérer le déploiement de nouveaux services réseau. Ils ont découvert que

NFV (*Network Function Virtualization*) est une approche consistant à réaliser certaines fonctions permettant d'accélérer l'innovation et le provisionnement des services. Avec cela, plusieurs fournisseurs se sont regroupés dans l'ETSI (*European Telecommunications Standards Institute*) afin de créer l'architecture et les exigences de base de NFV [28].

NFV est une facette importante du SDN particulièrement étudiée chez les fournisseurs de services qui voient ici une solution pour mieux ajuster l'investissement selon les besoins de leurs clients. Au lieu de déployer du matériel ad-hoc pour chaque demande, l'opérateur va piocher dans une capacité de calcul globale, facilement extensible via l'ajout de serveurs [41].

1.5.4 Programmation des équipements

Programmer le réseau n'est possible que si chaque équipement offre une certaine dose de programmabilité. Or, la programmation des équipements réseau nécessite sur ces derniers la capacité de recevoir des directives de l'extérieur. Pour cela des interfaces de programmation sont nécessaires, des API. Nous citons quelques APIs les plus communément supportées sur les équipements réseau :

- **Netconf/Yang** : Netconf (*Network Cofiguration Protocol*) est le protocole de configuration de réseau de l'IETF visant à standardiser les directives réseau pour l'installation, la manipulation et la suppression de la configuration des périphériques réseau tandis que Yang est le langage utilisé pour modéliser à la fois la configuration et les données d'état des éléments du réseau. Ce modèle standardisé est le plus souvent privilégié par les service providers qui cherchent au maximum à être indépendants des constructeurs, sans pour autant perdre la maîtrise du réseau.

- **RESTful API** : *Representational state transfer* est un style architectural logiciel qui définit un ensemble de contraintes à utiliser pour créer des services Web. Les services Web conformes au style architectural REST, appelés services Web RESTful, assurent l'interopérabilité entre les systèmes informatiques sur Internet.

- **CLI** : L'accès en ligne de commande aux équipements via telnet/ssh est une API la plus souvent utilisée pour programmer/configurer le réseau.

- **OpenFlow** : est une API permettant la programmation du plan de données.

1.5.5 Orchestration

Dans le cas de déploiement d'une application dans différents types d'environnement (WAN, LAN, DATACENTER ...) un seul contrôleur ne pourra pas programmer l'ensemble de la chaîne, ce qui a permis l'apparrition de l'orchestrateur qui offre une programation de bout en bout comme illustré dans la Figure 1.6.

Ce dernier reçoit un ordre depuis une application (par exemple un portail de fournisseur de services) et effectue les actions nécessaires pour réaliser la tâche demandée. L'orchestrateur doit être capable de valider chaque étape quand il réalise une action afin de revenir en arrière en cas d'échec. Il ne peut pas laisser le réseau dans un mode bancal où seule une partie aurait été configurée avec un nouveau service.

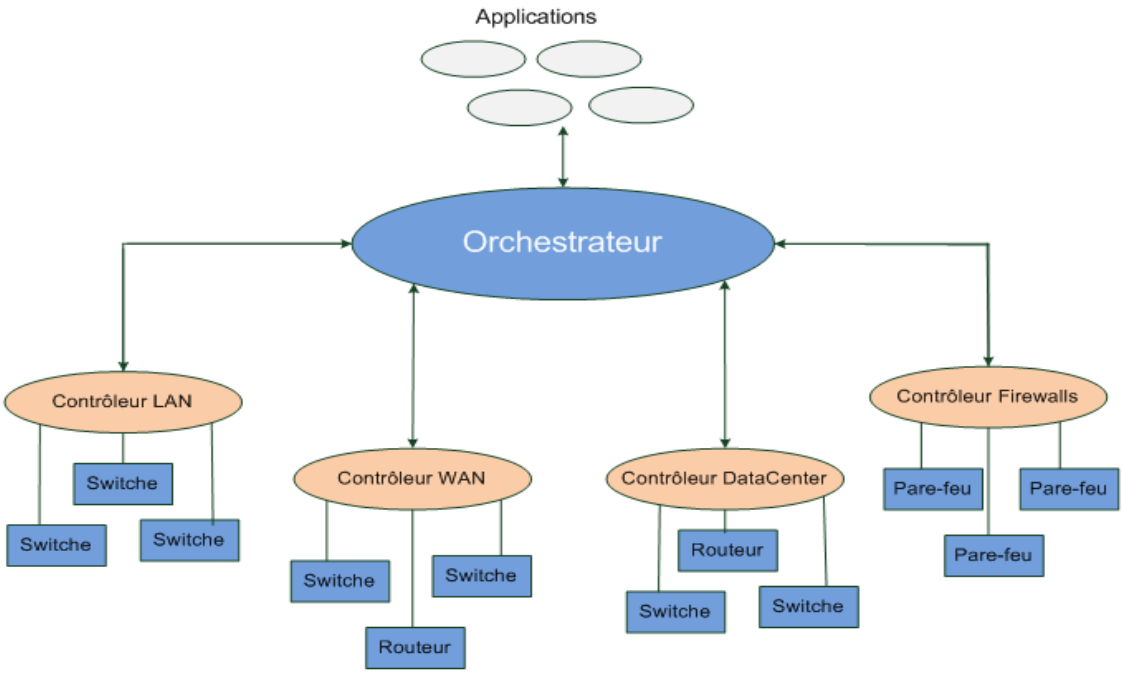


FIGURE 1.6 – Exemple d'utilisation d'un orchestrateur[46]

1.6 Domaines d'applications du SDN

Les réseaux SDN ont une large variété d'applications dans les environnements réseaux et permettent un contrôle personnalisé. Ils simplifient également le développement et le déploiement de nouveaux services et protocoles réseaux. Selon IDC (*International Data Corporation*), le marché de SDN est composé des équipements d'infrastructure, des logiciels de contrôle, de virtualisation ; des applications de réseau et de sécurité sera évalué à 12.5 milliards de dollars en 2020 [4]. Cette section présentera quelques exemples d'applications SDN.

1.6.1 Data center et Cloud computing

SDN est hautement considéré comme une des solutions les plus récentes, qui permet de configurer et de gérer facilement le Cloud et les centres de données. Le géant d'Internet Google a expérimenté les bénéfices du SDN dans la gestion et le contrôle de ses centres de données autour du globe [48].

1.6.2 Multimédia et QOS

L'architecture Internet d'aujourd'hui repose sur l'envoi des paquets sans atteindre les performances requises lors de l'augmentation du trafic. Les applications multimédias comme le streaming vidéo, la vidéo à la demande, la vidéo-conférence ...etc nécessitent des ressources réseau stables et tolèrent les erreurs et les retards de transmission. En se basant sur la vue centralisée du réseau offerte par SDN, on peut sélectionner selon le débit des chemins différents pour les divers flux du trafic ; ce qui a permis la naissance d'une architecture réseau qui détermine la trajectoire optimale en utilisant une vue d'ensemble du réseau VSDN [43].

1.6.3 Les Réseaux mobiles sans fil

SDN a été aussi appliqué dans le domaine des réseaux mobiles sans fil, en particulier la 5G. En fait, selon le rapport d'Ericsson, le trafic mobile a augmenté de 60 % entre 2015 et le premier trimestre de l'année 2016 [3]. Gérer cette nouvelle masse de données est le plus grand défi des opérateurs, puisqu'une mauvaise gestion conduirait à une interférence entre les différents signaux sans fils. Le SDWN (*Software Defined Wireless Network*) a été proposé comme solution d'intégration du SDN pour les réseaux mobiles sans fil, pour faciliter la gestion et supporter le déploiement de nouvelles applications dans l'infrastructure réseau.

1.7 Avantages du SDN

Face à un réseau de plus en plus complexe le SDN s'impose comme une évidence, même si une certaine méfiance est encore de mise chez certains décideurs en entreprise, grace aux services et aux avantages qui le caractérise [47] :

1.7.1 Configuration automatique

SDN permet aux administrateurs réseaux de configurer, administrer, sécuriser et optimiser les réseaux rapidement grâce à des programmes SDN dynamiques et automatisés. Ces programmes peuvent être développés par eux mêmes car ils ne dépendent plus de logiciels propriétaires.

1.7.2 Capacité d'extension : scalabilité

Avec le SDN, on acquiert une dimension de programmabilité du réseau. Ce dernier permet en effet de changer facilement le comportement du réseau, de rajouter une nouvelle pièce ou un nouveau service sans que cela ne se traduise par des délais très importants.

1.7.3 Souplesse et flexibilité

Le SDN est indissociable des notions de souplesse et de flexibilité. En effet, les services liés aux réseaux (priorités, routage...) sont regroupés dans un contrôleur, qui chapeaute différents équipements. Des règles peuvent ainsi être édictées afin d'automatiser les tâches. On parle alors de la programmabilité du réseau. Les règles définies au sein du contrôleur sont ainsi transmises aux différents équipements. Il en découle que l'administrateur peut très facilement automatiser, voire programmer son réseau.

1.7.4 Basé sur des standards ouverts

L'implémentation à travers des standards ouverts permet de simplifier l'architecture réseau car les instructions sont fournies par un ou plusieurs contrôleurs au lieu de multiple équipements propriétaires.

1.8 Les fails du SDN

S'il existe un point précis qui mérite une réflexion poussée sur la sécurité, c'est bien le contrôleur centralisant les opérations. Sa sécurisation n'est pas intuitive, car c'est par définition un système ouvert, éminemment critique et devant, à ce titre, bénéficier de fonctions de haute disponibilité.

- Le contrôleur centralisé est un point potentiel d'attaque et d'échec.

- La SBI est vulnérable aux menaces qui pourraient dégrader la disponibilité, les performances et l'intégrité du réseau.

- la vulnérabilité des contrôleurs et des commutateurs Openflow aux attaques de déni de service [54].

1.9 Conclusion

Ce chapitre transcrit le travail de documentation et de recherche qui a été mené dans l'objectif de vulgariser le plus possible la technologie SDN, qui reste sujette à plusieurs interprétations . Nous avons notamment cité quelques exemples d'applications et explicité le rôle du contrôleur au sein d'une architecture SDN ainsi que le fonctionnement du Protocol Openflow. Ces éléments nous ont été utiles lors de la mise en place de la solution proposée.

Chapitre 2

La gestion du trafic dans les réseaux SDNs

2.1 Introduction

La gestion du trafic est un mécanisme important pour optimiser les performances d'un réseau de données en analysant, prédisant et régulant dynamiquement le comportement des données transmises. Elle a été largement exploitée dans les réseaux de données traditionnels, tels que les réseaux ATM *Asynchronous Transfer Mode* et les réseaux *Multi-Protocol Label Switching* (IP/MPLS). Cependant, ces paradigmes ne sont plus en adéquation avec les réseaux actuels pour 2 raisons principales[50].

Premièrement, les applications d'aujourd'hui nécessitent que l'architecture de réseau sous-jacente réagisse en temps réel et soit évolutive pour une grande quantité de trafic. L'architecture doit être capable de classer une variété de types de trafic provenant de différentes applications, et de fournir un service adapté et spécifique pour chaque type de trafic dans un laps de temps très court.
Deuxièmement, face à la croissance rapide du *cloud computing* et donc à la demande de centres de données à grande échelle, une gestion de réseau adaptée devrait être en mesure d'améliorer l'utilisation des ressources pour de meilleures performances du système. Ainsi, de nouvelles architectures de mise en réseau et des outils de gestion du trafic plus intelligents et efficaces sont nécessaires.

Dans la littérature, les mécanismes de gestion du trafic actuels sont largement étudiés dans les réseaux ATM, Internet IP et MPLS. D'un autre côté, SDN promet de simplifier considérablement la gestion du réseau, de réduire les coûts d'exploitation et de promouvoir l'innovation et l'évolution dans les réseaux actuels et futurs. Dans ce chapitre, nous traitons brièvement les mécanismes de gestion de trafic classiques développés pour les réseaux ATM, IP et MPLS, puis dressons un état de l'art de la gestion du trafic pour les réseaux SDN du point de vue industriel et recherche universitaire [50].

2.2 La gestion du trafic dans les réseaux traditionnels

La gestion du trafic signifie que le trafic du réseau est mesuré et analysé afin d'améliorer les performances d'un réseau opérationnel au niveau du trafic et des ressources exploitées [26]. Dans cette section, nous passons en revue le concept et les mécanismes de la gestion du trafic du point de vue historique comme le montre la Figure 2.1.

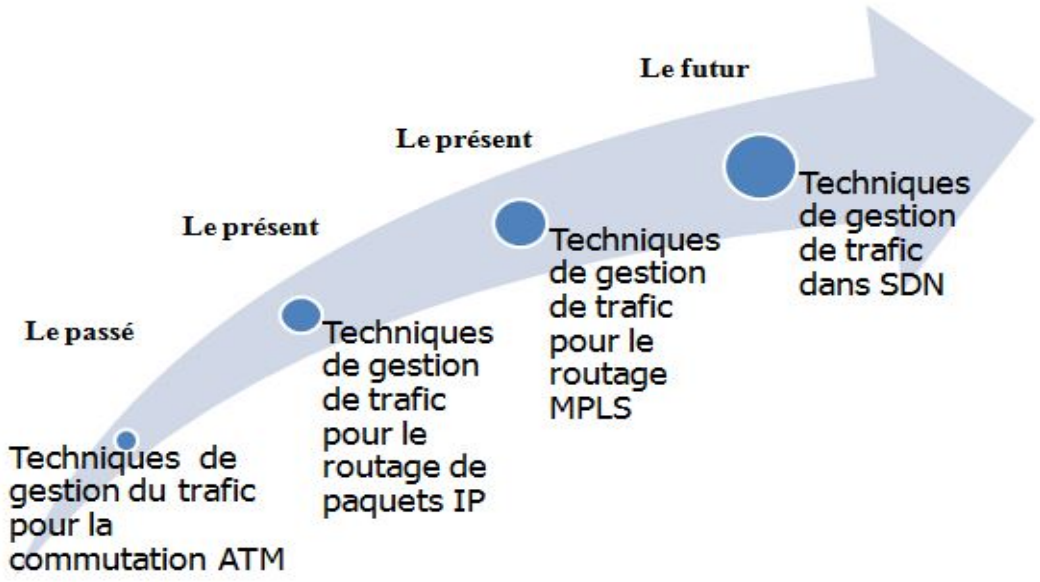


FIGURE 2.1 – La gestion du trafic du passé au futur [50].

2.2.1 La gestion du trafic basée sur l'ATM

À la fin des années 80, le mode de transfert asynchrone (ATM) a été développé et sélectionné pour permettre la pleine utilisation des réseaux numériques à service intégré à large bande (RNIS-LB). L'ATM combine l'acheminement par commutation de circuits des réseaux téléphoniques publics, la commutation de paquets de réseaux de données privés et le multiplexage asynchrone de paquets.

L'objectif clé de la gestion du trafic dans ATM était de résoudre principalement le problème de contrôle de la congestion. Ce contrôle est fait pour répondre aux diverses exigences de service et de performance du trafic multimédia telles que : le haut débit et la faible latence en raison de la demande croissante de celle-ci (par exemple : les données, la voix et la vidéo).

Les schémas de contrôle de la congestion sont classés en deux méthodes : contrôle réactif et contrôle préventif.

- Le contrôle réactif demande aux nœuds sources de limiter leurs flux de trafic au début de la congestion grâce à l'envoie d'indicateur de congestion. Cependant, un problème majeur avec le contrôle réactif dans les réseaux à grande vitesse est sa lenteur avec rétroaction parce que le contrôle réactif est invoqué pour la congestion après qu'elle se soit produite [45].
- Dans les schémas de contrôle préventif, contrairement au contrôle réactif, les nœuds sources évitent la congestion. Ils essaient d'empêcher le réseau d'atteindre un niveau de congestion inacceptable.

Ce contrôle préventif de l'ATM peut être effectué de trois façons : contrôle d'admission, application de la bande passante et classification du trafic.

2.2.1.1 Contrôle d'admission

Avec ce contrôle, le réseau décide d'accepter ou de rejeter une nouvelle connexion en fonction du fait que l'exigence de QoS requise de la nouvelle demande peut être satisfaite. Lorsqu'une nouvelle connexion est demandée, le réseau examine ses besoins en matière de service (délai de transmission et probabilité de perte) et les caractéristiques du trafic (débit maximum, débit moyen, etc.). Le réseau examine ensuite la charge actuelle et décide d'accepter ou non la nouvelle connexion.

2.2.1.2 Application de la bande passante

Le mécanisme d'application de la bande passante est implémenté aux bords du réseau en utilisant la méthode Leaky Bucket. Une implémentation possible de cette méthode consiste à contrôler le flux de trafic au moyen de jetons, dans lesquels un modèle de mise en file d'attente est utilisé et le débit total du réseau peut être amélioré en utilisant la méthode de marquage [22].

2.2.1.3 Classification du trafic

Les réseaux ATM doivent prendre en charge diverses exigences de service et de performances. Pour cela plusieurs classes de trafic et des mécanismes de priorité peuvent être utilisés, plutôt que des mécanismes de contrôle uniformes. Ce qui signifie que différents niveaux de priorité sont accordés à différentes classes de trafic. Il existe deux façons d'utiliser les priorités :

- Utiliser un mécanisme de priorité comme méthode de planification. C'est-à-dire une discipline de mise en file d'attente. De cette manière, différentes exigences de retard peuvent être satisfaites en planifiant d'abord le trafic sensible au retard ou urgent.
- Utiliser un schéma de priorité pour contrôler la congestion.

2.2.2 La gestion du trafic basée IP

La gestion du trafic IP est une fonctionnalité importante pour les fournisseurs Internet qui essaient d'optimiser les performances du réseau et la distribution du trafic. Dans [26], la gestion du trafic Internet est définie comme une ingénierie de réseau à grande échelle qui traite l'évaluation et l'optimisation des performances du réseau IP.

2.2.2.1 Le routage de chemin le plus court

L'idée de base du routage de chemin le plus court est de définir les poids de liaison des protocoles de passerelle intérieure IGP en fonction de la topologie de réseau donnée et de la demande de trafic pour contrôler le trafic intra-domaine afin de répondre aux objectifs de gestion du trafic. La plupart des grands réseaux IP exécutent des protocoles de passerelle intérieure IGP tels que Open Shortest Path First OSPF ou Système intermédiaire – Système intermédiaire IS – IS qui sélectionnent les chemins en fonction des poids des liens statiques. Les routeurs utilisent ces protocoles pour échanger les poids de liaison et construire une vue complète de la topologie à l'intérieur du système autonome (AS). Ensuite, chaque routeur calcule les chemins les plus courts et crée une table qui contrôle la transmission de chaque paquet IP au prochain saut.

2.2.2.2 Routage multi-chemins à coût égal

Dans le routage multi trajet à coût égal, les grands réseaux sont généralement divisés en plusieurs zones OSPF/IS – IS. Dans certains cas, le réseau peut avoir plusieurs chemins les plus courts entre la même paire de routeurs. Le routage basé sur cette technique n'est pas suffisamment flexible, en raison de la demande dynamique de trafic, les volumes de trafic fluctuent dans le temps et des pannes inattendues peuvent entraîner des modifications de la topologie du réseau.

2.2.3 La gestion du trafic basée sur MPLS

MPLS [31] était présenté comme une solution attrayante pour la gestion du trafic en abordant les contraintes des réseaux IP. La plupart des avantages de l'ingénierie du trafic MPLS reposent sur le fait qu'il peut prendre en charge efficacement le routage explicite entre la source et la destination. Il est flexible et peut donc répartir le trafic efficacement à des fins d'optimisation du routage et de la transmission.

2.2.3.1 Tunnels LSP

Une caractéristique importante de la gestion du trafic basée sur MPLS est : *Tunnels Label Switching Path* LSP établis par un protocole de signalisation tels que le protocole de réservation de ressources RSVP. Les ressources du réseau peuvent être allouées par plusieurs tunnels LSP qui peuvent être créés entre deux nœuds, et le trafic entre les nœuds est divisé entre les tunnels selon une politique locale [50].

2.3 La gestion du trafic dans les réseaux SDNs

Les mécanismes de gestion du trafic dans les réseaux SDN peuvent être plus efficacement et intelligemment mis en œuvre comme un système centralisé comparé aux approches précédemment citées. En raison des avantages majeurs de l'architecture SDN -voir la section 1.7-. Les mécanismes de gestion du trafic actuels se concentrent principalement sur quatre volets : Figure 2.2[20].

