**RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE**
**MINISTÈRE DE L'ENSEIGNEMENT SUPERIEURE**
**ET DE LA RECHERCHE SCIENTIFIQUE**

**UNIVERSITÉ SAAD DAHLEB BLIDA**

**FACULTÉ DES SCIENCES**

**DÉPARTEMENT INFORMATIQUE**

# MÈMOIRE DE FIN D'ÉTUDES

**Pour l'obtention**

**D'un diplôme Master en informatique**

**Spécialité : Systèmes Informatiques et Réseaux**

# THÈME

## Étude et implémentation de l'approche Software Defined Network dans un réseau local

**Réalisé par :**

CHIKHI Mehdi

DJEMIL Adel

**Encadré par :** Mme. NEMMAR Mehdia

**Organisme d'accueil :** Sonatrach

**Jury :**

| | | |
|---|---|---|
| **Président :** | M. Ben yahya Mohamed | Maitre Assistant B à USDB1 |
| **Examinateur :** | Mme. Mancer Yasmine | Maitre Assistant A à USDB1 |
| **Promoteur :** | M. OULD-KHAOUA Mohamed | Professeur à USDB1 |

**Promotion :** 2018/2019

# 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,*

*Que dieu leur prête bonheur et longue vie.*

*Je dédie aussi ce travail à*

*Toutes ma famille surtout ma mère,*

*mes amis,*

*Tous mes professeurs qui m’ont enseigné*

*Et à tous ceux qui me sont chers.*

Mehdi.

# 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,*

*Que dieu leur prête bonheur et longue vie.*

*Je dédie aussi ce travail à.*

*Mes frères et sœurs*

*Toutes ma famille,*

*Mes amis,*

*Tous mes professeurs qui m’ont enseigné*

*Et à tous ceux qui me sont chers.*

Adel.

# Remerciements

*Nous tenons tout d'abord à remercier Allah le tout puissant et miséricordieux*
*qui nous a donné la force et la patience d'accomplir ce modeste travail.*
*En second lieu, nous tenons à remercier très chaleureusement notre encadreur*
*Madame Nemmar Mehdia pour ses aides précieuses, pour sa spontanéité et ses compétences professionnelles incontestables, durant toute cette période d'encadrement. « Elle est et elle sera pour nous un exemple de rigueur et de droiture dans l'exercice de la profession. »*
*Nous remercions également notre promoteur monsieur OULD-KHOUA pour la confiance placée en nous et pour avoir accepté de diriger ce travail. Aussi, nos vifs remerciements aux membres du jury pour l'intérêt accordé à*
*notre travail en l'examinant minutieusement et avec attention. Nous tenons à exprimer nos sincères remerciements à tous les professeurs qui nous ont enseigné et qui par leurs compétences nous ont soutenu pour la réussite dans nos études.*
*A nos familles et nos amis qui par leurs prières et leurs encouragements, on a*
*pu surmonter tous les obstacles.*
*Enfin, nous tenons également à remercier toutes les personnes qui ont participé de près ou de loin à la réalisation de ce travail.*

**Merci**

# Résumé

Le contrôle de la qualité de service (QoS) est un concept important dans les réseaux informatiques car il est lié à l'expérience de l'utilisateur final. Les garanties de qualité de service de bout en bout, en particulier, peuvent donner des garanties stables aux hôtes finaux. Avec l'émergence du Software Defined Network (SDN) et d'OpenFlow [1] comme standards les plus populaires, nous avons l'opportunité de réintroduire le concept de contrôle de la QoS. La nature centralisée et la programmabilité d'OpenFlow permettent un contrôle plus flexible et plus simple de la QoS.

Dans ce projet, nous proposons une méthode de garantie de bande passante pour OpenFlow. La principale considération de conception de la mèthode est de donner des garanties de bande passante par flux. Afin de maximiser davantage l'utilisation globale du réseau, les flux du meilleur effort sont autorisés d'utiliser toute bande passante inutilisée sur le réseau. Le traffic shaping est utilisé pour atteindre cet objectif. Pour s'assurer que cela n'affectera pas la bande passante garantie pour les flux QoS, nous analysons sa fiabilité sous Linux HTB, qui est utilisé comme mécanisme sous-jacent de la file d'attente OpenFlow. Ainsi, il est possible de garantir la bande passante et de maximiser ou minimiser l'utilisation de la bande passante en même temps.

Nous explorons également la possibilité d'utiliser le marquage DSCP pour donner la priorité à un flux par rapport à un autre. Le DSCP est configuré d’une façon automatique vers tous les commutateurs et ne donne la priorité qu’a certains trafics souhaités.

**Mots clefs :** SDN, QoS, OpenFlow, DSCP, Linux HTB.

# Abstract

Quality of service (QoS) control is an important concept in computer networks because it is linked to the end-user experience. End-to-end service quality guarantees, in particular, can provide stable guarantees to end hosts. With the emergence of Software Defined Network (SDN) and OpenFlow [1] as the most popular standards, we have the opportunity to reintroduce the concept of QoS control. The centralized nature and programmability of OpenFlow allows for more flexible and simpler QoS control.

In this project, we propose a bandwidth guarantee method for OpenFlow. The main design consideration of the method is to give bandwidth per stream guarantees. In order to further maximize overall network utilization, best effort flows are allowed to use any unused bandwidth on the network. Traffic shaping is used to achieve this objective. To ensure that this will not affect the guaranteed bandwidth for QoS flows, we analyze its reliability under Linux HTB, which is used as the underlying mechanism of the OpenFlow queue. This way, it is possible to guarantee bandwidth and maximize or minimize bandwidth usage at the same time.

We are also exploring the possibility of using DSCP marking to give priority to one flow over another. The DSCP is automatically configured to all switches and only gives priority to certain desired traffic.

**Keywords :** SDN, QoS, OpenFlow, DSCP, Linux HTB.

## ملخص

يعتبر التحكم في جودة الخدمة (QoS) مفهومًا مهمًا في شبكات الكمبيوتر لأنه مرتبط بتجربة المستخدم النهائي. ضمانات جودة الخدمة من طرف إلى طرف، على وجه الخصوص، يمكن أن توفر ضمانات مستقرة للمستخدمين النهائيين. مع ظهور الشبكات المعرّفة بالبرمجيات (SDN) وOpenFlow [1] باعتبارها أكثر المعايير شيوعًا، لدينا الفرصة لإعادة تقديم مفهوم التحكم في جودة الخدمة. تسمح الطبيعة المركزية وبرمجة OpenFlow بتحكم أكثر مرونة وأسهل في جودة الخدمة.

في هذا المشروع، نقترح طريقة لضمان عرض النطاق الترددي لـ OpenFlow. يتمثل التصميم الرئيسي لهذه الطريقة في تقديم ضمانات للنطاق الترددي لكل تيار. لزيادة الاستفادة الكلية من الشبكة إلى أقصى حد، يُسمح لأفضل تدفقات جهد باستخدام أي نطاق ترددي غير مستخدم على الشبكة. يستخدم تشكيل حركة المرور لتحقيق هذا الهدف. للتأكد من أن هذا لن يؤثر على النطاق الترددي المضمون لتدفقات جودة الخدمة، فإننا نحلل مدى موثوقيتها ضمن لينكس HTB، والذي يستخدم كآلية أساسية لقائمة انتظار OpenFlow. وبالتالي، من الممكن ضمان عرض النطاق الترددي وتعظيم أو تقليل استخدام عرض النطاق الترددي في نفس الوقت.

نحن نستكشف أيضًا إمكانية استخدام علامات DSCP لإعطاء الأولوية لتدفق واحد على الآخر. يتم تكوين DSCP تلقائيًا لجميع المحولات ويعطي الأولوية فقط لحركة المرور المطلوبة.