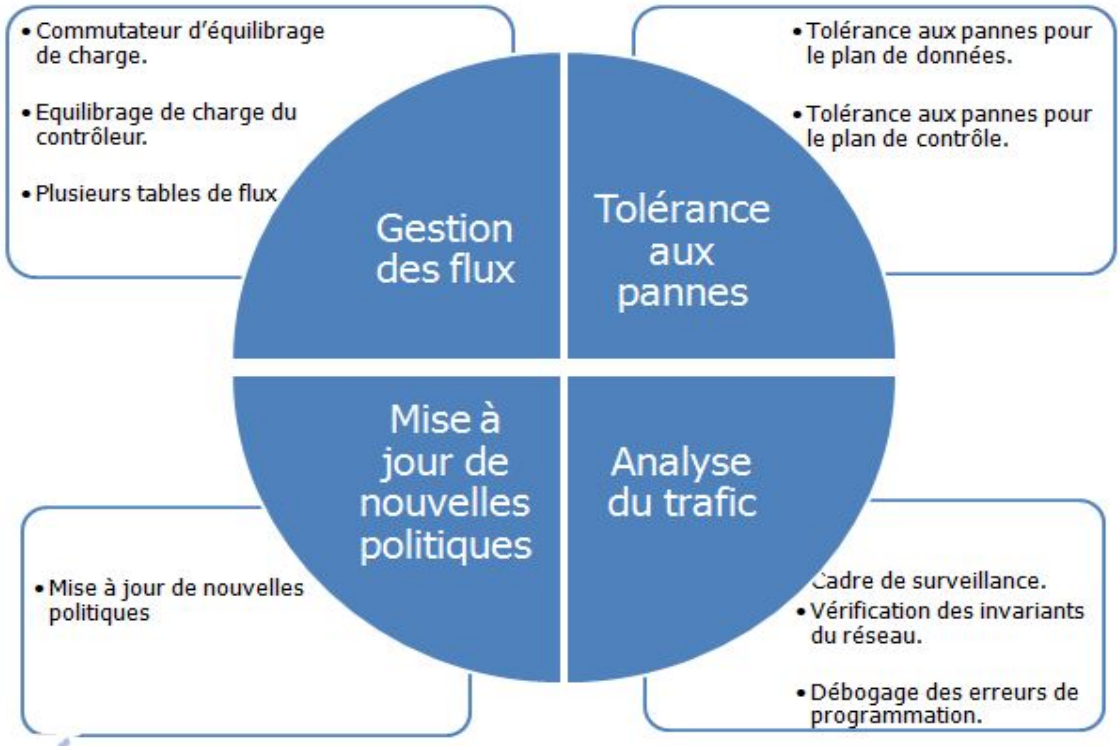


FIGURE 2.2 – Les approches de gestion du trafic dans les les réseaux SDN [20]

2.3.1 Gestion des flux

Dans SDN, lorsqu'un commutateur OpenFlow reçoit un flux qui ne correspond à aucune règle dans l'entrée de flux d'un commutateur, le premier paquet du flux est transmis au contrôleur. En conséquence, le contrôleur décide s'il est nécessaire d'installer une nouvelle redirection/règle dans les commutateurs ; ce qui peut entraîner une charge de trafic dans le réseau. Cependant, ce processus d'installation de règle de transfert peut prendre du temps et générer des pics de retard. De plus, si un nombre élevé de nouveaux flux sont agrégés, un surdébit significatif peut être généré à la fois dans le plan de contrôle et dans le plan de données.

Dans cette section nous étudions les solutions qui visent à éviter ce goulot d'étranglement dans SDN en considérant les compromis entre latence et équilibre de charge.

2.3.1.1 Equilibrage de charge pour le trafic du plan de donnée

La technologie de routage par trajets à coût égal (ECMP) basée sur un algorithme de hachage est une solution efficace d'équilibrage de charge [27]. Elle permet de répartir les flux entre les chemins disponibles en utilisant la technique de hachage de flux. Les commutateurs compatibles ECMP sont configurés avec plusieurs chemins de transfert possibles pour un sous-réseau donné. Lorsqu'un paquet avec plusieurs chemins candidats arrive, il est transmis sur celui qui correspond à un hachage des champs sélectionnés des en-têtes de ce paquet modulo le nombre de chemins, répartissant ainsi la charge de chaque sous-réseau sur plusieurs chemins.

Ce mappage statique des flux vers les chemins ne tient compte ni de l'utilisation actuelle du réseau ni de la taille du flux, ce qui entraîne des collisions qui submergent les tampons de commutation et dégradent l'utilisation globale des commutateurs et des liaisons. Pour pallier un tel problème, trois solutions d'équilibrage de charge sont proposées :

1. Hedera [50] : *Dynamic Flow Scheduling for Data Center Networks* est une planification de flux évolutive et dynamique pour éviter les limitations de l'ECMP. Il a une vue de routage et des demandes de trafic, collecte des informations de flux à partir des commutateurs, calcule des chemins non conflictuels pour les flux et ordonne aux commutateurs de réacheminer le trafic en conséquence.
2. Mahout [50] : *Low-Overhead Datacenter Traffic Management using End-Host-Based Elephant Detection* gère les flux de trafic en exigeant la détection de flux importants qui transportent une grande quantité de données et marque les paquets de ce flux en utilisant un mécanisme de signalisation intra-bande. Les commutateurs de ce réseau sont configurés pour transmettre ces paquets marqués au contrôleur Mahout. A ce moment, le contrôleur calcule le meilleur chemin pour ce flux et installe une entrée spécifique au flux dans le commutateur.
3. MicroTE [50] : *Fine Grained Traffic Engineering For Data Centers* est une approche très similaire à Mahout. Il s'agit d'un schéma d'ingénierie du trafic pour détecter les flux importants sur les hôtes finaux de sorte que lorsqu'une grande partie du trafic est prédite, MicroTE les achemine de manière optimale. Sinon, les flux sont gérés par le schéma ECMP avec seuil heuristique.

2.3.1.2 Équilibrage de charge du contrôleur

Chaque fois qu'un flux est initié dans le réseau, le commutateur Openflow doit transmettre le premier paquet du flux au contrôleur pour décider d'un chemin de transfert approprié. Un contrôleur unique et centralisé ne peut pas fonctionner efficacement, en raison de l'augmentation du nombre d'éléments de réseau ou de flux de trafic. En outre, tout en n'offrant qu'un seul type de garanties de service, ce contrôleur unique ne parvient pas à gérer les différentes demandes entrantes. Pour cela, plusieurs solutions prometteuse sont proposé de déployer multiples contrôleurs pour éviter ce goulot d'étranglement entre les contrôleurs et les commutateurs OpenFlow :

1. HyperFlow [23] : est un plan de contrôle qui utilise le protocole OpenFlow pour configurer les commutateurs. HyperFlow localise la prise de décision sur les contrôleurs individuels pour minimiser le temps de réponse du plan de contrôle aux demandes du plan de données et offre une évolutivité tout en gardant le contrôle du réseau centralisé de manière logique. Il permet à tous les contrôleurs de partager la même vue cohérente à l'échelle du réseau et de servir localement les demandes sans contacter activement aucun nœud distant, ce qui minimise les temps de configuration du flux.
2. Kandoo[19] : est un plan de contrôle distribué basé sur une hiérarchie à deux niveaux pour les contrôleurs : les contrôleurs locaux et un contrôleur racine logiquement centralisé. Les contrôleurs locaux exécutent les applications qui traitent les évènements localement par contre le contrôleur centralisé exécute les applications qui nécessitent un accès à l'état du réseau. Chaque commutateur est contrôlé par un seul contrôleur Kandoo et chaque contrôleur Kandoo peut contrôler plusieurs commutateurs. Si le contrôleur racine doit installer des entrées de flux sur les commutateurs d'un contrôleur local, il délègue les demandes au contrôleur local respectif.
3. BalanceFlow[57] : est une architecture d'équilibrage de charge de contrôleur pour des réseaux OpenFlow étendus, qui peut partitionner la charge de trafic de contrôle entre différents contrôleur de manière plus flexible. Le principe de BalanceFlow se concentre sur la répartition dynamique des demandes de flux entre les contrôleurs pour obtenir une réponse rapide, et sur la répartition de la charge sur un contrôleur surchargé entre les contrôleurs appropriés à faible charge pour maximiser l'utilisation du contrôleur. Il permet à tous les contrôleurs de conserver leurs propres informations de demandes de flux et de publier régulièrement ces informations via un système de communication inter-contrôleurs pour prendre en charge l'équilibrage de charge.

2.3.1.3 Plusieurs tables de flux

Les multiples tables de flux pour la gestion du trafic sont également à prendre en considération. Les flux sont définis par une séquence de paquets pour chaque flux, de son origine à sa destination, qui partagent certaines caractéristiques communes. Au début, les commutateurs basés sur la spécification OpenFlow v1.0 [9] ont un modèle de table de correspondance unique généralement construit sur *Ternary Content Addressable Memory* TCAM [55]. Dans ce concept OpenFlow, un flux est identifié en utilisant son champ d'en-tête de paquet correspondant à une combinaison d'au moins 10 tuples comprenant le port d'entrée, l'ID de VLAN, Ethernet, IP et les champs d'en-tête TCP. Ces champs agrégés sont placés dans une seule table de flux dans le TCAM d'un commutateur OpenFlow. Cependant, la table unique pour l'implémentation des règles de flux crée un ensemble de règles énorme et pourrait entraîner une incapacité pour les déploiements à grande échelle car l'espace TCAM est une ressource limitée et coûteuse.

Pour rendre la gestion des flux plus flexible et efficace, OpenFlow v1.1 [10] a introduit le mécanisme de plusieurs tables de flux et actions associées à chaque entrée de flux, comme illustré à la Figure 2.3. L'orsqu'un paquet arrive au commutateur, il identifie l'entrée de flux de correspondance et applique ensuite des instructions ou des actions en fonction des champs de flux. Les flux inconnus qui ne correspondent à aucune entrée de flux dans les multiples tables de flux peuvent être transmis au contrôleur ou supprimés. Ainsi, en décomposant la table de flux unique (avec la longueur d'une entrée de flux avec environ 40 attributs, dépassant 1000

bits) plusieurs ensembles de tables plus normalisés, un tel mécanisme améliore considérablement l'utilisation de TCAM et accélère également le processus d'appariement.

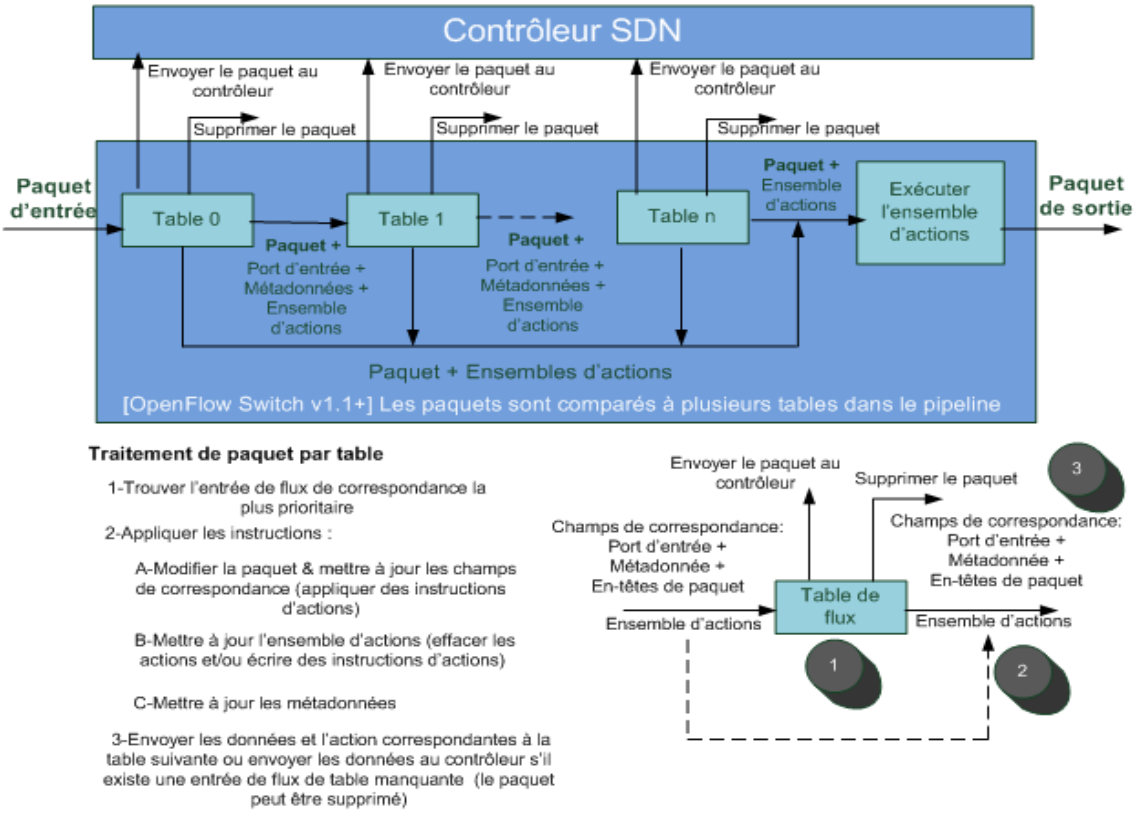


FIGURE 2.3 – Flux de paquets traversant plusieurs pipelines de table de flux

2.3.2 Tolérance aux pannes

Pour garantir la fiabilité du réseau, SDN doit avoir une capacité pour effectuer la reprise après incident de manière transparente. Lorsque des pannes se produisent dans l'infrastructure réseau (c'est-à-dire les contrôleurs, les commutateurs et les liaisons), les mécanismes de gestion du trafic doivent fournir une récupération rapide des défaillances. La réalisation d'une récupération rapide après une défaillance est une tâche difficile pour le SDN car le contrôleur central en restauration doit calculer de nouvelles routes et informer immédiatement tous les commutateurs concernés d'une action de récupération.

Dans cette section, nous étudions les efforts de recherche actuels sur la réalisation d'une récupération rapide après défaillance dans les réseaux SDN.

2.3.2.1 Tolérance aux pannes pour le plan de données

La récupération après défaillance des éléments réseau et des liaisons est basée principalement sur deux types de mécanismes : [61].

- Restauration : les chemins de récupération peuvent être soit pré-planifiés soit alloués dynamiquement, mais les ressources ne sont pas réservées jusqu'à ce que la panne se produise. Lorsqu'une défaillance se produit, une signalisation supplémentaire est nécessaire pour établir le chemin de restauration.

- Protection : les chemins sont pré-planifiés et réservés avant une panne. En cas de défaillance, aucune signalisation supplémentaire n'est nécessaire pour établir le chemin de protection.

La restauration est une stratégie réactive tandis que la protection est une stratégie proactive. Les solutions de restauration et de protection pour les réseaux SDN fonctionnent souvent comme mentionné dans le Tableau 2.3

- Restauration du plan de données [61] : Une fois que le contrôleur a reçu une notification de défaillance d'une liaison, une liste de tous les chemins affectés est créée. Pour tous ces chemins affectés, un chemin de restauration est calculé à l'aide d'un algorithme de chemin le plus court sur la topologie restante. Pour les commutateurs affectés qui sont à la fois sur le chemin de travail et de restauration, l'entrée de flux est modifiée. Pour les autres commutateurs, il y a 2 possibilités. Si les commutateurs se trouvent uniquement sur le chemin d'accès défaillant, les entrées sont supprimées. S'ils se trouvent uniquement sur le chemin de restauration, les nouvelles entrées sont ajoutées.
- Protection du plan de données [36] : dans cette opération, le chemin de protection est pré-calculé et il est installé avec le chemin de travail dans les entrées de flux au niveau des commutateurs, de sorte que chaque commutateur possède deux informations de transmission, une pour le chemin de protection et l'autre pour le chemin d'origine. Une fois la défaillance détectée, par exemple, via la détection de retransmission bidirectionnelle (BFD) [51] dans le chemin de travail, le commutateur utilisera le chemin de protection pour le transfert du flux.

Approches proposées	Schémas de reprise après incident	Temps de restauration maximum	Temps de protection maximum
Schéma de récupération rapide après une défaillance	Restauration du plan de données	(80—130 ms) > 50 ms	N/A
Programme de récupération Carrier-grade	Restauration et protection du plan de données	60 ms > 50 ms	(42—48 ms) < 50 ms
Schéma de protection de segment (OSP) basé sur OpenFlow	Protection du plan de données	N/A	64 ms > 50 ms

TABLE 2.1 – Vue d'ensemble qualitative des différents schémas de tolérance aux pannes pour le plan de données[35]

2.3.2.2 Tolérance aux pannes pour le plan de contrôle

SDN étant une architecture logique centralisée, qui repose sur le contrôleur pour mettre à jour les politiques et prendre des mesures lorsque de nouveaux flux sont introduits dans le réseau ; la fiabilité du plan de contrôle est d'une importance critique. Le mécanisme le plus fondamental pour récupérer les défaillances du plan de contrôle est l'approche de «réplication de sauvegarde primaire», où les contrôleurs de sauvegarde reprendront le contrôle du réseau en cas de défaillance des contrôleurs principaux. Deux problèmes doivent être résolus pour prendre en charge les schémas de réplication dans SDN.

1. Protocoles de coordination entre les contrôleurs principal et de secours : le protocole OpenFlow offre la possibilité de configurer un ou plusieurs contrôleurs de sauvegarde, mais OpenFlow ne fournit aucun mécanisme de coordination entre le contrôleur principal et les sauvegardes. Ainsi, les protocoles de coordination sont nécessaires et sont capables d'effectuer la coordination entre les contrôleurs pour maintenir la sauvegarde cohérente avec le contôleur principal. Ce qui mettra le réseau dans un état sûr avec une surcharge minimale imposée aux hôtes et aux commutateurs [39].
2. Déploiement du contrôleur de sauvegarde : des contrôleurs sont préparés à l'avance pour assurer la fonction de remplacement du contôleur principal en cas de panne afin de maximiser la fiabilité des réseaux [56].

2.3.3 Mise à jour de nouvelles politiques

Dans cette section, nous nous concentrons sur les changements planifiés (tels que les modifications des règles de politique réseau) au lieu des événements imprévus (tels que les défaillances de l'élément / lien réseau). Les opérations de mise à jour générales sont implémentées comme suit : chaque paquet ou flux est identifié lors de la mise à jour du réseau de l'ancienne stratégie vers la nouvelle stratégie sur plusieurs commutateurs, puis chaque paquet ou flux individuel est garanti d'être géré par l'ancienne stratégie ou la nouvelle mais pas par la combinaison des deux [38]. Il existe deux types de cohérence :

- Cohérence par paquet : signifie que chaque paquet traversant le réseau sera traité selon une configuration réseau unique.

- Cohérence par flux : signifie que tous les paquets du même flux seront gérés par la même version de la stratégie. Par conséquent, l'abstraction par flux conserve toutes les propriétés de chemin. Ces propriétés sont exprimées par les ensembles de paquets appartenant au même flux qui transitent par le réseau.

2.3.4 Analyse du trafic SDN

Dans cette section, nous discutons des outils de surveillance de réseau actuels pour la gestion du réseau, la vérification de réseau et le débogage dans les architectures SDN.

La surveillance est cruciale pour la gestion du réseau. Les applications de gestion nécessitent des statistiques précises et opportunes sur les ressources du réseau à différents niveaux d'agrégation (tels que le flux, les paquets et les ports). Les réseaux programmables basés sur le flux, tels

que les SDN, doivent surveiller en permanence les mesures de performances afin d'adapter rapidement les règles de transfert en réponse à l'évolution de la charge de travail.

De nombreuses architectures SDN utilisent les outils de surveillance de réseau basés sur les flux existants des réseaux IP traditionnels. Par exemple, le plus répandu est NetFlow [8] de Cisco, qui utilise des méthodes installées sur les commutateurs en tant que modules spéciaux pour collecter des statistiques de trafic complètes ou échantillonnées et les envoyer à un collecteur central. Un autre sFlow [16] de InMon, qui utilise un échantillonnage basé sur le temps pour capturer les informations de trafic. Par conséquent, les solutions suivantes recherchent des mécanismes de surveillance plus efficaces afin d'atteindre à la fois une grande précision et faible surcharges.

1. PayLess[25] : est un cadre de surveillance basé sur des requêtes pour SDN où les commutateurs sont interrogés périodiquement sur chaque flux actif. L'objectif de PayLess est de fournir une *API RESTful* flexible pour la collecte des statistiques de flux à différents niveaux d'agrégation (tels que le flux, le paquet et le port). Cette méthode permet d'effectuer une collecte d'informations très précises en temps réel sans encourir une surcharge importante du réseau.
2. FlowSense[59] : est une méthode de surveillance basée sur la poussée pour analyser les messages de contrôle entre le contrôleur et les commutateurs. Cette méthode utilise les messages du contrôleur pour surveiller et mesurer l'utilisation du réseau telle que la bande passante consommée par les flux traversant la liaison, sans induire de surcharge supplémentaire.

L'aperçu comparatif des différentes solutions de surveillance est résumé dans le Tableau 2.2.

Approches proposées	Description	Technologie de gestion du trafic	Analyse
PayLess[25]	Surveillance basée sur les requêtes	Interrogation adaptative basée sur un algorithme de collecte de statistiques de flux à fréquence variable.	- Haute précision et surcharge élevée dans un court intervalle d'interrogation minimum. - Faible précision et faible surcharge dans un intervalle d'interrogation minimum important.
FlowSens [59]	Surveillance passive basée sur la poussée.	Utilisation des messages Packet In et Flow Removed dans les réseaux OpenFlow pour estimer l'utilisation par liaison de flux afin de réduire la surcharge de surveillance.	- Haute précision et faible surcharge par rapport à la méthode d'interrogation.

TABLE 2.2 – Aperçu qualitatif des différentes solutions de surveillance.

2.3.4.1 Vérification des invariants du réseau

La vérification des invariants du réseau - c'est-à-dire la pratique générale consistant à vérifier que la configuration du réseau proposée (représentée par les tables de flux des commutateurs) obéit à divers invariants importants (comme aucune boucle de routage). Cette vérification est une tâche importante dans les réseaux SDN.

VeriFlow [32] est un outil de vérification en temps réel dans les réseaux SDN. Il est conçu comme un proxy résidant entre le contrôleur et les commutateurs du réseau pour surveiller toutes les communications dans les deux sens et vérifier dynamiquement les violations invariantes à l'échelle du réseau à mesure que chaque règle de transfert est insérée. La latence de vérification est de quelques millisecondes pour obtenir des réponses en temps réel.

2.3.4.2 Débogage des erreurs de programmation

Les langages utilisés pour programmer les réseaux actuels manquent de fonctionnalités modernes. Ils sont généralement définis au faible niveau d'abstraction fourni par le matériel sous-jacent et ne parviennent pas à fournir un support même rudimentaire pour la programmation modulaire. En conséquence, les programmes réseau ont tendance à être compliqués, sujets aux erreurs et difficiles à maintenir.
NICE [40] est un outil efficace et systématiquement technique, qui est une combinaison de vérification de modèle et d'exécution symbolique pour découvrir efficacement les violations des propriétés de correction à l'échelle du réseau en raison de bugs dans les programmes du contrôleur. En utilisant NICE, le programmeur OpenFlow peut être chargé de vérifier les propriétés d'exactitude génériques.