**كلمات أساسية** : الشبكات المعرّفة بالبرمجيات، جودة الخدمة، أوبن فلو، استخدام علامات DSCP، لينكس HTB.

# Table des matières

# Liste des Figures

# Liste des Tableaux

# Liste des abréviations

**API :** Application Programming Interface

**SBI :** Sounthbound Interface

**NBI :** Northbound Interface

**ASIC :** Circuits Intégrés Spécifiques A L'application

**Diffserv :** Differenciated Service

**DSCP :** Differentiated Services Code Point

**IETF :** Internet Engineering Task Force

**IntServ :** Integrated Service

**JVM :** Java Virtual Machine

**LAN :** Local Area Network

**NAT :** Network Address Translation

**NOS :** Network Operating System

**ODL :** OpenDaylight

**OF :** OpenFlow

**ONF :** Open Networking Foundation

**OVSDB :** Open VSwitch DataBase

**PHB :** Per-Hop Behavior

**QOS :** Quality Of Service

**RSVP :** Resource Reservation Protocol

**SDN :** Software Defined Network

**WAN :** Wide Area Network

**BE :** Beryllium

**FIFO :** First In First Out

**HTB :** Hierarchical Token Bucket

**OVS :** OpenvSwitch

**SLA** : Service-Level Agreement

**SFQ** : Stochastic Fairness Queuing

**OSGi** : Open Services Gateway initiative

# Introduction générale

La qualité de service (QoS) est la capacité d'un élément de réseau d'avoir un certain niveau d'assurance afin que ses exigences de trafic et de service puissent être satisfaites. La QoS reflète la performance qu'une application peut exiger et son expérience dans un réseau. Elle peut être considérée comme subjective, car les utilisateurs peuvent avoir des points de vue différents sur la qualité [2]. La QoS est particulièrement importante pour les applications avec des exigences strictes telles que la téléphonie et la diffusion multimédia. Néanmoins, le contrôle de la QoS n'a jamais été une tâche facile. La QoS de bout en bout telle que le service intégré (IntServ) [3] est jugée trop complexe et non évolutive. Malheureusement, comme la plupart du temps le réseau ne fonctionne pas à pleine capacité, il en résulte une faible utilisation du réseau [4]. C'est d'autant plus évident ces dernières années que l'émergence d'applications à haut débit oblige les fournisseurs de services à passer aux réseaux Gigabit.

Sans l'adaptation des applications utilisateur et un contrôle avancé du trafic, l'insuffisance des ressources réseau pourrait entraîner une congestion du réseau. Cela peut entraîner une dégradation du trafic important qui se manifeste par des pertes ou retards de trames. Ceci est particulièrement important dans les réseaux non gérés où il y a de plus en plus d'applications communicants à partir d'une variété d'utilisateurs différents.

De plus, avec des technologies comme la réalité virtuelle (VR) et la robotique, nous pouvons nous attendre à de nombreuses applications et services nouveaux. Dans un avenir pas si lointain, un employé pourrait accomplir sa tâche en utilisant une alimentation vidéo en direct à partir d'un ensemble d'yeux de caméra d'un robot qui se dirige vers une paire de lunettes VR. L'interactivité de la télécommande et des applications d'exploitation la rend très sensible au retard et aux jitters des trames. Ils exigent des normes élevées jusqu'au point où ils imposent des contraintes en temps réel au réseau.

Pour que ces nouvelles applications puissent répondre à leurs demandes, les réseaux de nouvelle génération devraient prendre conscience des applications et permettre aux applications qui sont très sensibles et qui ont les exigences les plus élevées de négocier avec un contrôleur réseau pour allouer des ressources et recevoir une garantie sur les ressources réseau reçues, par exemple, une garantie sur la bande passante maximum a utilisée.

Lorsqu'une application a effectué une négociation avec le contrôleur réseau, les ressources réseau sont allouées à un canal dédié pour cette application spécifique. Dans la pratique, cela se fait en configurant automatiquement les flux et les priorités pour respecter la garantie réseau promise.

Le Software Defined Network (SDN), en tant que nouveau paradigme du réseau, offre la possibilité de réintroduire le contrôle de la QoS sur les réseaux. La nature centralisée du SDN réduit considérablement la complexité généralement associée aux garanties de QoS. Grâce à son adoption et à l'appui d'entreprises chefs de file de l'industrie de la technologie, le SDN est sur la bonne voie pour être adopté comme norme de base. En ayant un fort contrôle de la QoS inclus dans le SDN, les réseaux du futur pourrait avoir un support natif de la QoS.

## Objectif du projet

L'entreprise Sonatrach vise a amélioré son système réseau LAN et de le gérer de manière centralisée tout en assurant une meilleure qualité de service. De ce fait, elle nous a proposé une solution moderne, moins couteuse et plus facile à contrôler par rapport à l'architecture traditionnelle, il s'agit d'une architecture SDN.

Pour avoir une bonne QoS, les caractéristiques suivantes doivent être appliquées :

- Donner des garanties de bande passante par flux.
- Prioriser correctement le trafic en fonction de la garantie, respectant ainsi les limites fixées.
- Autoriser le trafic excessif lorsque les ressources sont disponibles.

## Plan du mémoire

Ce travail est structuré en cinq chapitres :

- **Chapitre 1 :** présente la revue de littérature du software Defined Network.
- **Chapitre 2 :** présente une vue d'ensemble sur la qualité de service.
- **Chapitre 3 :** présente les deux méthodes de QoS utilisées (DSCP et traffic shaping) dans la réalisation de ce projet.
- **Chapitre 4 :** fournit une explication et analyse sur les outils SDN nécessaires pour l'implémentation de la QoS.
- **Chapitre 5 :** présente notre architecture SDN, puis l'implémentation de la QoS en utilisant le DSCP et traffic shaping ainsi que les tests générés, à la fin les résultats obtenus et une comparaison entre les résultats.

Nous terminerons par une conclusion et quelques perspectives pour des travaux futurs.

# Chapitre 1

# Introduction au SDN

## 1.1. Introduction

Durant ces dernières années, les réseaux informatiques classiques ont connu de grands défis qui résultent principalement des applications modernes déployées sur ces réseaux.

De ce fait, et suite à l'augmentation du nombre d'utilisateurs et des équipements, qui nécessitent un traitement distribué et l'adoption de l'information par plusieurs entreprises ont induit le besoin d'avoir des réseaux dynamiques qui offrent la possibilité de s'adapter rapidement aux changements des requis.

C'est dans ce contexte qu'est apparu le paradigme des réseaux définis par logiciels (SDN) qui permet principalement de s'adapter à la nature dynamique des différentes applications.

En effet, SDN permet principalement de centraliser la logique déterminant les politiques de gestion d'un réseau dans une ou plusieurs unités appelées contrôleurs. Ces contrôleurs communiquent avec le reste des équipements du réseau [5].

De ce fait, la définition d'une politique de gestion d'un réseau revient à écrire des programmes et les déployer dans les contrôleurs. Dans la plupart des cas, ces programmes seront compilés, en tenant compte de la topologie et des ressources disponibles, afin de générer les configurations nécessaires à chaque équipement du réseau pour mettre en œuvre les politiques désirées [5].

L'architecture SDN a été adoptée par plusieurs grandes entreprises et universités à savoir, **Google** et **Stanford**. D'autre part, les fabricants des équipements pour les réseaux tels que **Cisco**, **HP** et **Juniper** offrent désormais des solutions SDN qui permettent de gérer les centres de données [5].

## 1.2. Définition du SDN

Le terme SDN a été inventé à l'origine pour représenter les idées et le travail autour d'OpenFlow à l'Université de Stanford, Stanford, CA, USA [6]. Tel que défini à l'origine, SDN se réfère à une architecture de réseau où l'état de transmission dans le Data plane est géré à distance par un contrôle plane découplé du précédent. L'industrie réseau s'est à maintes reprises éloignée de cette vision originale du SDN en qualifiant de SDN tout ce qui concerne les logiciels. Nous tentons donc, dans cette section, de donner une définition beaucoup moins ambiguë du SDN.

SDN est une architecture réseau à quatre piliers :

- Les plans de commande et de données sont découplés. La fonctionnalité de contrôle est supprimée des périphériques réseau qui deviendront de simples éléments de transfert.

- Les décisions de transfert sont basées sur le flux et non sur la destination. Un flux est largement défini par un ensemble de valeurs de champs agissant comme critère de correspondance (filtre) et un ensemble d'actions (instructions). Dans le contexte SDN/OpenFlow, un flux est une séquence de trames entre une source et une destination. Toutes les trames d'un flux reçoivent des politiques de service identiques sur les dispositifs de transfert [7]. L'abstraction de flux permet d'unifier le comportement de différents types de périphériques réseau, y compris les routeurs, les commutateurs, les pares-feux et les middlebox [8]. La programmation du flux permet une flexibilité sans précédent, limitée uniquement aux capacités des tableaux de flux mis en œuvre [1].

- La logique de commande est transférée à une entité externe, le contrôleur SDN ou système d'exploitation réseau (NOS). Le NOS est une plate-forme logicielle qui s'appuie sur la technologie des serveurs de base et fournit les ressources et les abstractions essentielles pour faciliter la programmation des dispositifs de transfert sur la base d'une vue réseau abstraite et centralisée de manière logique. Son but est donc similaire à celui d'un système d'exploitation traditionnel.

- Le réseau est programmable au moyen d'applications logicielles s'exécutant sur le NOS qui interagit avec les dispositifs du plan de données sous-jacents. Il s'agit là d'une caractéristique fondamentale du SDN, considéré comme sa valeur principale.