2.4 Les outils de gestion de trafic existants pour les réseaux SDNs OpenFlow

OpenFlow est une spécification de protocole qui décrit la communication entre les commutateurs OpenFlow et un contrôleur OpenFlow. Tout comme le chapitre précédent présentait des normes et des propositions qui étaient des précurseurs de SDN, voyant SDN traverser une période de gestation, l'arrivée d'OpenFlow est le point de naissance de SDN. En réalité, le terme SDN n'est entré en vigueur qu'un an après qu'OpenFlow a fait son apparition sur la scène en 2008, mais l'existence et l'adoption d'OpenFlow par les communautés de recherche et les fournisseurs de réseaux ont marqué un changement radical dans les réseaux, que nous sommes toujours témoin.

Pour les raisons identifiées dans le chapitre précédent, OpenFlow a été développé et conçu pour permettre aux chercheurs d'expérimenter et d'innover avec de nouveaux protocoles dans les réseaux actuels. La spécification OpenFlow a encouragé les fournisseurs à implémenter et à activer OpenFlow dans leurs produits de commutation pour le déploiement dans les réseaux des campus universitaires. De nombreux fournisseurs de réseaux ont implémenté OpenFlow dans leurs produits.

Dans cette section nous présentons quelques outils de gestion de trafic pour les réseaux SDN dans l'industrie et dans le milieu académique.

2.4.1 Solutions industrielles

Nous présentons dans cette partie l'état de l'art de quelques outils de gestion du trafic pour les réseaux SDN dans l'industrie. L'ensemble de ces outils est résumé dans le Tableau 2.3.

- SWAN [29] est un WAN piloté par logiciel (SWAN) proposé par Microsoft, qui utilise des règles de stratégie pour permettre aux WAN inter-centres de données de transporter beaucoup plus de trafic pour des services de priorité plus élevée, tout en maintenant l'équité entre des services similaires. Classiquement, le WAN est exploité en utilisant la gestion du trafic MPLS basée sur le routage ECMP, qui peut propager le trafic sur un certain nombre de tunnels entre les paires de routeurs d'entrée-sortie. Cependant, cette approche donne une efficacité très faible en raison du manque de vue globale au niveau des routeurs / commutateurs périphériques. Dans ce cas, l'allocation de ressources gourmandes doit être effectuée pour un flux en utilisant le chemin le plus court avec la capacité disponible. Pour résoudre les problèmes ci-dessus, SWAN exploite la vue globale du réseau activée par le paradigme SDN pour optimiser les politiques de partage de réseau, ce qui permet au WAN de transporter plus de trafic et de prendre en charge un partage flexible à l'échelle du réseau.

- ADMCF-SNOS[29], le cadre de calcul adaptatif dynamique à chemins multiples pour les systèmes d'exploitation de réseau intelligent (ADMCF-SNOS) de Shannon Lab de Huawei utilise son système d'exploitation de réseau intelligent (SNOS) pour fournir un système intégré de contrôle et de gestion des ressources pour grands systèmes de réseau distribués à commande centrale ou à couplage lâche. Les applications de gestion sont construites au-dessus du contrôleur, amélioré par des API dynamiques orientées ressources. L'une de ces applications était l' *Adaptive Dynamic Multi-path Computation Framework* (ADMCF). Il a été conçu comme un cadre de solution ouvert et facilement extensible qui peut fournir l'infrastructure et les algorithmes nécessaires pour la collecte de données, l'analyse et divers algorithmes d'optimisation.

- Le routage dynamique pour SDN [53] proposé par Bell Labs, résoud le problème d'optimisation du routage en utilisant le schéma d'approximation du temps entièrement polynomial *Fully Polynomial Time Approximation Scheme*(FPTAS). Le problème d'optimisation vise à trouver les routes optimales pour les flux de trafic réseau de telle sorte que le retard et la perte de paquets au niveau des liaisons soient minimisés, définissant ainsi comment minimiser l'utilisation maximale des liaisons dans le réseau.

Approches proposées	Description	Technologie de gestion du trafic	Une analyse
SWAN de Microsoft [29]	Un WAN piloté par logiciel (SWAN) pour les WAN inter-centres de données.	-Utiliser deux types de politiques de partage : (a) les différentes classes de classement du trafic, et (b) la même priorité de trafic avec le principe d'équité max – min.	-98% du débit réseau maximal autorisé, contre 60% dans le WAN compatible MPLS.
ADMCF-SNOS de Huawei[33]	Un système intégré de gestion et de contrôle des ressources pour les grands systèmes de réseau distribués à contrôle central ou à couplage lâche.	-Utilisation de l'Adaptive Dynamic Multi-path Computation Framework (ADMCF) pour fournir l'infrastructure et les algorithmes nécessaires pour la collecte de données, l'analyse et divers algorithmes d'optimisation. -Utilisation d'algorithmes de routage hybride statique et dynamique pour calculer le routage optimal afin de fournir un routage hybride presque optimal et économe en ressources que des schémas de routage explicites à destination.	Le routage basé sur SDHR surpasse un routage explicite environ 95% d'espace TCAM économisé au débit normalisé d'environ 70.
Routage dynamique pour SDN de Bell Labs [53]	Un algorithme de contrôle de routage optimisé pour SDN.	-Utilisation du schéma d'approximation du temps entièrement polynomial (FPTAS) pour résoudre le problème d'optimisation du contrôleur SDN qui minimise l'utilisation maximale des liaisons dans le réseau.	Le routage basé sur FPTAS surpasse un routage OSPF standard dans les SDN.

TABLE 2.3 – Aperçu qualitatif de la gestion du trafic dans les outils industriels existants pour les réseaux SDN-OpenFlow.

2.4.2 Solutions académiques

Avec l'augmentation du nombre d'utilisateurs dans le réseau et en raison des ressources disponibles limitées pour répondre aux exigences de qualité de service de ces utilisateurs, l'un des points clés à prendre en compte est le problème de gestion du trafic. Ce problème consiste (dans le cas des solutions traitées) à répartir les ressources disponibles de façon optimale afin de maximiser l'efficacité et la fiabilité des ressources réseau disponibles. Cette gestion est effectuée au niveau du contrôleur basée sur des informations locales et des statistiques réseau.

Dans cette partie, on s'intéresse aux techniques de gestion et d'optimisation des flux. On invoquera les algorithmes de routage qui ont été traités dans [58] [21] [49] [60] parceque ces protocoles sont représentatifs de la problématique que nous traitons dans ce projet. On clôturera cette partie avec un tableau récapitulatif énumérant les avantages et les insuffisances de ces algorithmes ainsi que les métriques utilisés.

2.4.2.1 Le protocole DLPO [58]

Les auteurs de [58] proposent un algorithme d'optimisation de chemin DLPO *Dynamic Load Balanced Path Optimization*. Il est constituée de deux étapes :

- Étape 1 : Initialisation du chemin dynamique. Le choix du chemin temporaire en fonction de la bande passante disponible sur chaque lien, parmi tous les chemins possibles entre la source et la destination. Le chemin dont la bande passante disponible est la plus grande sera choisie comme chemin temporaire.
- Étape 2 : Optimisation. L'algorithme récupère les statistiques du réseau en utilisant le protocole OpenFlow. Si la charge des liaisons dans le réseau est déséquilibrée $\partial(t) > \partial^*$, l'algorithme d'optimisation sera déclanché. Avec :

$$\partial(t) = \frac{1}{P} \sum_{i=T-P}^{T} (\frac{1}{N} \sum_{i=1}^{n} (\bar{load}(t) - load_i(t))^2); \tag{2.1}$$

Cette formule permet de calculer la valeur moyenne de la variance des charges de liaisons au cours d'une période t.
N : Le nombre de liens dans le réseau.
$\bar{load}(t)$: charge moyenne de toutes les liaisons au temps t dans un réseau.
$load_i(t)$: charge du lien i au temps t
T : temps courant.
$\partial(t)$: paramètre de detection de l'équilibrage de charge.
Lorsque $\partial(t) > \partial^*$, le DLPO est déclanché :

- DLPO multi-liens : changer les chemins des 10 % des liens les plus surchargés vers des liens avec la bande passante la plus grande disponible.
- Si le premier algorithme échoue (la surcharge est toujours présente), le second : DLPO à liaison unique sera déclanché. Il redirige le flux le plus important transitant sur une liaison surchargée vers un chemin dont la bande passante libre disponible est maximal.

2.4.2.2 Le protocole de routage sensible à la congestion [21]

Les auteurs de [21] proposent un algorithme efficace pour éviter la congestion dans les réseaux SDNs en redirigeant le moins de flux possible vers d'autres chemins non encombré afin de réduire la surcharge du réseau, Ils proposent un algorithme qui s'adapte à deux situations :

- Situation 1 : Un algorithme de recherche de chemins est effectué pour trouver l'ensemble de tous les chemins possibles P_i de la source à la destination. Avec le calcul de la charge acceptée par chaque chemin sans que ce chemin ne soit encombré à l'aide des deux équations suivantes :

$$C_{ji} = \{\cup(l_{ji}) * l_{ji}\_capacity(l_j \in L_i)\}; \tag{2.2}$$

$$Acceptable\_Load\_P_i = \min\{(0,7 * l_{ji}\_capacity) - C_{ji}\}; \tag{2.3}$$

  Où :
  - C_{ji} : Le coût de chaque lien L_i dans le chemin P_i
  - $\cup(l_{ji})$: L'utilisation de chaque lien l_j appartenant à l'ensemble des liens L_i dans le chemin P_i

  Ce flux est acheminé vers le chemin le plus court si ce chemin n'est pas encombré. Sinon, l'algorithme essaie de trouver un chemin qui peut accepter la charge de ce flux (chemin de sauvegarde).
- Situation 2 : Lorsqu'un commutateur est proche de l'encombrement (un seuil), l'algorithme choisit le nombre maximum de flux possible pour résoudre l'encombrement et les redirige vers les meilleurs chemins de sauvegarde en se basant sur l'équation ci-dessous :

$$Shift\_Need = Link\_load - (0,7 * Link\_capacity); \tag{2.4}$$

  Où :
  $Shift\_Need$: la charge totale qui doit être déplacer de la liaison encombrée vers un chemin non encombré.
  $Link\_load$: le taux de transmission moyen d'un port de sortie du commutateur.

2.4.2.3 Le protocole CAMOR [49]

Les auteurs [49] ont proposé une solution de routage optimal multivoie pour la prise en charge du problème de congestion. La technique proposé combine le routage par trajets multiples et des mécanismes adaptés au problème de congestion pour calculer le chemin optimal pour tout nouveau flux de données. La procédure suivie décrite ci-apès :

- Utilisation de l'algorithme de Dijkstra pour calculer tous les chemins les plus court à coût égale (bande passante) entre la source et la destination.
- Choisir les chemins optimaux parmi tous les chemins P_i séléctionnés où les liens ne sont pas congestionnés et le débit de la liaison satisfait l'exigence du débit de l'application : calculer la congestion, la charge et le débit disponibles. Pour cela quatre équations ont été utilisées.

$$L_i(i = 1, 2, ...n); \forall L_i \in P_n; \tag{2.5}$$

$$L_k = \sum_{i=1}^{n}(K_i); \tag{2.6}$$

$$L_l = \sum_{i=1}^{n}(l_i); \tag{2.7}$$

$$L_t^{P_i} = \frac{L_k - L_l}{L_i}; \tag{2.8}$$

- L'équation (2.5) définit le nombre total de liens L_i appartenant à un l'ensemble P_i des plus court chemins.
- L'équation (2.6) calcule la capacité totale L_k des liaisons.
- L'équation (2.7) calcule la charge totale L_i des liaisons.
- L'équation (2.8) calcule le débit total disponible $L_t^{P_i}$.

2.4.2.4 Le protocole d'équilibrage de charge [60]

L'utilisation des ressources réseaux d'une manière efficace et optimale est primordiale, pour satisfaire les utilisateurs finaux et assurer une disponibilité des services demandés.
Pour cela les auteurs de l'article [60] ont proposé une méthode permettant la satisfaction des besoins, en terme de ressource, des utilisateurs réseau. En proposant un shéma d'équilibrage de charge efficace basé sur l'architecture SDN, en utilisant le temps de réponse en temps réel de chaque serveur mesuré par le contrôleur SDN. De plus, prouver l'efficacité de cette méthode comparant aux régimes traditionnels.

L'équilibrage de charge proposé par cet article est comme suit :

- Étape 1 : le contrôleur gère la demande de l'utilisateur : Après les utilisateurs lancent une demande de service au serveur, le contrôleur recevra également le paquet de service de la demande de cet utilisateur et sélectionnera le serveur avec un temps de réponse minimum ou stable selon les données du serveur obtenues. Le processus de sélection est expliqué comme suit :

$$T_{max} = \max\{T_{1,0}, T_{2,0}, T_{3,0}, \ldots, T_{n,0}\}; \tag{2.9}$$

$$T_{min} = \min\{T_{1,0}, T_{2,0}, T_{3,0}, \ldots, T_{n,0}\}; \tag{2.10}$$

  En utilisant la formule (2.9) et (2.10), nous obtenons la valeur maximale T_{max} et valeur minimale T_{min} des temps de réponse du serveur pour l'ensemble des serveurs actuels.
  Où, $T_{i,j}$ est le temps de réponse du ième serveur dans l'intervalle de temps j avant l'heure actuelle où chaque intervalle est t, et $Ti, 0$ est le temps de réponse actuel du ième serveur.
- Étape 2 : en fonction des T_{min} et T_{max} obtenus, nous calculons $|T_{min} - T_{max}|$. Si $|T_{min} - T_{max}| < \lambda$, exécuter l'étape 3 sinon exécuter l'étape 4. λ représente que les serveurs ont une charge similaire lorsque la réponse $|T_{min} - T_{max}|$ est de l'ordre de λ.
- Étape 3 : obtenir l'écart-type de chaque serveur (2.11) en terme de temps de réponse en calculant l'écart type de l'historique de m données et sélectionner le serveur avec l'écart-type minimum S_{min}, puis exécuter l'étape 5.

$$S_i = \sqrt{(T_{i,0} - \bar{T})^2 + (T_{i,1} - \bar{T})^2 + \ldots + (T_{i,m-1} - \bar{T})^2}; \tag{2.11}$$

  $\bar{T}$: représente la valeur moyenne de m données historiques. S_i : représente l'écart type des données historiques pour la ième serveur.
- Étape 4 : sélectionner le serveur avec T_{min}.
- Étape 5 : le contrôleur enverra la table de flux aux commutateurs selon le serveur sélectionné, les demandes des utilisateurs seront envoyées au serveur sélectionné.

Articles	Objectifs	Simulation	Avantage	Insuffisance	Métriques
[58]	Équilibrage de charge dynamique dans les réseaux SDNs	-Contrôleur : NOX. -Environnement : Mininet -Topologie : Fat-tree. -Bande passate de réf : 750kbps -Taille des flux partagés : 500M -Flux TCP.	-Meilleur degré d'équilibrage de charge. -Améliorer l'utilisation de la bande passante.	-La méthode n'a pas été mise en œuvre dans un véritable réseau à grande échelle	Bande passante.
[21]	Équilibrage de charge efficace évitant la sur utilisation des liens dans les environements SDNs	-Contrôleur : ONOS -Environnement : Mininet -Topologie :08 OVS,03 Hôtes, 01 serveur. -Trafic tcp	-Éviter la congestion. -Technique rapide et efficace	-Augmentation considérable du délai lors du choix du chemin de sauvegarde.	Débit -Taux de perte des paquets.
[60]	Schéma d'équilibrage de charge basé sur la réponse du serveur LBBSRT	-Contrôleur : Floodlight. -Topologie : 30 clients et des serveurs -Flux http.	-Bonne évolutivité -Facile à mettre en œuvre. -Temps de réponse minimum. -Utilisation moyenne des ressources.	-Faible disponibilité. -Présence des goulots. -La non prise en considération de l'économie d'énergie dans l'équilibrage de charge	Temps de réponse. -Utilisation du processeur.

TABLE 2.4 – Tableau comparatif entre les solutions citées

2.5 Conclusion

Dans ce chapitre, nous avons abordé le domaine de la gestion du trafic dans les réseaux SDNs, en commançant par expliquer en quoi il consiste et en définissant ses aspects. Afin de mieux comprendre comment agira la stratégie de gestion de trafic adoptée dans ce projet, un travail de recherche et de documentation a donné lieu à une liste de differentes classifications des stratégie de gestion du trafic existantes dans la littérature.

Ce chapitre a porté aussi sur quelques applications de la gestion du trafic dans les réseaux SDNs, dans le domaine industriel et académique. Nous avons traité quelques articles qui ont une relation avec notre stratégie de gestion du trafic. Nous avons conclu ce chapitre par une comparaison des différentes traveaux connexes ainsi que quelques observations.

Chapitre 3

Conception de la solution

3.1 Introduction

Dans ce chapitre, nous détaillons la conception de la solution qui vise à mettre en place une stratégie de routage et d'acheminement de flux réseau dans une architecture SDN.
Nous proposons d'intégrer une stratégie de routage qui facilite au contrôleur SDN la prise de décision la plus adéquate en temps réel basée sur les ressources réseau disponibles et la charge en trafic réseau.

Nous commençons par une présentation de la conception générale de la stratégie de routage proposée ainsi que son fonctionnement. Cette section sera suivie par une vue approfondie sur les différents modules constituants notre approche de routage ainsi que les algorithmes de surveillance des flux réseau et d'acheminement du trafic déployés au sein de l'architecture SDN. Nous détaillons la conception de notre approche de routage qui consiste à maintenir le bon fonctionnement du réseau en assurant l'acheminement le plus optimal du trafic entre deux entités finales/terminales en tenant compte du trafic en transit et les ressources réseaux disponibles.

Nous finirons par donner l'organigramme regroupant les modules constituants notre approche de routage et leurs interractions.

3.2 Conception générale

La stratégie de routage adoptée est conçue au niveau de la couche de contrôle SDN comme le montre la Figure 3.1. Elle comprend principalement un module de surveillance de trafic, une interface de communication et un module d'acheminement du trafic.

L'utilisation de SDN permet d'assurer les fonctionnalités de contrôle centralisé et de la vue globale. Sur la base des données fournies par le module d'acquisition d'informations globales qui appartient à l'interface de communication, nous concevons une stratégie de surveillance des flux. Le module de surveillance des flux comprend les statistiques sur les flux et les ports. La stratégie d'acheminement utilise les informations collectées par le module de surveillance pour calculer la bande passante, le délai de bout en bout et élire le chemin optimal. Des mises à jour sont effectuées au niveau des tables de flux afin d'acheminer le trafic du nœud source au nœud de destination. L'interface vers le sud entre la couche de contrôle et la couche de données utilise le protocole OpenFlow.

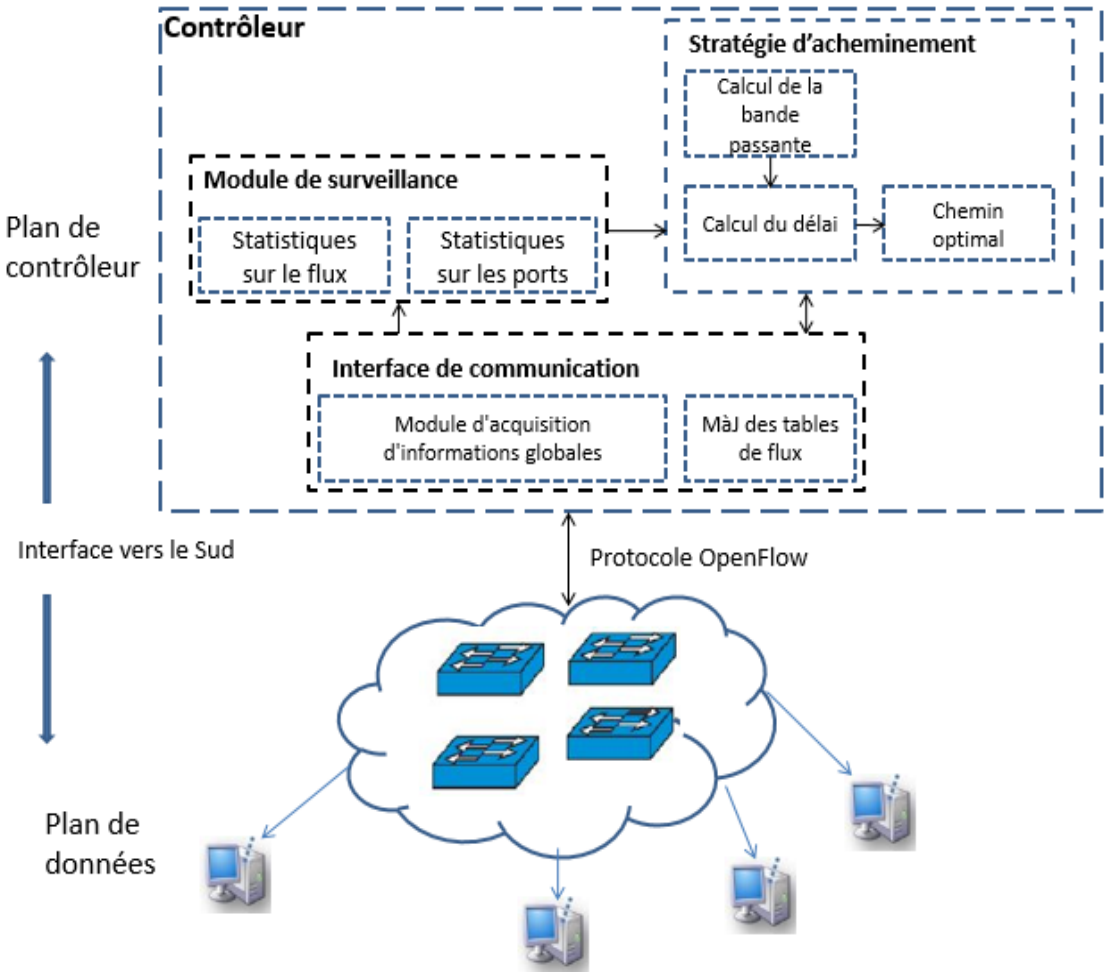


FIGURE 3.1 – Positionnement de la stratégie de routage du trafic réseau au niveau de l'architecture SDN.

3.3 Vue approfondie de la conception

Dans un environnement réseau réel, les ressources disponibles et les informations réseau de base ; notamment la topologie et les statistiques du trafic doivent être mesurés en temps réel pour permettre au contrôleur SDN la prise des décisions du routage les plus sûres et efficaces sur la base des informations collectées. La conception de notre stratégie de routage illustrée par la Figure 3.2 est basée sur l'implémentation des différents modules où chacun offre des fonctionnalités qui contribuent au choix du chemin optimal.

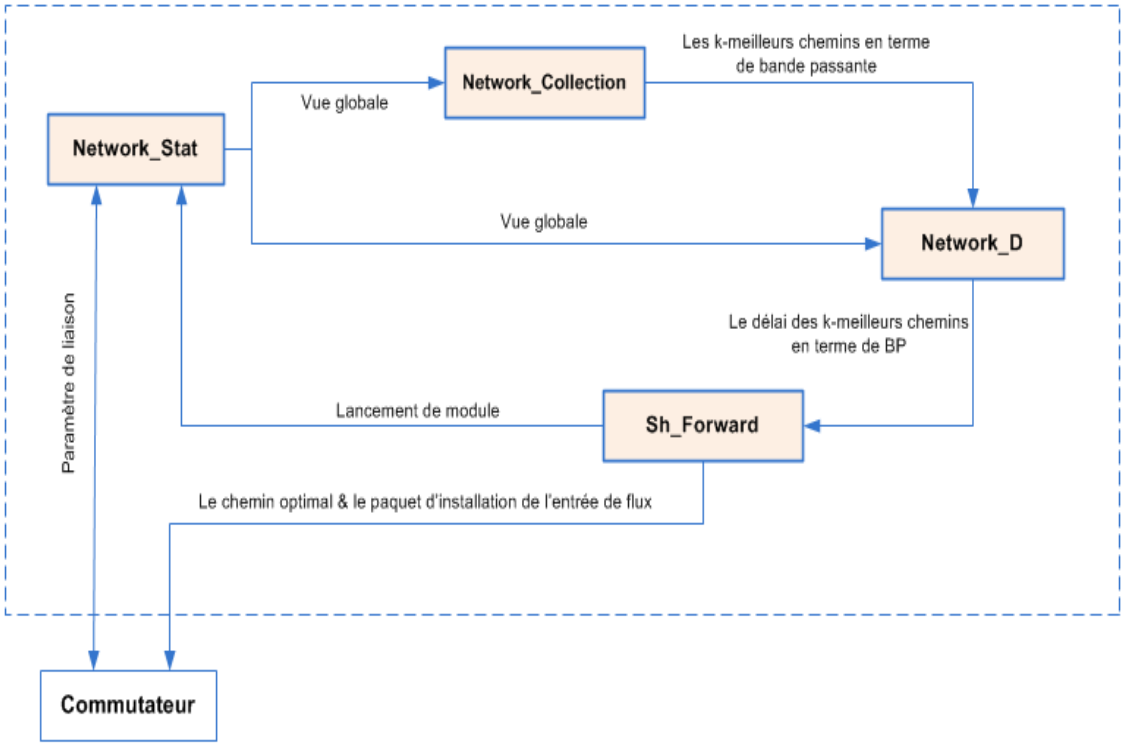


FIGURE 3.2 – Schéma détaillé de la stratégie de routage proposée au niveau du contrôleur SDN.

3.3.1 Le module Network_Stat

Il constitue le module de base de la couche de contrôle. *Network_Stat* permet de détecter les changements au niveau des ressources réseau en temps réel y compris les informations sur la topologie du réseau, les hôtes ainsi que les commutateurs. Ce module reçoit et traite l'état de connexion du commutateur, de l'hôte et les enregistre. Il permet ainsi le mappage des ports et des liaisons des commutateurs.

La phase de découverte se fait à travers les échanges des paquets de découverte et des paquets de réponses entre le plan de données et ce module afin de récupérer les informations sur l'ensemble des éléments constituant la topologie tels que les commutateurs réseau, les liaisons... etc.

Pour découvrir dynamiquement la topologie dans les réseaux SDN, le service de découverte de liens à l'intérieur des contrôleurs tire parti de la couche de liaison de données en utilisant le protocole LLDP *(Link Layer Discovery Protocol)* pour détecter les liens entre les commutateurs. Les informations LLDP sont envoyées par le contrôleur à un intervalle fixe, sous la forme d'une trame Ethernet. Le tableau 1 montre le format de la trame Ethernet utilisé dans LLDP [24].

DA et SA représentent l'adresse MAC de destination de multidiffusion et adresse MAC source respectivement. Chaque trame LLDP contient une donnée LLDPDU *(LLDP Data Unit)*.

Champ	Valeur/ Description	Longueur (octet)
DA	Adresse multicast LLDP	6
SA	Adresse MAC	6
Ether type	0x88cc	2
Data pad	LLDPDU	1 500
FCS	Chiffre de contrôle	4

TABLE 3.1 – Format de la trame LLDP[24]

L'étape de découverte est décrite par le pseudo code suivant :

Algorithm 1 get_topology()

Input : Discovery_Packet Output : List_switches, Link_to_port
`/* List_switches: liste des commutateurs du réseau. */`
`/* Link_to_port : liste des liens et Num ports */`
`/* receive : variable d'état */`

```
Send(Discovery_Packet) to all switches ;
Receive (Discovery_Packet_Reply) of switch ;
while receive do
    List_switches = Discovery_Packet_Reply.id_switch ;
    Link_to_port = (Discovery_Packet_Reply.n_port,
     Discovery_Packet_Reply.link) ;
    Receive(Discovery_Packet_Reply) from switch ;
end
```

Après la récupération de toutes les réponses et les informations concernant la topologie réseau, le module *Network_ Stat* traduit ces informations en un graphe dirigé afin de construire la vue globale du réseau pour faciliter la gestion et la manipulation des ressources. En se basant sur le graphe généré, ce module permet aussi de sélectionner tous les chemins possibles de la source à la destination à l'aide de l'algorithme ci-après :

Algorithm 2 get_paths()

Input : Graph, source, destination.
Output : paths. `// Fonctionnement`

```
for src ∈ Graph.nodes() do
    for dst ∈ Graph.nodes() do
        if ( source == src ) and ( destination == dst ) then
            All_path [ src ][ dst ] ← All_possible_paths(Graph, src, dst);
        end
    end
end
```

3.3.2 Le module Network_Collection

Le mécanisme d'acheminement des flux entre deux entités finales nécessite une surveillance active du trafic réseau. Afin d'obtenir des statistiques précises sur l'état du réseau nous adoptons la méthode de mesure active du trafic en utilisant le module *Network_Collection*. Ainsi le contrôleur envoie périodiquement des paquets de demande des statistiques sur le trafic au commutateur pour collecter et stocker les données reçues (bande passante en utilisation, vitesse de transfert, le temps de transfert des flux...) dans une structure de graphe.

Le module *Network_Collection* raporte les statistiques sur les ports et les statistiques sur les flux. Une fois les statistiques collectées, nous procédons au calcul de la bande passante libre, la vitesse de transfert des flux et le débit.

Sur la base des statistiques des ports, nous calculons la vitesse du trafic transitant le réseau. En fonction de la capacité et de la vitesse de transfert nous calculons la bande passante résiduelle. La méthode de calcul du débit et de la bande passante résiduelle est indiquée par les deux formules ci-dessous :

$$port\_speed_{ij} = \frac{now\_portbyte - pre\_portbyte}{period} \tag{3.1}$$

Où :
i : représente l'ID du commutateur.
j : représente le port du commutateur.
port_speed : représente la vitesse de transfert du j ème port du commutateur ID i.
now_portbyte : la valeur du trafic du port du commutateur au moment actuel.
pre_portbyte : la valeur du trafic du port du commutateur à la mesure précédente.
periode : Intervalle de temps entre deux mesures.

En soustrayant pre_portbyte de now_portbyte nous obtenons le trafic total de la période. Le calcul de la bande passante résiduelle s'effectue en soustrayant la vitesse de transfert de la capacité maximale du lien :

$$BPR_{ij} = capacity_{ij} - port\_speed_{ij} \tag{3.2}$$

Où :
BPR_{ij} : représente la bande passante restante du j ème port avec l'id du commutateur i.

Le contrôleur envoie de manière proactive des demandes d'informations de statistiques de trafic au commutateur pour obtenir une réponse, tout en surveillant l'événement de réponse d'informations de statistiques de trafic.

Après la réception de la réponse du commutateur, le contrôleur analyse le message de réponse des statistiques et envoie une réponse en fonction des informations reçues. Nous obtenons l'IP source, l'IP de destination, les données de flux et la durée d'envoi des flux qui sont enregistrés dans une structure de graphe.

En plus de la surveillance réseau, *Network_ Collection* permet de trouver la bande passante de chaque chemin parmi tous les chemins possibles de la source à la destination. Ceci est fait en se basant sur la structure de graphe et la formule condensée que nous proposons :

$$BPR_{Path} = \min\{BPR_{src,dst}\}; \tag{3.3}$$

La version étendue de l'équation précédente est donnée comme suit :

$$BPR_{Path} = \min\{BPR_{src,src+1}, \ldots, BPR_{dst-1,dst}\}; \tag{3.4}$$

Où :
$path \in paths$.
$n = \|paths_(src, dst)\|$.
src : le nœud source.
dst : le nœud de destination.
BPR_{path} : est la bande passante résiduelle du chemin.

L'obtention du minimum de bande passante résiduelle des liaisons permet de sélectionner la bande passante de tout le chemin de la source à la destination. Le pseudo code suivant traduit la formule (3.3) :

Algorithm 3 minimal_bandwidth

Input : Graph, path.
Output : minimal_bandwidth.

```
// N: le nombre de chemin
N ← length(path)
minimal_bandwidth = capacity
for i ∈ N do
    pre ← path[i];
    curr ← path[i + 1];
    bw ← Graph[pre][curr]['bandwidth'];
    minimal_bandwidth ← min(bw, minimal_bandwidth);
end
```

La formule (3.5) permet de trouver les k meilleurs chemins ayant une bande passante maximale parmi tous les chemins possibles entre deux noeuds du graphe.

$$Bestpath[j] \leftarrow Path[\max\{BPR_{paths}\}] \tag{3.5}$$

Où :
$Bestpath$: un tableau des k meilleurs chemins en terme de bande passante.
$paths \leftarrow allpaths - Bestpath[j]$.
BPR_{paths} : la bande passante résiduelle du chemin.
$j = \|1, k\|$.

Le pseudo code suivant traduit la formule (3.5).

Algorithm 4 Bandwidth_best_paths

```
Input : Graph, source, destination, k.
Output : k_best_paths.                          // k: le nombre de chemin à sélectionner
all_paths ← Network_Stat.get_paths(source, destination)
i = 0
for path ∈ all_paths do
    Bandwidth [ i ] ← minimal_bandwidth(Graph, path);
    i = i + 1 ;
end
i = 0
if length (all_paths) > 1 then
    while ( i < length (all_paths) − 1) and ( i < k ) do
        j = i + 1
        while j < length (all_paths) do
            if Bandwidth [ i ] < Bandwidth [ j ] then
                X ← Bandwidth[ i]
                Bandwidth [ i ] ← Bandwidth[ j]
                Bandwidth [ j ] ← X
                Path ← all_paths[ i]
                all_paths [ i ] ← all_paths[ j]
                all_paths [ j ] ← Path
            end
            j = j + 1
        end
        k_best_paths.add(all_paths[ i])
        i = i + 1
    end
end
```

3.3.3 Le module Network_D

Ce module permet le calcul du délai de bout en bout, et ceci en calculant les délais entre les liaisons internes de la topologie réseau afin de sélectionner les chemins ayant le plus petit délai.

Comme expliquer précédemment, les contrôleurs SDN existants utilisent le protocole LLDP pour découvrir la topologie global du réseau. Nous proposons une nouvelle approche de surveillance de la latence des liens basée sur LLDP. Les mesures de la latence des liens d'un réseau nous permet de trouver rapidement le chemin avec la latence la plus faible pour un flux donné.

Étape 1 : Le calcul du délai de chaque liaison.

Afin de sélectionner les chemins ayant le plus petit délai, nous calculons d'abord le délai de chaque liaison et nous enregistrons ce délai sur le graphe généré par le module *Network_stat*. Pour ce faire, *Network_D* se base sur le protocole LLDP. Ce protocole s'exécute en plusieurs étapes comme le montre la Figure 3.3

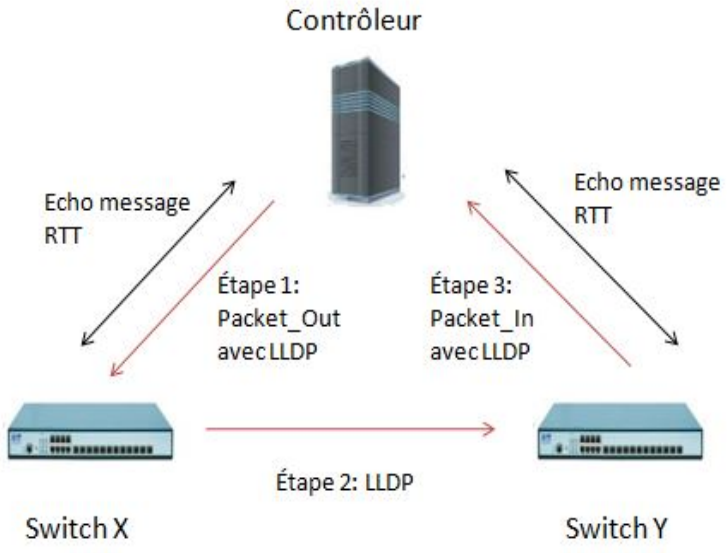


FIGURE 3.3 – Diagramme schématique de mesure du retard de liaison.

Initialement, le contrôleur envoie un message *Packet_ Out* horodaté au commutateur X avec la charge utile d'un paquet LLDP spécifique au contrôleur. La charge utile du paquet LLDP contient DPID *(DataPath Identity)* et l'ID du port de sortie du commutateur X. À la réception du paquet LLDP, le commutateur X le diffuse sur tous les autres ports.

Lors de la réception du paquet LLDP de diffusion envoyé par le commutateur X, le commutateur voisin Y signale le paquet LLDP entrant au contrôleur avec l'ID du port d'entrée et le DPID du commutateur Y via un message *Packet_ In*. Lors de la réception du message d'entrée de paquet du commutateur Y, le contrôleur peut détecter une liaison unidirectionnelle du commutateur X au commutateur Y. Le contrôleur peut récupérer le temps écoulé du paquet LLDP afin de mesurer le délai de la liaison entre les commutateurs (X,Y).

Le contrôleur envoie des Echos messages aux commutateurs X et Y. Nous définissons le retard de la liaison interne entre 2 commutateurs quelconque avec l'équation :

$$t_{srctodst} = t_{measure} - \frac{1}{2} t_{Echo_{ctltosrc}} - \frac{1}{2} t_{Echo_{ctltodst}} \tag{3.6}$$

Où :
$t_{measure}$: représente le retard du paquet LLDP vue au contrôleur depuis le moment où il est envoyé jusqu'au moment où il revient.
$t_{Echo_{ctltosrc}}$: la latence du paquet Echo transmis entre le contrôleur et le commutateur emetteur.
$t_{Echo_{ctltodst}}$: la latence du paquet Echo transmis entre le contrôleur et le commutateur recepteur.

Le pseudo code suivant permet le calcul de la latence entre le contrôleur et les commutateurs source et destination.

Algorithm 5 get_Latency ()

```
Input : List_switch.
Output : Latency.                          // Packet.time: l'heure d'envoi du packet.
for switch_id ∈ List_switch do
    Send( Echo_request ) to switch_id ;
end
for switch_id ∈ List_switch do
    Packet ← Recieve(Echo_reply)fromswitch_id
    time_now ← current_time();
    Latency [ switch_id ] ←time_now – Packet.time
end
```

Le pseudo code ci-dessous traduit la formule (3.6) permetant le calcul du délai :

Algorithm 6 get_delay()

```
Input : Latency, Graph.
Output : Graph.                            // Packet_delay_lldp: délai packet_lldp.
for src ∈ Graph.nodes() do
    for dst ∈ Graph.nodes() do
        fwd_delay ← Graph[src][dst]['Packet_delay_lldp'];
        src_latency ← Latency[src];
        dst_latency ← Latency[dst];
        final_delay ← fwd_delay − 1/2(src_latency + dst_latency);
        Graph [ src ] [ dst ] [ 'delay' ] ← final_delay;
    end
end
```

Étape 2 : La sélection du chemin ayant le plus petit délai.

Par la suite, nous procédons au choix du chemin ayant le plus petit délai aprés avoir calculé le délai de bout en bout des N chemins. Pour ce faire, nous nous sommes basés sur les deux algorithmes suivants :

Algorithm 7 total_delay_path ( )

```
Input : Graph, path.
Output : delay_path.
N= length(path) ;                          // N: le nombre de chemins.
for i ∈ N do
    pre ∈ path[i];
    curr ∈ path[i + 1];
    delay ← Graph[pre][curr]['delay'];
    delay_path ← delay_path + delay;
end
```

Algorithm 8 Delay_best_path ( )

Input : paths.
Output : delay_best_path.

```
delay_min ← total_delay_path(paths[0])
delay_best_path ← paths[0]
j = 1
while ( j < length ( paths ) ) do
    Delay ← total_delay_path(paths[j])
    if ( delay_min > Delay ) then
        delay_min ← Delay
        delay_best_path ← paths[j]
    end
end
```

3.3.4 Le module Sh_Forward

C'est le module principal de la stratégie de routage proposée. Il permet la sélection du chemin optimal en se basant sur les critères de sélection précédemment présentés à savoir la bande passente résiduelle et le retard des liens.

Le contenu des modules : *Network_Stat*, *Network_Collection* et *Network_D* sont chargés en tant que service du module *Sh_Forward* lors de son exécution. A ce niveau, le chemin de transmission optimal entre une source et une destination est sélectionné en se basant sur les statistiques collectées et les ressources disponibles.

Lorsque le commutateur reçoit le paquet de données, il analyse le paquet et recherche la correspondance entre les informations contenues dans ce dernier et sa table de flux. Si la correspondance réussit, le paquet est transmit conformément à la table de flux.
Si la correspondance échoue le paquet est envoyé au contrôleur. Le contrôleur prend la décision de routage la plus optimale comme suit :

Il fait appel à l'interface de communication *Network_Stat* pour obtenir les informations sur la topologie réseau. Par la suite, nous déterminons tous les chemins possibles entre une source et une destination. Les K meilleurs chemins en terme de bande passante sont sélectionnés par le module *Network_collection*. En se basant sur ces k chemins, le module *Network_D* choisit le chemin ayant le plus petit délai de transfert. Ce chemin est considéré comme le chemin optimal.

En cas d'égalité des chemins en terme de délai de transfert, nous optons pour le chemin le plus court parmi ces chemins égaux. Ce processus est effectué afin d'assurer une transmission la plus optimale possible.

Le pseudo code suivant récapitule le fonctionnement de base de ce module.

Algorithm 9 get_best_path ( )

Input : packet_out.
Output : best_path.

```
    // get_switch_id: c'est une fonction qui permet de récupérer l'id du switch
 emetteur et recepteur.
1  source ← packet_out.src_ip
2  destination ← packet_out.dst_ip
3  source_id ← get_switch_id(source)
4  destination_id ← get_switch_id(destination)
5  Bandwidth_path ← Network_Collection.Bandwidth_best_paths(source_id,
    destination_id, k)
6  Delay_path ← Network_D.Delay_best_path(Bandwidth_path)
7  if ( length ( Delay_path ) > 1 ) then
8  |  min_hop = length ( Delay_path [ 0 ] )
9  |  best_path ← Delay_path[0]
10 |  while ( j < length ( Delay_path ) ) do
11 |  |  if ( length ( Delay_path [ i ] ) < min_hop ) then
12 |  |  |  min_hop = length ( Delay_path [ i ]
13 |  |  |  best_path ← Delay_path[i]
14 |  |  end
15 |  end
16 end
```

Après la sélection du chemin optimal comme expliqué ci-dessus, le module *Sh_Forward* effectue des mises à jour au niveau des tables de flux des commutateurs afin d'installer les entrées de flux. Ces mises à jours sont basées sur l'envoi des paquets *flow_mod* à tous les commutateurs qui appartiennent à ce chemin comme mentionné dans le pseudo code suivant.

Algorithm 10 Add_flow_entry ( )

Input :best_path.

```
                              /* get_port: récupérer les ports des commutateurs. */
   /* time_out: temps d'expiration de l'entrée de flux dans la table de flux. */
            /* match : l'ensemble des information nécessaires, port d'entrée. */
                                 /* src, dst, le protocole utilisé ( TCP/UDP ). */
                               /* instruction :l'action effectuée sur le flux. */
1  N ← length(best_path) ;
2  for i ∈ N − 1 do
3  |  id_switch ← best_path[i] ;
4  |  port ← get_port(best_path[i−1], best_path[i]) ;
5  |  src_port ← port
6  |  dst_port ← Next_port
7  |  match ← (src_port, ip_src, ip_dst, proto)
8  |  flow_mod ← (id_switch, priority, time_out, match, instruction) ;
9  |  Send (flow_mod ) to best_path [i] ;
10 end
```

3.4 Vue globale de l'approche adoptée

Notre stratégie de routage exploite les caractéristiques des réseaux SDN et l'état du réseau afin de prendre des décisions adéquates pour assurer un routage optimisé.