## 1.3. Comparaison entre les réseaux SDN et réseaux traditionnels

La structure de l'Internet et les réseaux informatiques se composent généralement de différents dispositifs de réseau tels que routeur, commutateur et différents types de middlebox qui sont intégrés verticalement et conçus par des puces et ASIC (circuits intégrés spécifiques à l'application) avec un débit élevé et une fonction spécifique [9].

Pour la gestion et la configuration de ces périphériques réseau, un ensemble de commandes de ligne spécifiques et prédéfinies basées sur un système d'exploitation intégré est utilisé. Par conséquent, on peut faire valoir que la gestion d'un grand nombre de périphériques réseau est un grand défi qui est sujet à de nombreuses erreurs. Ainsi, les réseaux traditionnels souffrent d'importantes lacunes en matière de recherche et d'innovation, la fiabilité, l'extensibilité, la flexibilité et gestion. Depuis la naissance de l'internet, les réseaux se développent et de nouvelles technologies telles que le cloud, les réseaux sociaux et la virtualisation ont vues le jour, le besoin de réseaux avec une bande passante plus large, une plus grande accessibilité et une gestion dynamique plus élevée est devenu un problème critique [10].

Pour résoudre les problèmes et les limites des réseaux traditionnels, une structure a été proposer, connue sous le nom de SDN, où le contrôle du réseau est séparé du mécanisme de transmission et peut être programmé et contrôlé directement [9].

La différence architecturale entre l'Internet traditionnelle et le SDN est illustrée dans la Figure 1. Il représente clairement comment le contrôle est géré (logiquement) de manière centralisée, et le data plane est simplifié en simples éléments de transfert. Les commutateurs programmables du plan de données peuvent être implémentés dans le matériel ou le logiciel à condition qu'ils supportent le protocole OpenFlow [1] pour la communication et la configuration avec le contrôleur.

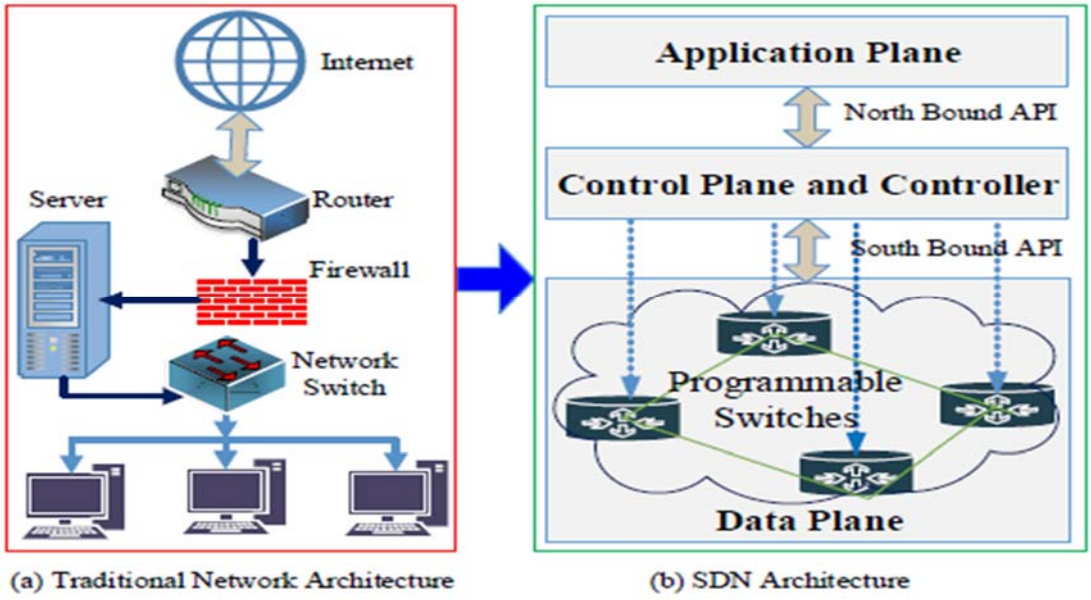



Figure 1 : Comparaison entre architecture SDN et réseau traditionnel [11]

## 1.4. Objective du SDN

L'objectif de ces innovations est de simplifier l'administration du réseau et à l'instar de ce que la virtualisation a réalisé dans le monde des serveurs, de rendre la consommation des ressources réseaux par les applications plus flexible. La Figure 2 représente les bénéfices essentiels du SDN.