Premièrement, nous avons procédé à l'acquisition d'informations réseau, y compris les informations sur la topologie du réseau, les hôtes et commutateurs. Par la suite, nous avons procédé à la conception globale du mécanisme de surveillance du trafic, y compris les statistiques sur les flux et les ports. Et enfin, nous avons développé le module de routage et d'acheminement du trafic. L'organigramme ci-dessous synthétise notre stratégie précédemment détaillée Figure 3.4

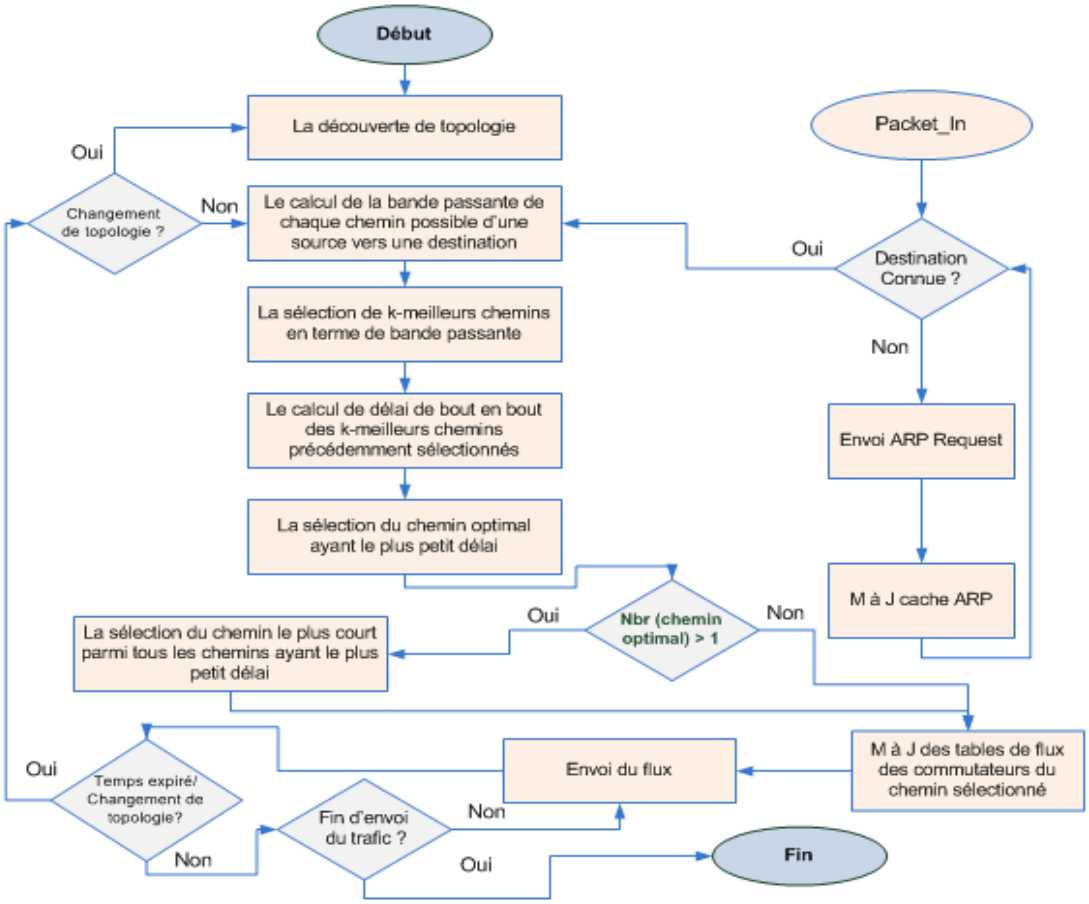


FIGURE 3.4 – Déroulement de la stratégie de routage proposée.

3.5 Conclusion

Dans ce chapitre nous avons décrit en détail la conception de notre stratégie de routage réseau. Notre approche s'intéresse principalement à la surveillance et la collecte de la quantité de trafic circulant sur le réseau afin de choisir les chemins optimaux pour le transfert des données. De la création de la vue globale qu'on a nommé « *get_ topology* », vers l'étape de routage, incluant tous les modules de la Figure 3.2, c'est-à-dire : la collecte, l'analyse (ressources réseaux disponibles et la quantité du trafic en transit) et enfin l'acheminement des données en tenant en compte les caractéristiques pertinentes du réseau.
Dans le chapitre suivant, nous présentons notre évaluation de performances à travers différentes simulations.

Chapitre 4

Implémentation de la solution et test de performances

4.1 Introduction

Ce chapitre est rédigé dans l'optique d'offrir un manuel de mise en place d'une architecture SDN virtualisée supportant le routage qui est basé sur le trafic en transit et les ressources réseau disponibles. Le travail documenté ci-dessous vise à expliquer la manière dont nous avons intégré concrètement la stratégie de routage au sein du réseau SDN.

Afin de montrer l'impact de l'intégration proposée, une série de mesures de performance a été réalisée à l'aide de l'outil standard **Iperf** pour mesurer le rendement du réseau. Ces simulations ont pour objectif d'étudier les performances de notre stratégie de routage en faisant varier plusieurs metriques, en vue d'analyser son comportement et de valoriser les paramètres utilisées dans la décision de routage. Cela devra également montrer que la stratégie de routage adoptée peut coexister avec la pile de protocoles de réseau TCP/IP conventionnelle.

4.2 Spécifications et besoins de la solution

La technologie SDN est idéale pour les réseaux distribués, les solutions cloud, la mise en place d'une ferme de VM(*Virtual Machine*) , les services de partage de contenu ou la voix IP. Néanmoins, dans un environnement réel, ces réseaux génèrent beaucoup de trafic de données ce qui engendre des pertes de paquets et des dépassements de délais.

Dans ce contexte, nous avons proposé d'intégrer un mécanisme de routage qui facilite au contrôleur SDN la prise de décision de routage la plus correcte en temps réel, basée sur les ressources réseau disponibles et la charge en trafic réseau.

Afin de mettre en place la stratégie de routage proposée, plusieurs composants nous ont été nécessaires. La suite de cette section a pour but de définir les outils qui constituent la solution.

4.2.1 Contrôleur

Après une recherche approfondie, parmi les divers contrôleurs Open source (Annexe 1), nous

avons opté pour RYU car il s'avère plus modulaire et ce, dans le souci d'avoir une application facile à gérer et à exploiter. Il supporte aussi, coté SBI, toutes les versions d'OpenFlow et sa documentation est riche.

4.2.2 Le commutateur virtuel OVS

L'interconnexion des VMs et le transport des flux nécessitent l'utilisation des commutateurs. Etant dans un environnement SDN, ces commutateurs doivent présenter les SBIs adéquates. Il existe, sur le marché, plusieurs solutions adéquates à ce cas de figure ; parmi elles, VMware Virtual switch, Cisco Nexus 1000V et Open vSwitch. Dans un contexte de virtualisation et dans l'optique de mettre en place des topologies réseau maniables et compatibles avec Openflow, notre choix s'est porté sur Open vSwitch.

Open vSwitch (OVS) est une implémentation logicielle open source d'un switch ethernet avec la particularité d'être multicouche et distribuée [12]. Il permet de virtualiser des commutateurs ayant le même comportement que les commutateurs physiques.
OVS supporte les protocoles et les APIs de configuration réseau classique (ex CLI, 802.1q pour les VLAN , SNMP , NAT , IPSEC pour la sécurité etc.) ainsi que des APIs *southbound* (SBI) tels que Openflow et OVSDB (*Open Virtual Switch Data Base*) , ce qui en fait un outil adéquat pour notre solution. La Figure 4.1 situe OVS au sein d'une architecture SDN et montre comment se fait l'interconnexion à son niveau.

OVS est utilisé pour mettre en place l'infrastructure réseau qui sera chapotée par le contrôleur et qui interconnectera les noeuds finaux. Les commutateur OVS contiennent les tables de flux et peuvent dialoguer avec le contrôleur grâce à OpenFlow (pour le maintien des tables) et OVSDB pour la configuration. OVS peut servir à interconnecter des machines virtuelles. Il peut servir aussi à interconnecter des machines virtuelles avec d'autres machines ou équipements réseau physiques se trouvant sur des réseaux différents.

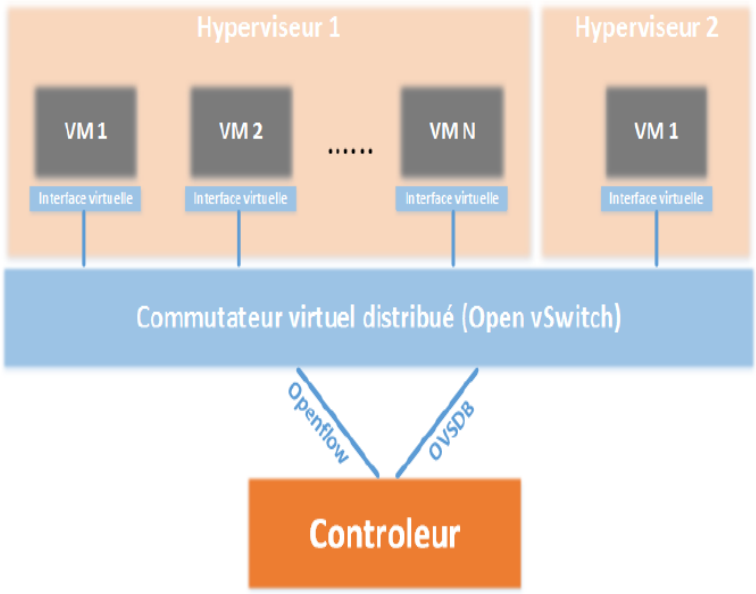


FIGURE 4.1 – Positionnement d'OVS au sein d'une architecture SDN [12].

4.2.3 Mininet

Mininet est un émulateur de réseau qui crée un réseau d'hôtes, commutateurs, contrôleurs et liens virtuels [7]. Les hôtes Mininet exécutent un logiciel réseau Linux standard et ses commutateurs prennent en charge OpenFlow pour un routage personnalisé très flexible au niveau des réseaux défini par logiciel (SDN). Nous avons opté pour cet émulateur car :

- Il permet d'avoir des topologie complexes, sans avoir besoin de câbler un réseau physique.
- Il comprend une CLI sensible à la topologie et à OpenFlow, pour le débogage ou l'exécution de tests à l'échelle du réseau.
- Il prend en charge des topologies personnalisées arbitraires et inclut un ensemble de base de topologies paramétrées.
- Il fournit également une API Python simple et extensible pour la création et l'expérimentation des réseaux.

4.2.4 Package Networkx

Networkx est un package de langage Python pour créer, exploiter et étudier la dynamique, la structure et les fonctions de réseaux complexes [11]. Il prend en charge la création de graphiques simples non orientés, de graphiques dirigés et de multi graphes de telle sorte que les nœuds peuvent être de plusieurs types de données.

Le package Networkx intègre plusieurs algorithmes de théorie des graphes, y compris l'algorithme du chemin le plus court de *Dijkstra*, DFS (*Depth First Search*) et l'algorithme Traversal BFS (*Breadth First Search*) [1]. Nous avons utilisé l'algorithme de Dijkstra dans notre stratégie de routage.

4.3 Mise en place de l'environnement

4.3.1 Le contrôleur

Étape 1 :Vérification des pré requis.

Pour installer RYU, nous avons opté pour une version récente d'ubuntu 18.04 64 bits, 4GO de RAM, et Python 2.7 Les commandes suivantes nous permettront de télécharger et d'installer python 2.7.

```
$ sudo apt-get update
$ sudo apt-get install build-essential  checkinstall
$ sudo apt-get install libreadline-gplv2-dev libncursesw5-dev libssl-dev libsqlite3-dev tk-dev libgdbm-dev libc6-dev libbz2-dev
```

```
$ cd /usr/src
$ sudo wget https://www.python.org/ftp/python/2.7.16/Python-2.7.16.tgz
$ sudo tar xzf Python-2.7.16.tgz
$ cd Python-2.7.16
$ sudo ./configure --enable-optimizations
$ sudo make altinstall
```

L'installation de RYU nécessite l'installation d'un gestionnaire de packages (pip). Pip est un système de gestion de packages standard utilisé pour installer et gérer des packages logiciels écrits en Python. L'installation du pip se fait par les commandes suivantes :

```
$ sudo apt-get update
$ sudo apt install python-pip
```

Étape 2 : Installation.

Nous téléchargeons RYU du site officiel, puis nous l'installons à partir des commandes suivantes :

```
$ git clone git://github.com/osrg/ryu.git
$ cd ryu
$ pip install
```

Étape 3 : Vérification de l'installation.

A ce niveau, nous avons terminé l'installation de RYU. Pour s'assurer de son bon fonctionnement, nous exécutons la commande ci-dessous :

```
sdn@ubuntu:~$ cd ryu
sdn@ubuntu:~/ryu$ PYTHONPATH=. ./bin/ryu-manager
loading app ryu.controller.ofp_handler
instantiating app ryu.controller.ofp_handler of OFPHandler
```

4.3.2 Mininet

Nous procédons à l'installation comme suit :

Étape 1 : Vérification des pré requis.

Nous installons git, qui est le système de contrôle de version de logiciel utilisé par le projet Mininet.

```
$ git clone git://github.com/osrg/ryu.git
$ cd ryu
$ pip install
```

Étape 2 : Installation.

Nous utilisons git pour télécharger le code source de Mininetà partir de github et nous installons sa version 2.2.0 à l'aide des commandes suivantes :

```
$ git clone git://github.com/mininet/mininet
$ cd mininet
$ git tag
$ ~/mininet/util/install.sh -a
```

Étape 3 : Vérification de l'installation.

Nous testons que l'installation a réussi en exécutant la commande suivante :

```
sdn@ubuntu:~$ sudo mn --test pingall
[sudo] password for sdn:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 6.658 seconds
```

Après cette étape, Mininet est bien installé et il est prêt à être utilisé.

4.3.3 Networkx

Étape 1 : Vérification des pré requis.

Avant d'installer une version récente, nous devons d'abord désinstaller la version standard.

```
$ pip uninstall networkx
```

Étape 2 : Installation.

Nous procédons à l'installation d'une version récente à l'aide des commandes :

```
$ git clone https://github.com/networkx/networkx.git
$ cd networkx
$ pip install -e .
```

Étape 3 : Vérification de l'installation.

Nous testons que l'installation a réussi et que le package networkx est prêt à être importé dans nos scripts en exécutant la commande suivante :

```
sdn@ubuntu:~$ python
Python 2.7.17 (default, Apr 15 2020, 17:20:14)
[GCC 7.5.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import networkx
>>>
```

4.4 Mise en place de la solution

Dans cette section, nous allons montrer les étapes et la manière avec laquelle nous avons procédé pour réaliser et implémenter notre stratégie de routage précédemment expliquée dans le chapitre3.

Les modules de notre stratégie de routage sont programmés en utilisant le langage Python supporté par le contrôleur RYU. Aussi, nous avons eu besoin d'autres fonctionnalités et protocoles offerts par le contrôleur que nous expliquons ci-dessous.

4.4.1 Initialisation des packages

A ce niveau, nous initions les scripts python et les classes nécessaires au fonctionnement de notre stratégie de routage, qui sont :

- **ofproto_v1_3** : Ce script permet de charger le contenu du répertoire ofproto figurant dans l'emplacement ryu/ofproto. Il permet d'indiquer la version du protocole OpenFlow utilisée dans cette communication et les attributs communs qu'OpenFlow doit utiliser, tels que le port d'écoute, la longueur de l'en-tête et le format d'encapsulation de l'en-tête.
- **ofp_event** : Il assure la communication via le système d'événements entre les modules constituant notre solution. Chaque fois que le commutateur établit une connexion avec Ryu, il instanciera un objet Datapath pour gérer la connexion. Dans l'objet Datapath, les données reçues seront analysées dans les messages correspondants, puis converties en événements correspondants, puis publiées. Le module enregistré avec l'événement correspondant recevra l'événement, puis appellera le gestionnaire correspondant pour gérer l'événement. Ce script se trouve au niveau de ryu/controller.

- **set_ev_cls** : Ce script spécifie la classe d'événements prenant en charge le message reçu et l'état du commutateur OpenFlow. Il spécifie l'une des phases de négociation suivantes :
  1. **CONFIG_DISPATCHER** : Version négociée et envoi du message de demande de fonctionnalités.
  2. **MAIN_DISPATCHER** : C'est un message de fonctionnalités de commutateur reçu et envoyé, message de configuration.
  3. **DEAD_DISPATCHER** : La déconnexion en raison d'erreurs irrécupérables.
- **lookup_service_brick** : Le fonctionnement de la stratégie de routage proposée nécessite l'utilisation des données d'autres modules. Cela est possible en utilisant la classe app_manager.lookup_service_brick ('nom du module') pour obtenir une instance d'un module en cours d'exécution. Cette classe se trouve au niveau de ryu/base/app_manager.
- **packet** : Ce script nous permet d'analyser et de créer divers paquets de protocole de couche 2 et 3. Il se trouve au niveau de ryu/lib/packet.
- **Switches & LLDPPacket** : Le protocole LLDP (Link Discovery Protocol) permet la détection du délai au niveau des liaisons réseau et ce en effectuant les changements nécessaires au niveau de la classe LLDPPacket du fichier switches.py.

```
import logging
import struct
import copy
from ryu import cfg
from ryu.topology.switches import Switches
from ryu.topology.switches import LLDPPacket
import networkx as nx
from ryu.base import app_manager
from ryu.ofproto import ofproto_v1_3
from ryu.controller import ofp_event
from ryu.controller.handler import set_ev_cls
from ryu.controller.handler import MAIN_DISPATCHER, DEAD_DISPATCHER
from ryu.controller.handler import CONFIG_DISPATCHER
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ipv4
from ryu.lib.packet import arp
```

FIGURE 4.2 – Initialisation des packages

4.4.2 Le module Network_stat

Le module *Network_stat* est programmé afin de construire la vue globale du réseau, y compris les informations d'interconnexion des hôtes. Ceci est réaliser grâce à l'implémentation de la classe **NetworkStat** qui permet l'initialisation des différents structures utilisés et le déclenchement de la découverte réseau comme décrit dans le code de la Figure 4.3

```
class NetworkStat(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    def __init__(self, *args, **kwargs):
        super(NetworkStat, self).__init__(*args, **kwargs)
        self.topology_api_app = self
        self.name = "discover"
        self.link_to_port = {}       # (src_dpid,dst_dpid)->(src_port,dst_port)
        self.correspondence_table = {}       # {(sw,port) :[host_ip], [host_mac]}
        self.switch_port_list = {}  # dpid->port_num
        self.host_access_ports_list = {}  # dpid->port_num
        self.switches_access_ports_list = {} # dpid->port_num
        self.graph = nx.DiGraph()
        self.pre_graph = nx.DiGraph()
        self.pre_correspondence_table = {}
        self.pre_link_to_port = {}
        self.all_paths = None
        self.switches = {}
        # Lancement du thread pour la decouverte du reseau.
        self.discover_thread = hub.spawn(self.exploration)

    def exploration(self):
        self.build_topology(None)
```

FIGURE 4.3 – La classe du module Network_stat

Les structures des données utilisées et leurs rôles sont donnés dans la suite :

- **link_to_port** : Permet de stocker la relation de mappage entre les liens et les ports des différents commutateurs OpenFlow.
- **correspondence_table** : Permet de stocker les informations d'accès de l'hôte.
- **switch_port_list** : Permet de stocker la liste de tous les ports des commutateurs OpenFlow.
- **host_acces_ports_list** : Permet de stocker les ports d'accès sortants (interfaces connectées aux terminaux).
- **Switches_access_ports_list** : Permet de stocker les ports d'accès connectés aux différents commutateurs (interfaces connectées aux commutateurs).
- **graph** : Permet de stocker le graphe de topologie du réseau.
- **pre_graph** : Permet de stocker la dernière topologie du réseau (avant la nouvelle découverte). Cela est faisable grâce au package networkx qui contient les structures nécessaires pour construire un graphe.
- **pre_link_to_port & pre_correspondence_table** : Sont utilisées pour enregistrer les dernières informations acquises et pour comparer avec les informations actuellement acquises.

Les principales fonctions du module Network_Stat sont donnés dans la suite :

- **exploration()** : Est la fonction de boucle principale. Elle permet l'exécution des fonctions constituant le module.
- **build_topology()** : Le contrôleur obtient des informations de commutateur et de port,

des informations de liaison, des informations d'accès à l'hôte...etc . De plus, le contrôleur met à jour les informations sur les ressources réseau en détectant les événements asynchrones des modifications du réseau en temps réel tel que l'ajout, la modification ou la suppression d'un port et l'ajout ou la suppression d'un lien. Cette fonction fait appel à 5 autres fonctions à savoir :

1. **get_switch()** : Permet d'obtenir la liste de tous les switches constituant le réseau à travers l'envoi des messages Openflow ' EventSwitchRequest'.
2. **get_switch_port()** : Permet d'obtenir la liste des ports de chaque switch identifié.
3. **get_link()** : Permet d'obtenir la liste des liens utilisés pour interconnecter les commutateurs entre-eux à travers l'envoi des messages Openflow 'EventLinkRequest'.
4. **get_links_ports()** : Permet d'obtenir la liste des ports des commutateurs connectés entre eux.
5. **get_graph()** : Permet d'obtenir le graphe construit qui est sous forme d'une matrice d'adjacence (lien, port).

- **show_topology()** : Cette fonction affiche les informations réseau sur le terminal (la table des ports de liaison et la table hôte d'accès ...).

4.4.3 Le module Network_Collection

Le module *Network_Collection* est implémenté afin de collecter périodiquement les statistiques sur les flux pour superviser la quantité de flux circulant dans le réseau et les statistiques sur les ports afin de calculer la bande passante résiduelle des liaisons réseau. Ceci est réalisé grâce à la classe **NetworkCollection** qui permet d'initialiser les structures de données utilisées et le démarrage de la collecte des informations réseau comme mentionné dans le code de la Figure 4.4.

Les structures des données utilisées et leurs rôle sont donnés dans la suite :

- **datapaths** : Enregistre le chemin de données connecté au contrôleur.
- **port_statistics** : Enregistre les statistiques de port.
- **port_speed** : Enregistre les informations de vitesse du port.
- **flow_statistics** : Enregistre les statistiques de flux.
- **flow_speed** : Enregistre les informations du débit.
- **statistics** : Enregistre toutes les informations des statistiques.
- **port_features** : Enregistre les informations sur les caractéristiques du port.
- **free_bandwidth** : Enregistre les informations de bande passante des liaisons.
- **discover** : Récupère l'instance du module *Network_stat*.
- **bandwidth_paths** : Enregistre la bande passante des chemins.
- **best_paths** : stocke les k-meilleurs chemins en termes de bande passante.

Les principales fonctions du module Network_Collection sont donnés dans la suite :

- **Collection()** : Utilisée pour lancer la collecte des statistiques de manière périodique.

- **request_statistics()** : Utilisée pour envoyer des paquets Openflow *'OFPPortStatsRequest'* et *'OFPFlowStatsRequest'* permet la demande d'informations sur les statistiques des ports et du trafic réseau respectivement.
- **flow_statistics_reply_handler() & port_statistics_reply_handler()** : Les fonctions de traitement des messages de réponse aux demandes de statistiques de flux et de port.
- **save_bandwidth_graph()** : Enregistrer les données de la bande passante des liaisons dans l'objet graphique Networkx.
- **minimum_bandwidth_link()** : Permet d'obtenir la bande passante actuelle minimal du chemin.
- **bandwidth_path()** : Permet d'obtenir les k-meilleurs chemins en termes de bande passante.
- **Calculate_speed()** : Permet de calculer la vitesse de transfert des données.
- **get_free_bandwidth()** : Permet d'avoir la bande passante libre à chaque instant t.
- **show_statistics()** : Permet d'avoir les statistiques réseau en fonction du trafic (port source, port destination, vitesse de transfert...) et en fonction du port (nombre de paquets transmis,nombre de paquets reçu, la capacité du lien, l'état du port, l'état du lien ...).

```
class NetworkCollection(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    def __init__(self, *args, **kwargs):
        super(NetworkCollection, self).__init__(*args, **kwargs)
        self.name = 'Collection'
        self.datapaths = {}
        self.port_statistics = {} # Contient les statistiques de port
        self.port_speed = {}
        self.flow_statistics = {} # Contient les statistiques de flux
        self.flow_speed = {}
        self.statistics = {} # self.statistics['flow'] / self.statistics['port']
        self.port_features = {}
        self.free_bandwidth = {}
        self.discover = lookup_service_brick('discover')
        self.graph = None
        self.Bandwidth_paths=None
        self.best_paths = None
        # Lancement de thread pour la collection des statistiques et
        # le calcul de la bande passante residuelle
        self.Collection_thread = hub.spawn(self.Collection)
        self.save_freebandwidth_thread = hub.spawn(self.launch_bandwidth_graph)
```

FIGURE 4.4 – La classe NetworkCollection

4.4.4 Le module Network_D

Le module *Netwok_D* permet le calcul du délai des liaisons réseau de manière périodique à travers le calcul du temps écoulé des paquets LLDP et des paquets Echo afin de sélectionner le meilleur chemin ayant le plus petit délai. Ceci est implémenté grâce à la classe **NetworkD** qui permet l'initialisation des structures de données utilisées et le démarrage du calcul comme montré dans le code de la Figure 4.5.

```
class NetworkD(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    def __init__(self, *args, **kwargs):
        super(NetworkD, self).__init__(*args, **kwargs)
        self.name = 'delay_D'
        self.sending_echo_request_interval = 0.05
        # Obtention de l'objet actif des switches et du module network_stat
        # Pour qu'il puisse utiliser leurs donnees.
        self.sw_module = lookup_service_brick('switches')
        self.discover = lookup_service_brick('discover')
        self.delay_paths = None
        self.best_path = None
        self.pos=None
        self.delay_min=None
        self.datapaths = {}
        self.latency_controller_switch = {} # stocker le delai controleur-commutateur
        self.measure_delay_thread = hub.spawn(self.Trigger) # lancement de calcul du delai
```

FIGURE 4.5 – La classe NetworkD

Les structures des données utilisées et leurs rôles sont donnés dans la suite :

- **sending_echo_request_interval** : Spécifie l'intervalle d'envoi des paquets *echo_request*.
- **sw_module** : Récupère l'instance du fichier switches.
- **delay_paths** : Enregistre les délais des chemins.
- **best_pah** : Enregistre le meilleur chemin ayant le plus petit délai.
- **pos** : Enregistre la/les position(s) du/des chemin(s) ayant le plus petit délai.
- **delay_min** : Stocke la valeur minimale du délai.
- **Latency_controller_switch** : Sauvegarde le délai entre le commutateur et le contrôleur.

Les principales fonctions du module *Network_D* sont donnés dans la suite :

- **Trigger()** : Permet le lancement périodique des fonctions qui contribuent au calcul du délai.
- **send_echo_request()** : Permet l'envoi des messages Openflow *'OFPEchoRequest'* sur le chemin de données.
- **receive_echo_reply()** : Permet la gestion des messages Openflow *'OFPEchoReply'* pour obtenir la latence du lien contrôleur-commutateur.
- **Save_lldp_delay()** : Analyse les paquets LLDP et obtient le temps écoulé de ces derniers.
- **get_delay()** : Permet le calcul du délai d'une liaison réseau.
- **total_delay_path()** : Permet de calculer le délai de bout en bout du chemin.
- **delay_best_path()** : Permet de choisir le meilleur chemin ayant le plus petit délai.

4.4.5 Le module Sh_Forward

Le module *Sh_Forward* est le principal module qui permet le lancement des 3 modules précédents pour la sélection du meilleur chemin et l'installation des entrées de flux au niveau des tables des commutateurs concernés afin d'assurer le bon acheminement des données de bout en bout. Ceci est réalisé à l'aide de la création de la classe **ShForward** qui permet le lancement des modules et l'initialisation des structures de données utilisées comme mentionné dans le code de la Figure 4.6.

```
class ShForward(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    _CONTEXTS = {
        "network_Stat": network_Stat.NetworkStat,
        "network_Collection": network_Collection.NetworkCollection,
        "network_D": network_D.NetworkD}
    WEIGHT_MODEL = {'hop': 'weight', 'delay': "delay", "bw": "bw", 'bw_delay': "bw_delay"}
    def __init__(self, *args, **kwargs):
        super(ShForward, self).__init__(*args, **kwargs)
        self.name = 'Forwardsh'
        self.discover = kwargs["network_Stat"]
        self.Collection = kwargs["network_Collection"]
        self.delay_D = kwargs["network_D"]
        self.datapaths = {}
        self.weight = self.WEIGHT_MODEL[CONF.weight]
        self.best_paths = None
```

FIGURE 4.6 – La classe ShForward

Les structures des données utilisées et leurs rôles sont donnés dans la suite :

- **OFP_VERSIONS** : Spécifie la version d'OpenFlow, dans notre cas, la version 1.3.
- **_CONTEXTS** : Est l'attribut de la classe RyuApp. Le contenu de *_CONTEXTS* sera chargé en tant que service du module *Sh_Forward* lors de l'initialisation du module.
- **WEIGHT_MODEL** : Contient les métriques utilisées lors du routage.
- **discover& Collection & delay_D** : Lors de l'initialisation du module *Sh_Forword*, les instances des modules *network_stat*, *network_Collection* et *network_D* sont récupérées au niveau des structures *discover*, *Collection* et *delay_D* respectivement afin d'obtenir les informations fournies par chacun de ces modules.

Les principales fonctions du module Sh_Forward sont donnés dans la suite :

- **Forwardsh()** : Permet le déclenchement de la stratégie d'acheminement de flux.
- **get_switch_id()** : Obtention des identificateurs des commutateurs source et destination.
- **get_best_path()** : Obtient le meilleur chemin en fonction de la bande passante, le délai et le nombre de saut comme expliqué dans le chapitre 3.
- **install_flow()** : Création des paquets openflow *'OFPFlowMod'* et de les envoyer à tous les commutateurs constituant le meilleur chemin sélectionné afin d'installer les entrées de

flux au niveau des tables de flux pour assurer l'acheminement des données.

4.5 Simulations

Nous présentons la dépendance de chaque module à travers la Figure 4.7.

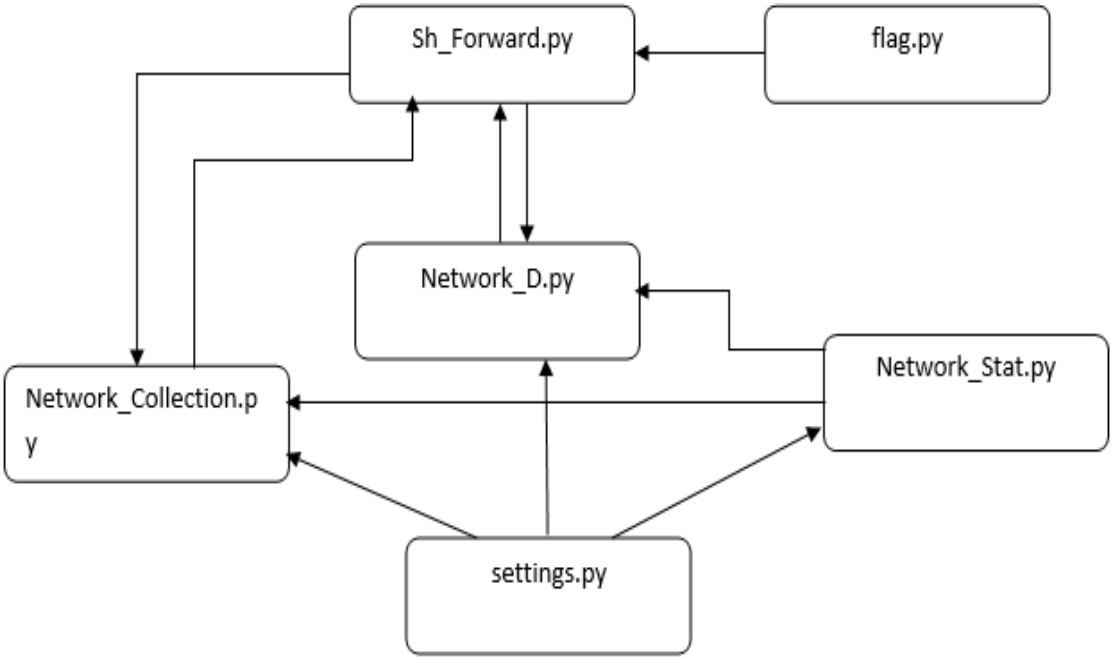


FIGURE 4.7 – La dépendance entre les différents modules de la stratégie de routage

Où :

-*Settings.py* : Est un fichier python contenant les paramètres de simulation telles que la période d'acquisition des informations de liaison, l'affichage des statistiques collectées et calculées, la table de la vue globale...etc.
-*flags.py* : Un fichier python qui appartient au répertoire du contrôleur ryu, dans lequel on spécifie les paramètres utilisés lors du lancement de notre stratégie de routage. Dans notre cas on rajoute les lignes suivantes :

```
CONF.register_cli_opts([

    cfg.IntOpt('k-paths', default=1, help='les k chemins'),
    cfg.StrOpt('weight', default='hop', help='choix de la metrique.')])
```

Notre application se situe au niveau du répertoire ryu/app, pour l'utiliser nous exécutons la commande suivante au niveau du répertoire racine de ryu.

```
$ sudo python setup.py install
```

4.5.1 Mise en place de la stratégie de routage

Une fois la réinstallation terminée, nous lançons l'application que nous avons construite au niveau du *Sh_Forward*. Comme nous l'avons déjà mentionné plus haut, Sh_Forward s'occupe du lancement de toutes les autres applications.
Nous exécutons *Sh_Forward* au niveau du contrôleur ryu en spécifiant le répertoire de l'application, le nombre de chemins que nous voulons prendre en compte et le paramètre choisi. Pour ce faire, nous spécifions le paramètre (bw + delay) comme le montre la Figure 4.8.

```
sdn@ubuntu:~/ryu$ PYTHONPATH=. ./bin/ryu-manager ryu/app/App-PFE/sh_Forward.py
--observe-links --k-paths=2 --weight=bw_delay
loading app ryu/app/App-PFE/sh_Forward.py
Generating grammar tables from /usr/lib/python2.7/lib2to3/Grammar.txt
Generating grammar tables from /usr/lib/python2.7/lib2to3/PatternGrammar.txt
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app None of NetworkCollection
creating context network_Collection
instantiating app None of NetworkD
creating context network_D
instantiating app None of NetworkStat
creating context network_Stat
instantiating app ryu.topology.switches of Switches
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ryu/app/App-PFE/sh_Forward.py of ShForward
Create bw graph exception
```

FIGURE 4.8 – Lancement de la stratégie de routage au niveau du contrôleur Ryu

Pour évaluer les performances de notre stratégie de routage, nous avons opté pour une topologie bien connue qui est "Abilene" [15]. Abilene est un réseau de haute performance qui permet le développement d'applications Internet avancées et le déploiement des services réseau. Cette topologie est constituée de onze commutateurs et onze hôtes connectés entre eux par des liaisons bidirectionnelles comme le montre la Figure 4.9.

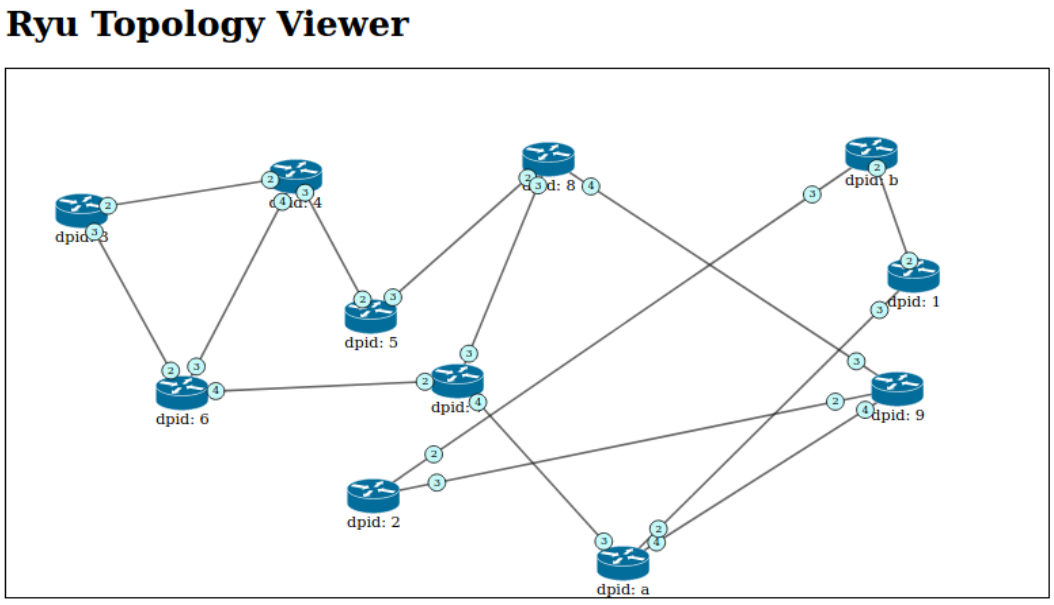


FIGURE 4.9 – L'affichage de la topologie optée pour le test au niveau du contrôleur Ryu

4.5.2 Test de connectivité

Après avoir lancer la stratégie de routage au niveau du contrôleur ryu ainsi que la topologie sur mininet. Nous testons la connectivité réseau en utilisant des paquets ICMP *(Internet Control Message Protocol)*. Dès que les paquets sont transmis, le contrôleur obtient les informations de flux et calcule le chemin optimal comme illustrée ci-dessous :

```
sdn@ubuntu:~/ryu$ PYTHONPATH=. ./bin/ryu-manager  ryu/app/App-PFE/sh_Forward.py
--observe-links --k-paths=4 --weight=bw_delay
loading app ryu/app/App-PFE/sh_Forward.py
Generating grammar tables from /usr/lib/python2.7/lib2to3/Grammar.txt
Generating grammar tables from /usr/lib/python2.7/lib2to3/PatternGrammar.txt
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app None of NetworkCollection
creating context network_Collection
instantiating app None of NetworkD
creating context network_D
instantiating app None of NetworkStat
creating context network_Stat
instantiating app ryu.topology.switches of Switches
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ryu/app/App-PFE/sh_Forward.py of ShForward
Create bw graph exception
switch:9 connected
switch:6 connected
switch:10 connected
switch:5 connected
switch:3 connected
switch:7 connected
switch:1 connected
switch:4 connected
switch:11 connected
switch:2 connected
switch:8 connected
location 10.0.0.8 is not found.
*******Demarage du strategie de routage **********
All possible paths:
[[11, 1, 10, 7], [11, 2, 9, 8, 7], [11, 2, 9, 10, 7], [11, 1, 10, 9, 8, 7], [11,
 2, 9, 8, 5, 4, 6, 7], [11, 1, 10, 9, 8, 5, 4, 6, 7], [11, 2, 9, 8, 5, 4, 3, 6,
7], [11, 1, 10, 9, 8, 5, 4, 3, 6, 7]]
{0: 9999.999520718922, 1: 9999.999520718922, 2: 9999.999520718922, 3: 9999.99952
0718922, 4: 9999.999520718922, 5: 9999.999520718922, 6: 9999.999520718922, 7: 99
99.999520718922}
The k-best paths in terms of Bandwidth:
[[11, 1, 10, 7], [11, 2, 9, 8, 7], [11, 2, 9, 10, 7], [11, 1, 10, 9, 8, 7]]
The bandwidth of each k-best path:
{0: 9999.999520718922, 1: 9999.999520718922, 2: 9999.999520718922, 3: 9999.99952
0718922}
delay of each path:
{0: 0.00396573543548584, 1: 0.005242586135864258, 2: 0.005079507827758789, 3: 0.
007471680641174316}
the smallest delay value
{0: 0.00396573543548584}
the positions of the paths with the smallest delay:
{0: 0}
The Best Path between:   10.0.0.1 <------> 10.0.0.8 is : [11, 1, 10, 7]
```

FIGURE 4.10 – Résultat de l'exécution de la stratégie de routage proposée au sein du contrôleur ryu

Lors du test de connectivité entre une source et une destination, le premier paquet **Packet_in** est transmis au contrôleur. Le contrôleur exploite le protocole ARP (*Adress Résolution Protocol*) qui a un rôle phare parmi les protocoles de la couche internet de la suite TCP/IP, car il permet de connaître l'adresse physique et l'associe à une adresse IP.

La table de correspondance (Adresse Mac,Adresse IP) est consultée. Si l'adresse de destination n'est pas dans cette table, le protocole ARP émet une requête **ARP_Request**. L'ensemble des machines du réseau vont comparer cette adresse logique à la leur, si l'une d'entre-elles s'identifie à cette adresse la machine va répondre par un **ARP_reply** et le couple d'adresses est sauvegardé dans la table de correspondance dans la mémoire cache. La communication peut ainsi avoir lieu.

Le contrôleur prend la décision de routage en calculant le chemin le plus optimal en se basant sur les informations réseau de base, notamment la topologie, les statistiques réseau, le délai de transfert et la bande passante résiduelle.

Selon notre stratégie, nous déterminons les k chemins ayant la bande passante maximal (le paramètre k est déterminé par l'utilisateur). Parmi ces k chemins, le contrôleur sélectionne le meilleur chemin ayant le délai de transfert le plus minimal. Ainsi, les paquets sont acheminés de la source à la destination et les mises à jour au niveau des tables de flux sont effectuées.

4.6 Outils uilisés pour les mesures de performance

Afin de prouver l'efficacité de notre stratégie de routage proposée, nous comparons les différents résultats de performance avec la stratégie de routage basée uniquement sur la bande passante et ceci est réalisableà l'aide de l'outil standard **Iperf** [6] et **Bwm-ng** [5] (*Bandwidth Monitor NG*).

4.6.1 Iperf

Iperf est un logiciel open source écrit en C. Il s'exécute sur diverses plateformes, incluant Linux, Windows et Mac. C'est un outil largement utilisé pour la mesure et le réglage du rendement réseau. Il peut produire des mesures de performance pour n'importe quel réseau. Iperf dispose de fonctionnalités client/serveur et peut créer des flux de données pour mesurer le débit entre deux extrémités dans l'une ou les deux directions. La sortie **Iperf** standard contient un rapport horodaté de la quantité de données transférées et du débit mesuré.

Le flux de données peut être, soit le protocole TCP soit le protocole UDP .

- UDP : Lorsqu'il est utilisé pour tester la capacité, **Iperf** permet à l'utilisateur de spécifier la taille du datagramme et fournit des résultats pour le débit de datagrammes et la perte de paquets.

- TCP : Lorsqu'il est utilisé pour tester la capacité, **Iperf** mesure le débit de la charge utile.

4.6.2 Bwm-ng

Bwm-ng est un programme de surveillance de la bande passante du réseau et du disque. Il peut être exécuté à partir de la console sur les plates-formes Linux, BSD et Solaris. Il automatise le suivi et l'affichage de la bande passante totale et la bande passante de chaque interface de sortie. On dirige les résultats obtenus par cet outil vers un fichier texte afin de pouvoir l'analyser et déduire les performances obtenues.

4.7 Les paramètres de simulation

Le tableau 4.1 décrit les différents paramètres utilisés dans l'environnement de la simulation afin d'évaluer la stratégie proposée :

Paramètre	**Valeur**
Contrôleur	RYU
Simulateur	Mininet
Topologie	Abilène
Durée de la simulation	60s - 1,7 heures
Taille des paquets	84 Bytes
Type de trafic	TCP
Nombre de trafic	60, 600, 6000, 60000 Paquet
Nombre d'émetteur	20
Nombre de recepteur	20

TABLE 4.1 – Paramètres établis pour l'évaluation de la stratégie de routage proposée.