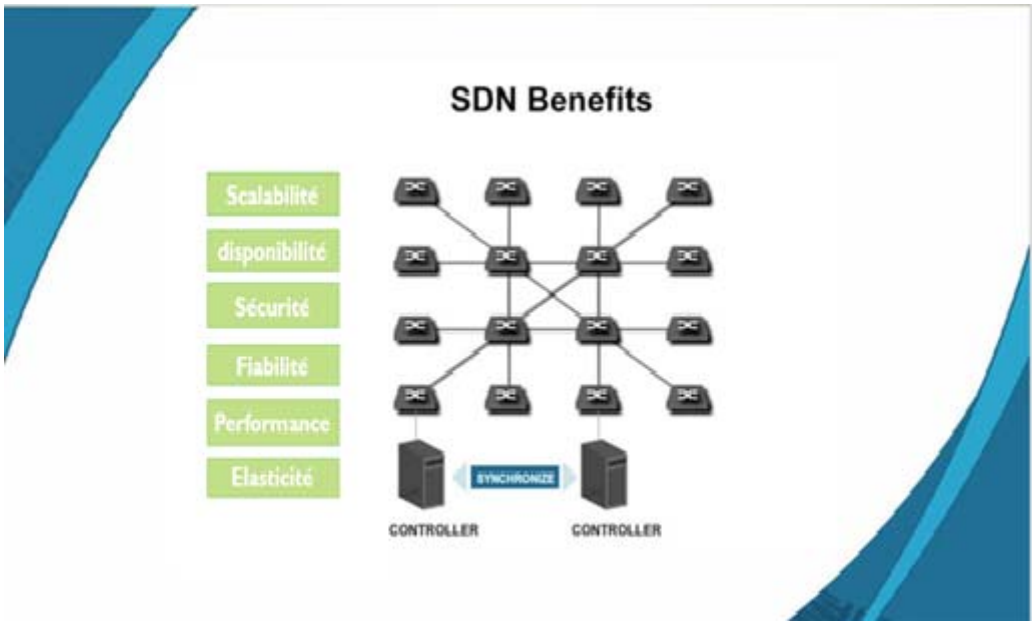



Figure 2 : Les bénéfices du SDN

***Scalabilité* :** Ceci définit la capacité du SDN, plus spécifiquement dans le plan de contrôle, à gérer et traiter une charge de travail croissante. La scalabilité vise à élargir la capacité du SDN en mettant en œuvre des mécanismes tels que le « devolving » **[12]**, le « clustering » **[13]** et le « high processing » **[14]** pour faire face à la charge croissante.

***Haute disponibilité* :** L'HD est un aspect important des services d'aujourd'hui qui devrait être disponible chaque fois qu'un client demande un service ou une ressource donnée. La disponibilité est habituellement exprimée en pourcentage du temps de disponibilité au cours d'une année donnée. L'indisponibilité des services peut généralement se produire en raison de pannes de réseau ou de pannes de système **[15]**, **[16]**. Les fournisseurs de réseaux déploient généralement des services de sauvegarde pour offrir une HD en implémentant du matériel serveur redondant, des composants d'OS serveur et de réseau.

***Sécurité* :** La sécurité du SDN consiste à protéger les informations contre le vol ou l'endommagement du matériel et des logiciels ainsi que contre l'interruption des services **[17]**, **[18]**. La sécurisation du SDN englobe la sécurité physique du matériel, ainsi que la prévention des menaces logiques qui peuvent provenir du réseau ou des données. Les vulnérabilités du SDN sont la porte d'entrée vers des attaques de sécurité intentionnelles ou accidentelles.

***Fiabilité* :** Le SDN est considéré comme fiable lorsqu'il notifie les défaillances des données en temps réel. Dans un tel réseau, il devrait y avoir une fiabilité minimale spécifiée pour l'acheminement des données critiques. Dans les implémentations actuelles, les contrôleurs SDN **[19]** doivent être capables de répondre en temps réel aux exigences de fiabilité et de ponctualité de l'acheminement.

***Performance* :** Le rendement (performance) fait référence à la quantité de tâches accomplies par les composantes SDN par rapport au temps et aux ressources (par ex., CPU et RAM) utilisés **[20]**. Il existe de nombreuses façons différentes de mesurer le rendement d'un réseau **[21]**, **[22]**, car chaque réseau est de nature et de conception différentes. En ce qui concerne le SDN, les mesures importantes sont la bande passante, le débit, la latence et le jitter.