4.8 Mesure des performances et résultats

Nous avons procédé à plusieurs simulations afin de dégager l'intérêt des métriques utilisées (débit moyen, délai d'aller-retour et le taux de perte) par notre stratégie de routage pour la prise de décision. Nous avons réalisé une comparaison avec le routage basé sur la bande passante.

4.8.1 Le débit moyen

Le débit moyen représente la quantité du trafic envoyée durant un temps t. Le calcul de cette métrique est effectué par le programme de surveillance Bwm-ng.

Les résultat des simulations des deux stratégies nous ont permis de construire un graphe sous forme d'histogramme.

Le graphe ci-dessous représente le débit moyen en Mb/s de la stratégie de routage basée uniquement sur la bande passante et de la stratégie de routage basée sur la bande passante, le délai et le nombre de saut en fonction de nombre de paquets envoyées dans le réseau.

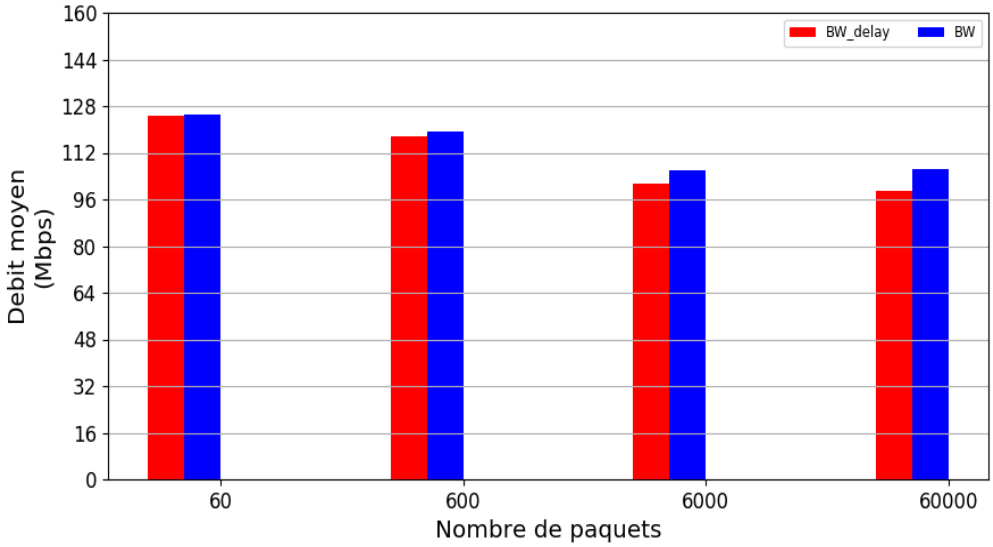


FIGURE 4.11 – débit moyen

L'augmentation du nombre de paquet transmit dans le réseau permet la diminution du débit moyen pour les 2 stratégies simulées. Par conséquent, le débit moyen de la stratégie du routage basée uniquement sur la bande passante est presque identique au débit moyen de notre stratégie de routage. Ceci est dû au fait que le chemin sélectionné par la première stratégie ayant toujours la bande passante la plus élevés par contre notre stratégie sélectionne un chemin parmi les k-meilleurs chemins en terme de bande passante (ce chemin peut-être un chemin ayant la bande passante la plus élevés comme il peut être un chemin parmi les (k-1) meilleurs chemins).

En comparant les deux stratégies, la stratégie basée uniquement sur la bande passante est moyennement meilleure que notre stratégie en terme de débit moyen.

4.8.2 Le délai d'aller-retour des paquets de trafic

Le temps d'aller-retour RTT (*Round-trip time*) est la durée en millisecondes (ms) nécessaire pour qu'un paquet soit émis à une destination et retourne une réponse à l'émetteur. RTT est une métrique importante pour déterminer l'état du réseau.

Le RTT est généralement mesuré à l'aide d'un ping qui est un outil de ligne de commande qui renvoie une requête sur un serveur et calcule le temps nécessaire pour atteindre une machine utilisateur.

Le graphe de la Figure 4.12 représente le délai moyen d'aller-retour des paquets du trafic en milliseconde de la stratégie de routage basé uniquement sur la bande passante et de la stratégie de routage basée sur la combinaison de la bande passante, le délai et le nombre de saut en fonction de nombre de paquets envoyées dans le réseau.

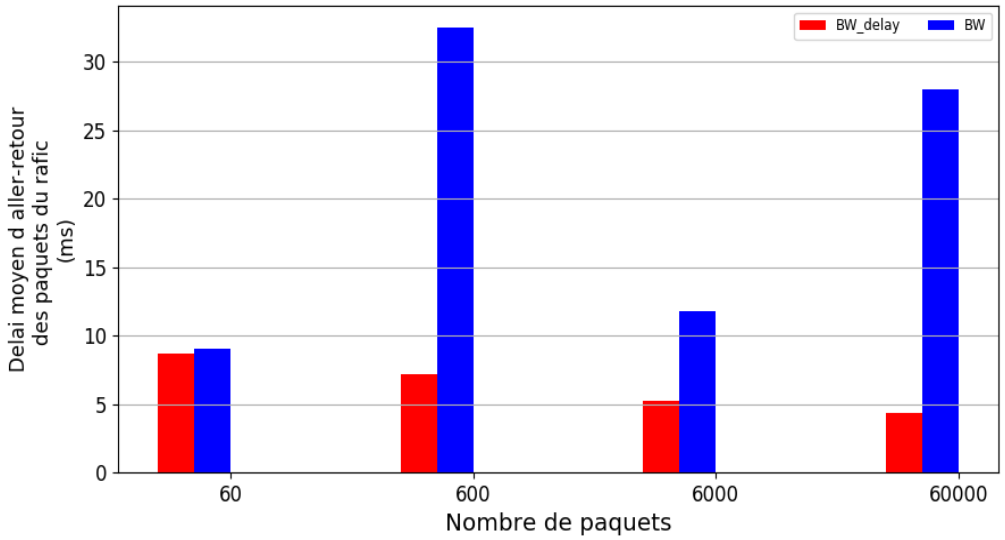


FIGURE 4.12 – Délai moyen d'aller-retour des paquets du trafic

La Figure 4.12 révèle la force de la stratégie de routage adoptée. Le temps de parcours nécessaire pour qu'une requête réseau passe d'un point de départ à une destination et retourne au point de départ de notre stratégie de routage est nettement meilleur que celui de la stratégie de routage utilisant la bande passante comme seule critère de choix du chemin.

En effet, la combinaison de plusieurs métriques (bande passante, délai et nombre de sauts) permet d'avoir le compromis entre vitesse de transfert, délai minimal des liaisons constituant le chemin et le plus court chemin permettant ainsi l'envoi des paquets et la réception des acquittements en un temps record. Ceci est dû à une distribution du trafic favorisée par l'application du critère 'retard de lien'. En effet, l'application du seul critère BP va pousser le trafic à traverser les même liens de manière permanente, ce qui crée une surcharge sur ces lien, des retards et des pertes de paquets.

4.8.3 Le taux de perte de paquets

Le taux de perte de paquets est le pourcentage de paquets perdus lors de la transmission de données. Il s'agit d'un critère de qualité de services d'un réseau. Nous nous sommes basées sur l'équation (4.1) pour le calculer :

$$Taux = \frac{\sum_{i=0}^{T}(NBP_{Lost})}{|NBP_{Lost}|}; \tag{4.1}$$

Où :

NBP_{Lost} : une liste contenant l'ensemble des paquets perdu qui est égale à la soustraction du nombre de paquets envoyés du nombre de paquets reçu pour chaque récepteur.
$|NBP_{Lost}|$: la taille de la liste qui est égale au nombre de récepteurs.

Le graphe ci-dessous représente le taux de perte des paquets du trafic de la stratégie de routage basé uniquement sur la bande passante et de la stratégie de routage basée sur la combinaison de la bande passante, le délai et le nombre de saut en fonction de nombre de paquets envoyées dans le réseau.

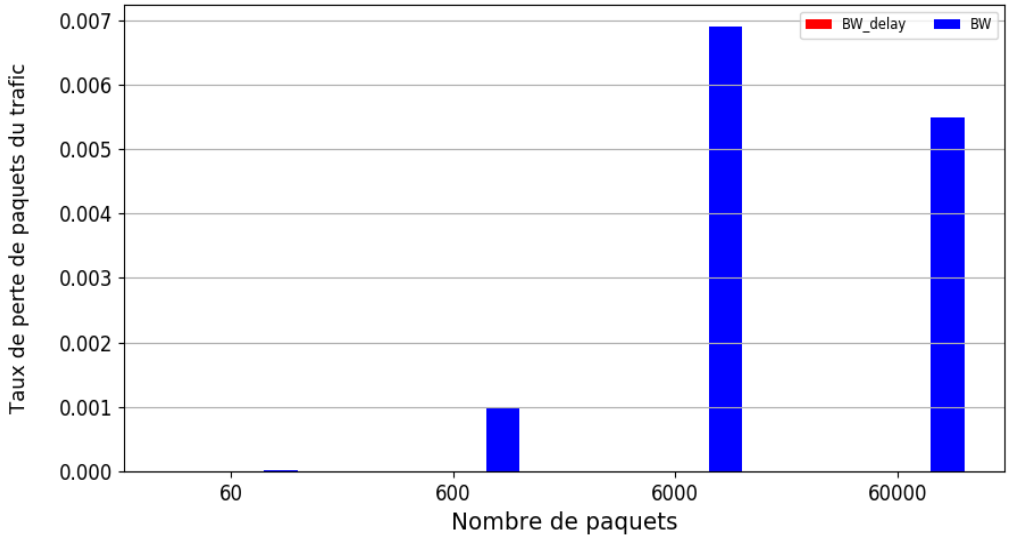


FIGURE 4.13 – Le taux de perte de paquets du trafic

Le taux de perte est en corrélation directe avec le délai d'aller-retour des paquets. En effet, l'augmentation du délai d'aller-retour affecte la latence du réseau et produira une file d'attente au niveau des noeuds dans laquelle l'information passera plus lentement, en finissant à être écartée après une certaine période de temps (*packet drop*).

Le graphe de la Figure 4.13 affirme cela, la stratégie de routage proposée prend en charge le délai des liaisons en privilégiant les plus petits et donc une délivrance avec succès de tous les ACKs.

A partir du graphe illustrée dans la Figure 4.13, notre stratégie de routage est meilleure que la stratégie basée uniquement sur la bande passante en terme de taux de perte des paquets du trafic.

4.9 Conclusion

Nous avons montré, tout au long de ce chapitre, la manière avec laquelle nous avons procédé dans la mise en place de l'environnement et l'intégration de notre stratégie de routage qui permet de prendre en considération la combinaison entre la bande passante, le délai des liaisons et le nombre de sauts.

Nous avons énoncé ainsi la manière avec laquelle les tests de performances et les comparaisons ont été menées. Ces simulations ont permis de mettre en avant la valeur ajoutée que procure la stratégie proposée, en la comparant avec un routage qui ne permet de prendre en considération que la bande résiduelle des liaisons. L'utilisation de notre stratégie permet de diminuer considérablement le taux de perte et le délai d'aller-retour des paquets grâce à la combinaison de la bande passante, le délai et le nombre de sauts dans cette ordre.

Conclusion Générale

Le SDN, comme le désigne son nom, est un réseau défini par logiciel. C'est un modèle d'architecture qui consiste à rendre les réseaux agiles, flexibles tout en permettant le déploiement de nouvelles applications, de nouveaux services et une nouvelle infrastructure pour répondre rapidement aux besoins en constante évolution. Le SDN a apporté des solutions pour différents problèmes que la technologie classique ne prend pas en compte.

L'approche SDN présente de multiples avantages. Il a été prouvé qu'une telle approche centralisée fournit un temps de configuration des chemins beaucoup plus court que l'utilisation de protocoles de routage distribués existants. De plus, en limitant le nombre de dispositifs du plan de contrôle et en simplifiant l'architecture des dispositifs du plan de données, la complexité et le coût du réseau peuvent être considérablement réduits. En raison de la vue globale du réseau et de la programmabilité, le SDN est un meilleur outil pour résoudre les problèmes de gestion du trafic qui existent dans les réseaux IP classiques.

Les propriétés suivantes du SDN montrent qu'une gestion du trafic peut être mise en œuvre :

- L'ensemble de la topologie du réseau est connu par le contrôleur SDN. Par conséquent, tous les chemins existants et leurs charges réelles peuvent être facilement identifiés et plusieurs routes peuvent être utilisées pour le transfert du trafic entre n'importe quelle paire source-destination.

- Le contrôleur, à l'aide d'OpenFlow, permet des mises à jour au niveau des tables de flux afin d'acheminer le trafic du nœud source au nœud de destination.

Après étude approfondie des besoins, nous avons conclu qu'un travail de conception globale est nécessaire afin de concevoir une stratégie de routage et de gestion du trafic de tout le réseau. Nous avons donc proposé d'intégrer au niveau du contrôleur SDN, un module de surveillance : collecte des statistiques sur les ports des équipements et collecte des statistiques sur les flux de données échangées pour assurer une surveillance continue du réseau. Nous avons également intégré une stratégie d'acheminement du trafic réseau au niveau du contrôleur, qui utilise les informations collectées par le module de surveillance pour calculer la bande passante, le délai de bout en bout, le plus court chemin et élire le chemin optimal.

Dans le cadre de ce projet, afin de montrer l'impact de l'intégration proposée, plusieurs simulations ont été réalisées afin de dégager l'intérêt des paramètres utilisés par notre stratégie de routage, à savoir la bande passante, le retard des liens et le plus court chemin pour la prise de décision. Ainsi, nous avons simulé d'abord en négligeant le retard des liens ensuite, nous l'avons intégré dans les simulations suivantes. Cette étude avait pour but de comparer les performances de communications de notre stratégie de routage proposée avec la stratégie qui prend la bande passante comme seule critère de choix du chemin optimal.

Nous avons pu répondre à la problématique de gestion du trafic dans les réseaux SDNs, en proposant une stratégie de routage et d'acheminement des flux réseaux qui prend en considération la quantité du trafic en transit et les ressources réseau disponibles.
Les simulations menées suite à cela ont permis de souligner les avantages en termes de délai d'aller-retour des paquets de données et de résistance aux pertes que présente la

stratégie adoptée, comparée à la stratégie de routage prenant la bande passante comme seul critère du choix de chemin. Nous avons montré également que la stratégie de routage proposée peut être appliquée en coexistence avec la pile de protocoles de réseau TCP / IP conventionnelle.

Comme perspectives, nous proposons d'implémenter un protocole de routage qui prend en charge d'autres paramètres, par exemple le paramètre énergie des commutateurs dans le sens où il soit possible aux opérateurs de réduire leurs factures d'électricité.
Un autre aspect peut améliorer le service de routage fournit par notre protocole, sera la prise en charge des pannes. En réalité, le protocole fournissant un calcul de chemin dynamique, prend automatiquement en considération les changements de la topologie. Cependant, la question de la panne mérite une réflexion de sorte à assurer une réaction immédiate du système.

En somme, ce projet nous a permis de toucher à plusieurs aspects du monde de l'informatique à savoir les réseaux, SDN, Virtualisation, programmation, ... Il nous a présenté un très bon moyen pour la mise en pratique des connaissances acquises durant ces dernières années de formations et c'est avéré être une bonne opportunité pour enrichir nos connaissances.

Annexes

ANNEXE 1 : Complément sur le SDN

1 Contrôleur

SDN a une large gamme de contrôleurs. Le Tableau suivant liste les différents contrôleurs ainsi que leurs caractéristiques.

contrôleur	language de programmation	GUI	Documentation	Modularité	Centralisé/Distribué	Platforme	Productivité	Southbound APIs	Northbound APIs
RYU	Python	oui	très bien	moyenne	C	Linux	élevé	OF1.0 1.2 1.3 1.4 NETCONF	REST API
ONOS	JAVA	basée web	bien	élevé	D	Linux Mac Os windows	acceptable	OF1.0 1.3 NETCONF	REST API
OpenDaylight	java	basée web	très bien	élevé	D	Linux Mac Os Windows	acceptable	OF1.0 1.3 1.4 NETCONF YANG	REST API
POX	python	python +QT4	mauvaise	faible	C	Linux mac Os Windows	élevé	OF1.0	REST API
NOX	C++	Python +QT4	mauvaise	faible	C	Linux	mauvaise	OF1.0	REST API
Flood-light	JAVA	baséé web	bien	Moyenne	C	Linux Mac Os Windows	faible	OF1.0 1.3	REST API

2 Interface Est/ouest

Les API est/ouest sont un cas particulier d'interfaces requises par les contrôleurs distribués. Actuellement, chaque contrôleur implémente sa propre API en direction est/ouest. Les fonctions 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).

Une API en direction est/ouest nécessite des mécanismes de distribution de données avancés, des techniques pour une composition distribuée concurrente et cohérente des politiques, des bases de données transactionnelles ou des algorithmes avancés pour une forte cohérence et tolérance aux pannes.

Dans une configuration multidomaine, les API orientées est/ouest peuvent également nécessiter des protocoles de communication plus spécifiques entre les contrôleurs de domaine SDN [52]. Certaines des fonctions essentielles de ces protocoles sont de coordonner la configuration du flux provenant des applications, d'échanger des informations d'accessibilité pour faciliter le routage inter-SDN, la mise à jour de l'accessibilité pour garder l'état du réseau cohérent, entre autres.

Cependant, le problème important concernant les interfaces est/ouest est l'hétérogénéité. Par exemple, en plus de communiquer avec des contrôleurs SDN homologues, les contrôleurs peuvent également avoir besoin de communiquer avec des contrôleurs non SDN. Pour être interopérables, les interfaces est / ouest doivent donc s'adapter à différentes interfaces de contrôleur, avec leur ensemble spécifique de services, et les diverses caractéristiques de l'infrastructure sous-jacente, y compris la diversité de la technologie, l'étendue géographique et l'échelle du réseau, et la distinction entre le WAN et le LAN potentiellement au-delà des frontières administratives. Dans ces cas, différentes informations doivent être échangées entre les contrôleurs, y compris la contiguïté et la découverte de capacités, les informations de topologie entre autres [42].

Enfin, une méthodologie [34] suggère une distinction plus fine entre les interfaces horizontales en direction est et en direction ouest, faisant référence aux interfaces en direction ouest en tant que protocoles SDN à SDN et API de contrôleur, tandis que les interfaces en direction est seraient utilisées pour faire référence aux protocoles standard utilisés afin de communiquer avec les plans de contrôle de réseau hérités.

3 Les tables manquantes

Chaque table de flux doit prendre en charge une entrée de flux de table manquante pour traiter les échecs de table. Cette entrée spécifie comment traiter les paquets qui ne correspondent à aucune entrée dans la table de flux, par exemple, envoyer des paquets au contrôleur, supprimer des paquets ou diriger des paquets vers une table suivante.

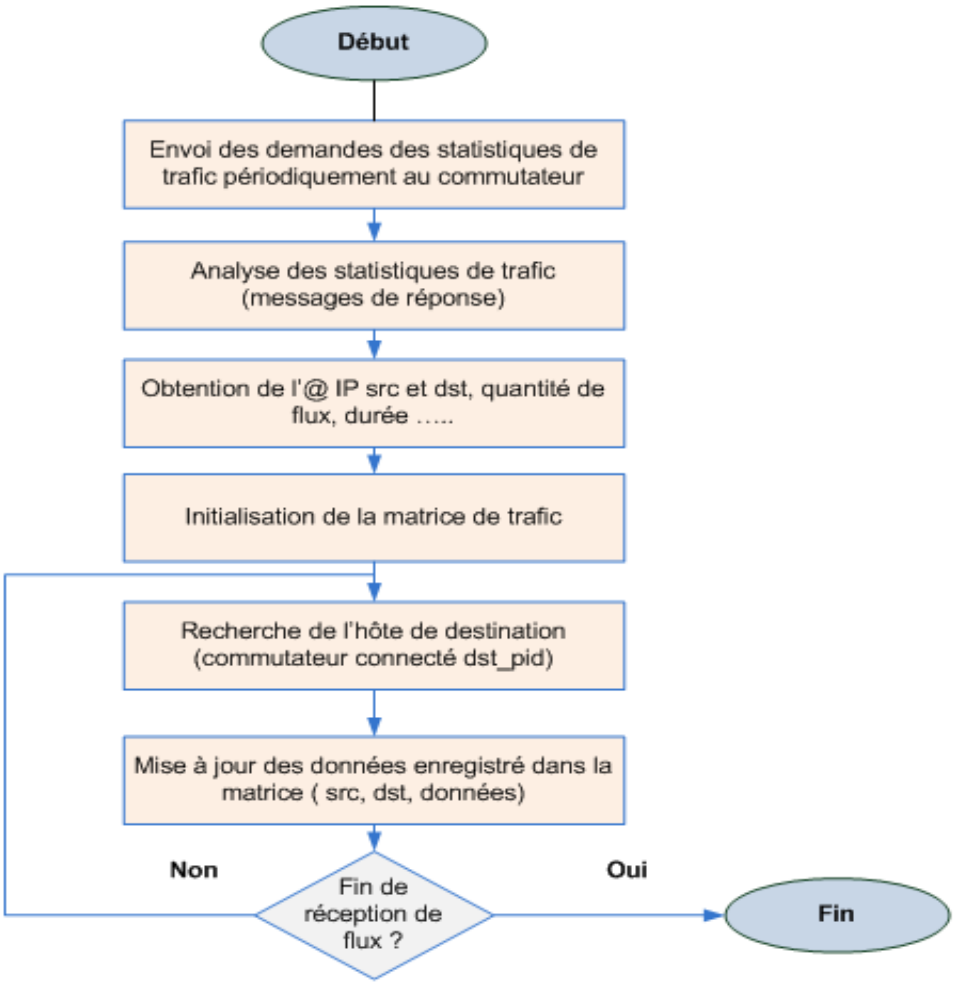
L'entrée de flux de table manquante est identifiée par sa correspondance et sa priorité, elle joker tous les champs de correspondance (tous les champs omis) et a la priorité la plus faible (0). La correspondance de l'entrée de flux de table manquante peut tomber en dehors de la plage normale de correspondances prises en charge par une table de flux, par exemple, une table de correspondance exacte ne prendrait pas en charge les caractères génériques pour d'autres entrées de flux, mais doit prendre en charge l'entrée de flux de table manquante avec des caractères génériques pour tous les champs. L'entrée de flux de cette table peut ne pas avoir la même capacité que l'entrée de flux normal.

L'entrée de flux de table manquante se comporte de la plupart des manières comme toute autre entrée de flux : elle n'existe pas par défaut dans une table de flux, le contrôleur peut l'ajouter ou la supprimer à tout moment et elle peut expirer. Cette entrée correspond aux paquets de la table comme prévu à partir de son ensemble de champs de correspondance et de priorité. Les instructions d'entrée de flux de table manquante sont appliquées directement aux paquets correspondant.

Si l'entrée de flux de la table manquante n'existe pas, par défaut, les paquets non appariés par les entrées de flux sont supprimés. Une configuration de commutateur, par exemple à l'aide du protocole de configuration OpenFlow, peut remplacer cette valeur par défaut et spécifier un autre comportement.

ANNEXE 2 : Organigrammes des processus de mise en place de la stratégie de routage

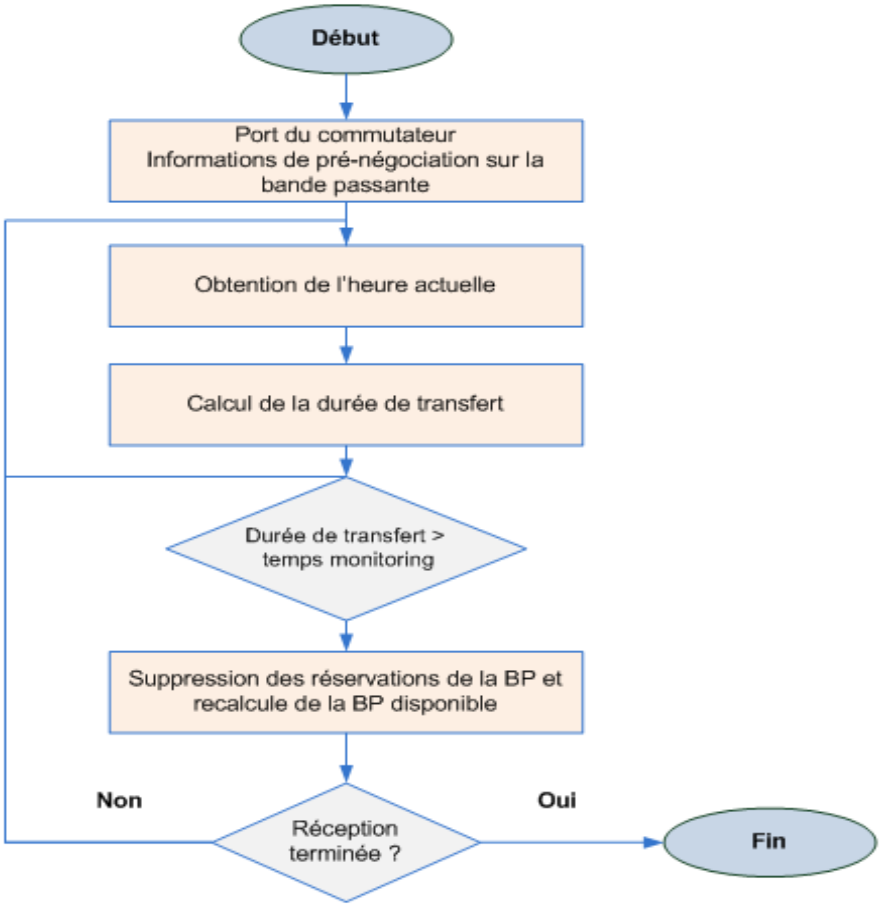
1 Le processus de collecte des statistiques sur les flux



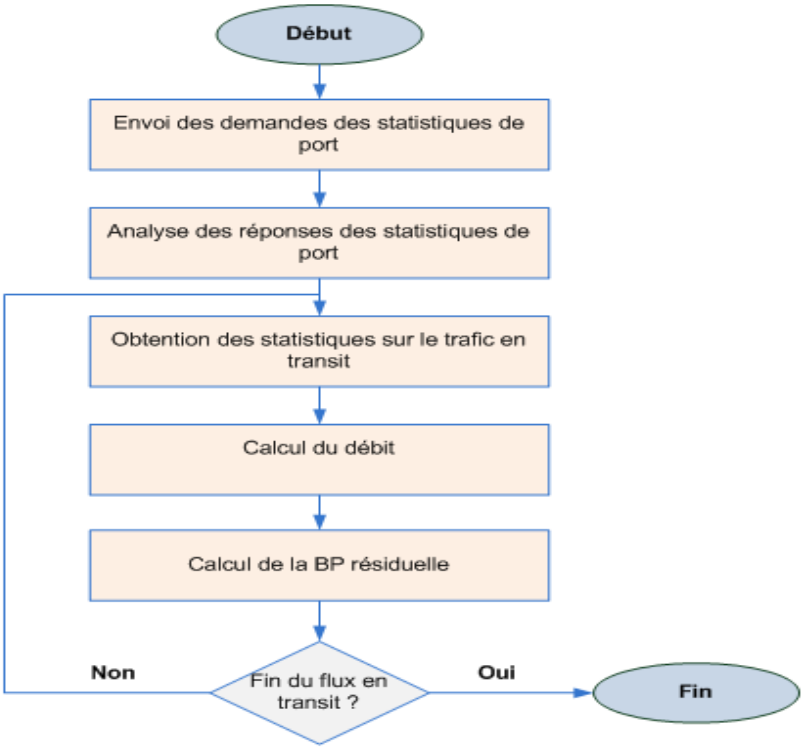
2 Processus de mise à jour de la bande passante

Afin de garantir la validité des données enregistrées au niveau de la structure de données "graphe" concernant la bande passante en utilisation et la bande passante résiduelle ;

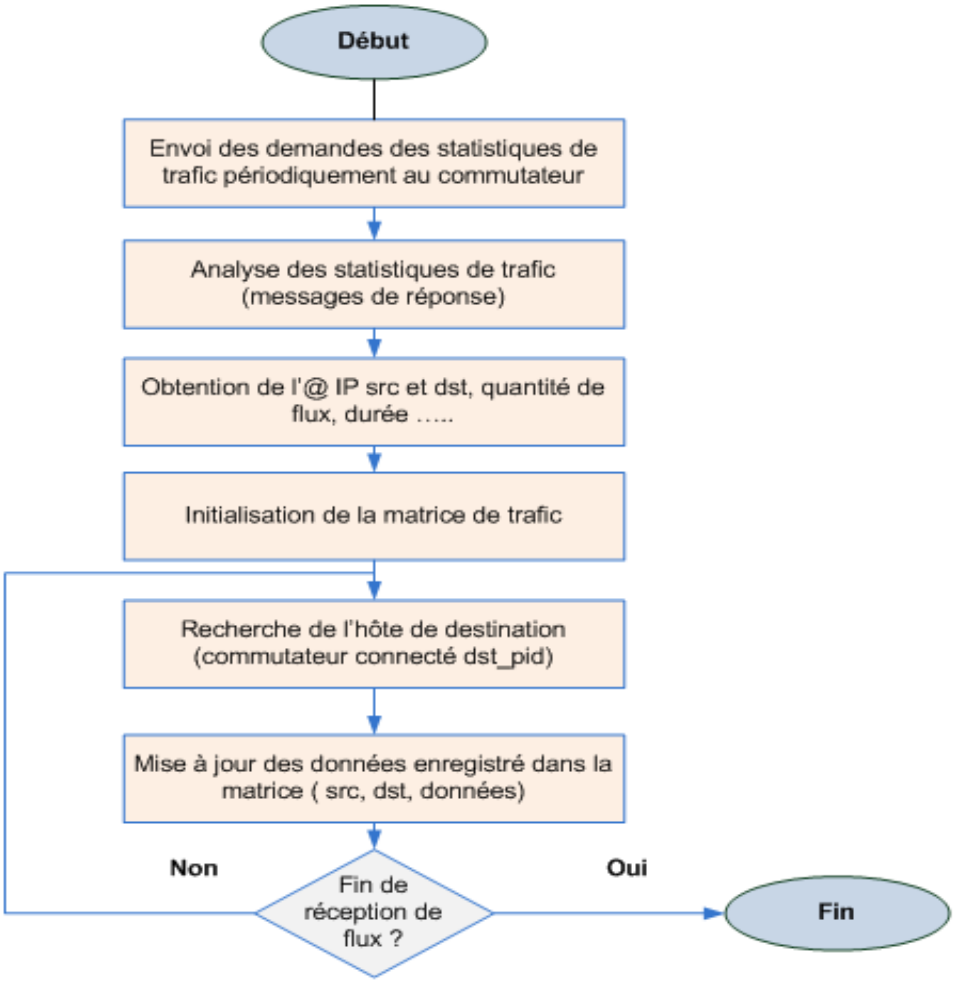
le contrôleur doit effectuer des mises à jour périodiques. L'organigramme de la Figure illustre cette partie.



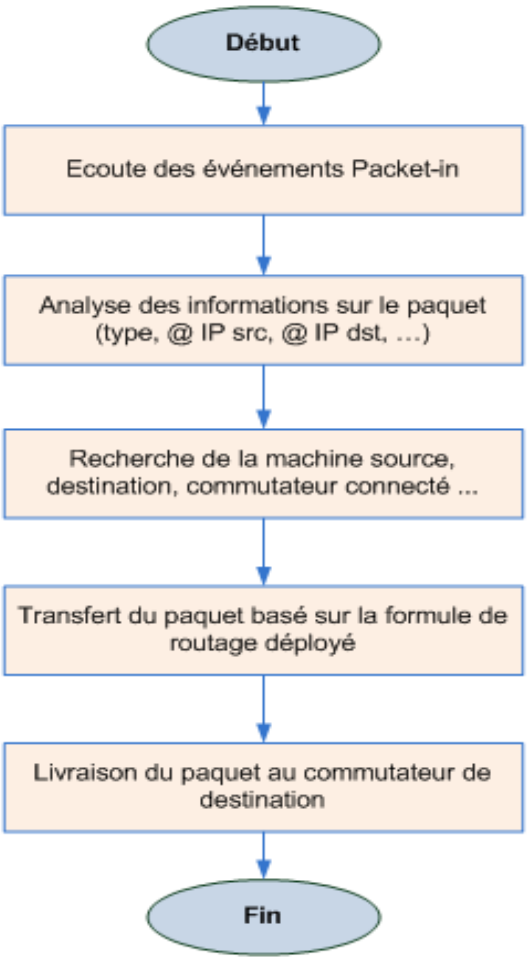
3 Acquisition des statistiques des ports



4 La collecte des statistiques sur le flux



5 La gestion des paquets entrants



Bibliographie

[1] Algorithmique avancée pour les graphes. https://perso.liris.cnrs.fr/christine.solnon/supportAlgoGraphes.pdf/.disponible 30/08/2020.

[2] Concepts cisco sdn. <https://cisco.goffinet.org/ccna/automation-programmabilite-reseau/concepts-sdn-cisco>. disponible 18-08-2020.

[3] Data ericsson sur l'iot, smartphone et usages. <https://www.servicesmobiles.fr/data-ericsson-sur-liot-smartphone-et-usages-32219>. disponible 18-08-2020.

[4] Enjeux des avancées technologiques : Sdn, 5g, identité numérique. <https://www.artci.ci/images/stories/pdf/publication/bulletin_veille_technologique_avril2016.pdf>. disponible 18-08-2020.

[5] l'outil bwm-ng. https://linux.die.net/man/1/bwm-ng.disponible 15/08/2020.

[6] L'outil iperf. https://iperf.fr/.disponible 04/08/2020.

[7] Mininet. http://mininet.org.> disponible, 19/03/2020.

[8] Netflow. <http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6555/ps6601/prod_white_paper0900aecd80406232.html>. disponible 18-08-2020.

[9] . Openflow switch specication. <http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf>. disponible 19-02-2020.

[10] . Openflow switch specification v1.0–v1.4. <https://www.opennetworking.org/sdn-resources/onf-specifications>. disponible 18-08-2020.

[11] Package networkx. https://networkx.github.io/>. disponible 18-08-2020.

[12] Production quality, multilayer open virtual switch. <https://www.openvswitch.org/> disponible 09/04/2020.

[13] Rfc 7419. <https://tools.ietf.org/html/rfc7419>. disponible 18-08-2020.

[14] Rfc 7426. <https://tools.ietf.org/html/rfc7426>. disponible 18-08-2020.

[15] Réseau abilène. https://www.internet2.edu/news/detail/1978/.disponible 8/08/2020.

[16] sflow. <http://www.sflow.org/sFlowOverview.pdf>. disponible 18-08-2020.

[17] Software-defined networking (sdn) definition,sur open networking foundation. <https://www.opennetworking.org/sdn-resources/sdn-definition>. disponible 18-08-2020.

[18] *Open Networking Foundation. OpenFlow Switch Specification [online].* 2016.

[19] Y. Ganjali . Hassas Yeganeh. Kandoo : a framework for efficient and scalable offloading of control applications, in : Proceedings of the first workshop on hot topics in software defined networks,. HotSDN'12, 2012.

[20] H. Rahimi A. Yassine and S. Shirmohammadi. "software defined network traffic measurement : Current trends and challenges,". IEEE Instrumentation Measurement Magazine, Vol.18 no.2, pp. 42-50, Mar, 2015.

[21] HOSSEINY Koosha Haji MIRJALILY Ghasem et al. ATTARHA, Shadi. A load balanced congestion aware routing mechanism for software defined networks. 2017 Iranian Conference on Electrical Engineering (ICEE). IEEE p. 2206-2210., 2017.

[22] MUSUMECI Francesco TORNATORE Massimo et al AYOUB, Omran. Efficient routing and bandwidth assignment for inter-data-center live virtual-machine migrations. IEEE/OSA Journal of Optical Communications and Networking, vol. 9, no 3, p. B12-B21., 2017.

[23] Ph.D. in Computer Science Bartosz Balis. Hyperflow : A model of computation, programming approach and enactment engine for complex distributed workflows, volume 55. Future Generation Computer Systems, 2016.

[24] System Management : Link Layer Discovery Protocol (LLDP) CCIE Routing Switching. `https://sunnynetwork.wordpress.com/2016/07/19/lab-xx/`. available 18-08-2020.

[25] BARI Md Faizul AHMED Reaz et al. CHOWDHURY, Shihabur Rahman. Payless : A low cost network monitoring framework for software defined networks. In : IEEE Network Operations and Management Symposium (NOMS). IEEE, p. 1-9., 2014.

[26] A.Elwalid I.Widjaja X. Xiao D.Awduche, A.Chiu. Overview and principles of internet traffic engineering. RFC 3272, Tech. Rep, May 2002.

[27] Muhammad Arief Nugroho. Fiqih Rhamdani, Novian Anggis Suwastika. Equal-cost multipath routing in data center network based on software defined network'. International Conference on Information and Communication Technology (ICoICT)., 2018.

[28] G. Ghoda, N. Meruliya, D. H. Parekh, T. Gajjar, D. Dave, and R. Sridaran. A survey on data center network virtualization. *International Conference on Computing for Sustainable Global Development (INDIACom)*, 2016.

[29] HONG Chi-Yao KANDULA Srikanth et al. GILL, Vijay. Swan : achieving high utilization in networks. U.S. Patent No 8,977,756, 10 mars, 2015.

[30] Khalid Bouragba Ihssane Choukri, Mohammed Ouzzif. Software defined networking (sdn) : Etat de l'art. colloque sur les objets et systèmes connectés,. Ecole Supérieure de Technologie de Casablanca (Maroc), Institut Universitaire de Technologie d'Aix-Marseille (France), June 2019.

[31] Fatma. KAMOUN, Faouzi et OUTAY. Ip/mpls networks with hardened pipes : service concepts, traffic engineering and design considerations. Journal of Ambient Intelligence and Humanized Computing, vol. 10, no 7, p. 2577-2584., 2019.

[32] ZOU Xuan ZHOU Wenxuan et al. KHURSHID, Ahmed. Veriflow : Verifying network-wide invariants in real time. In : Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13). p. 15-27., 2013.

[33] ZENG Yulong LI Jianfei et al LUO, Min. An adaptive multi-path computation framework for centrally controlled networks. Computer Networks,vol. 83, p. 30-44., 2015.

[34] T. Zinner M. Jarschel and al. . Interfaces, attributes, use cases : A compass for sdn. IEEE Communications Magazine. p. 210-217., 2014.

[35] AZIZ Benjamin AL-HAJ Ali et al. MALIK, Ali. Software-defined networks : A walkthrough guide from occurrence to data plane fault tolerance. PeerJ Preprints, 2019.

[36] BRAUN Wolfgang et MENTH Michael. MERLING, Daniel. Efficient data plane protection for sdn. in. 2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft). IEEE, p. 10-18., 2018.

[37] D. M. Dumitriu Midokura. *Using SouthBound APIs to build an SDN Solution*. 5 Feb 2014.

[38] SAAT Efi et MOSES Yoram. MIZRAHI, Tal. Timed consistent network updates in software-defined networks. IEEE/ACM Transactions on Networking, vol. 24, no 6, p. 3412-3425., 2016.

[39] TRUONG-HUU Tram et GURUSAMY Mohan. MOHAN, Purnima Murali. Primary-backup controller mapping for byzantine fault tolerance in software defined networks. In : GLOBECOM IEEE Global Communications Conference. IEEE, p 1-7, 2017.

[40] Rahamatullah. NDE, Gilbert N. et KHONDOKER. Sdn testing and debugging tools : A survey in :. 5th International Conference on Informatics, Electronics and Vision (ICIEV). IEEE. p. 631-635., 2016.

[41] A. ODINI, Marie-Paule et MANZALINI. Sdn in nfv architectural framework. *IEEE Software Defined Networks Newsletter,*, 2016.

[42] Open Networking Foundation (ONF). Sdn architecture. `<https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf>`. disponible 18-08-2020.

[43] I. Owens and A. Durresi. *"Video over Software-Defined Networking (VSDN)"*. 2016.

[44] M. Pham and D. B. Hoang. *SDN applications - The intent-based Northbound Interface realisation for extended applications*. pp. 372-377, June 2016.

[45] Arul Lawrence. RAJAN, P. et SELVAKUMAR. Internet traffic control system and mobile broadband communication system in atm networks. Journal of Current Research in Science, vol. 4, no 2, p. 6., 2016.

[46] et al. R.Vilalta, R.Casellas. Integrated sdn/nfv management and orchestration architecture for dynamic deployment of virtual sdn control instances for virtual tenant networks [invited]. Journal of Optical Communications and Networking. p. B62-B70., 2015.

[47] R.Casellas R.Vilalta and al. Software-defined networking : A comprehensive survey. Proceedings of the IEEE. p. 14-76, 2015.

[48] et al S. Jain. *"B4 : experience with a globally-deployed software defined wan"*. 2013.

[49] SEOK Woojin KIM Jin et al. SHAH, Syed Asif Raza. Camor : Congestion aware multipath optimal routing solution by using software-defined networking. In : International Conference on Platform Technology and Service (PlatCon). IEEE. p. 1-6., 2017.

[50] WAN Jiafu LIN Jiaxiang et al. SHU, Zhaogang. Traffic engineering in software-defined networking : Measurement and management. IEEE access,vol. 4, p. 3246-3256., 2016.

[51] Ning. SO. Bidirectional forwarding detection (bfd) protocol extension for detecting random traffic dropping. U.S. Patent No 8,989,020, 2015.

[52] W. Stallings. Software-defined networks and openflow. The Internet Protocol Journal. p. 1–6, 2013.

[53] JIA Zhiping ZHAO Mengying et al. SUN, Xiangshan. Multipath load balancing in sdn/ospf hybrid network. in :. IFIP International Conference on Network and Parallel Computing. Springer, Cham, p. 93-100., 2016.

[54] M. Tury. *Les risques d'OpenFlow et du SDN*. 2015.

[55] Ting-Sheng Chen ; Ding-Yuan Lee ; Tsung-Te Liu ; An-Yeu Wu. Dynamic reconfigurable ternary content addressable memory for openflow-compliantlow-power packetprocessing. IEEE Transactions on Circuits and Systems I : Regular Papers Vol. 63 , Issue. 10, 2016.

[56] X. Gong X. Que C. Shiduan Y. Hu, W. Wendong. Reliability-aware controller placement for software-defined networks, in :. 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)„ May 2013.

[57] X. Gong X. Que S. Cheng Y. Hu, W. Wang. Balanceflow : controller load balancing for openflow networks. in :. Proceedings of IEEE 2nd International Conference on Cloud Computing and Intelligent Systems. IEEE,p. 780–785., 2012.

[58] K. Wang Y. L. Lan and Y. H. Hsu. "dynamic load-balanced path optimization in sdn-based data center networks,". IEEE Inter. Conf. on Communication Systems, Networks and Digital Signal Processing, Prague, Czech Republic, pp. 1-6. July, 2016.

[59] SINGH Vishal WANG Ye et al. ZHANG, Yueping. Flowsense : light-weight networking sensing with openflow. U.S. Patent No 8,918,502, 23 déc., 2014.

[60] FANG Yaming et CUI Jie. ZHONG, Hong. Reprint of "lbbsrt : An efficient sdn load balancing scheme based on server response time". Future Generation Computer Systems, vol. 80, p. 409-416., 2018.

[61] Yichao Li Shui Yu Rongping Lin Zijing Cheng, Xiaoning Zhang and Lei He. Congestion-aware local reroute for fast failure recovery in software-defined networks. IEEE/OSA Journal of Optical Communications and Networking Volume : 9 , Issue : 11, 2017.