***Elasticité* :** L'élasticité dans le SDN est la capacité d'assurer et de maintenir un niveau de service acceptable même en cas de défaillance d'un service, d'un réseau ou d'un nœud. Lorsqu'un élément SDN est défectueux, le réseau doit fournir un service opérationnel continu avec les mêmes performances. Afin d'accroître l'élasticité du SDN, il faut cerner les défis et les risques potentiels et y faire face pour protéger les services **[23]**.

## 1.5. L'architecture du SDN

La structure du SDN se compose de trois parties principales. Le niveau le plus bas, inclut le data plane. Au plus haut niveau il existe l'application plane, le control plane se trouve entre eux. La communication entre les contrôleurs et le data plane est géré via le SBI (Southbound Interface) qui se trouve au niveau du commutateur SDN et la communication entre les applications et les

contrôleurs est assurée par NBI (Northbound Interface) qui se trouve dans le control plane (Figure 3) [9].

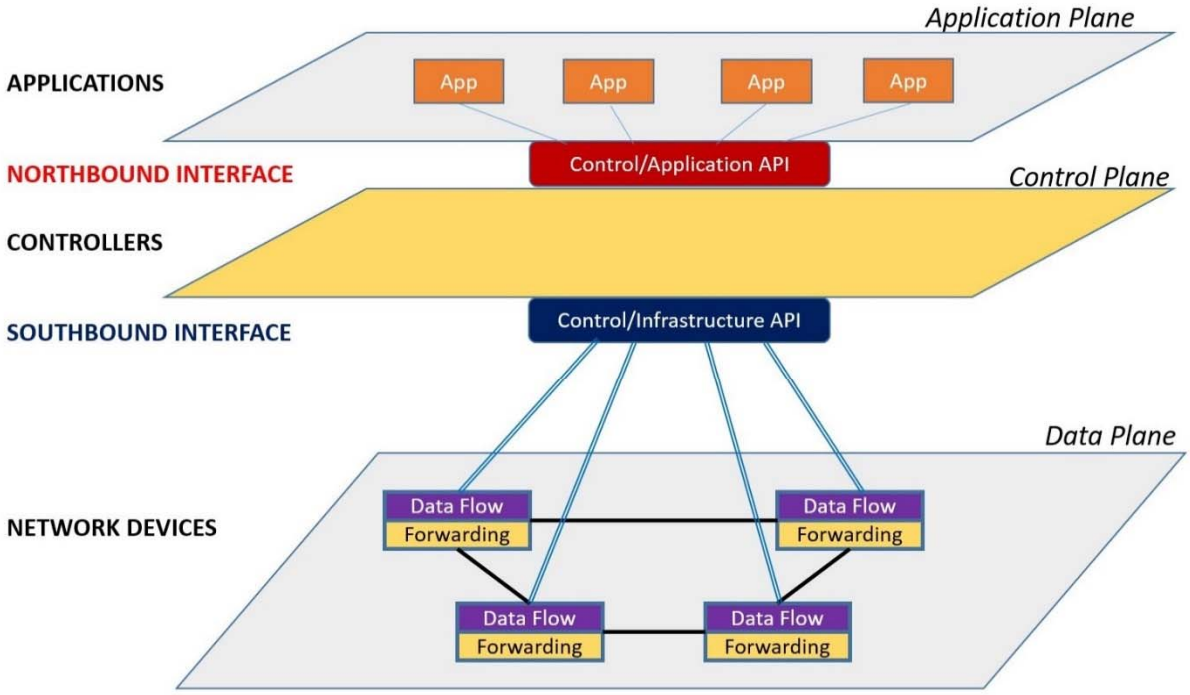



Figure 3 : L'architecture SDN [24]

L'utilisation de la répartition entre le control plane et le data plane, les applications suivent leur propre but particulier tel que la méthode de sécurité, la QoS, le trafic, l'ingénierie et les solutions de mesure et de surveillance des réseaux. De plus, le contrôleur aide pour atteindre leur objectif en contrôlant les commutateurs SDN par le biais de tables de flux. Dans d'autres le réseau s'adapte aux besoins des utilisateurs et, à l'aide du contrôleur et de l'API, les gestionnaires de réseau peuvent facilement contrôler le réseau automatiquement en ajoutant de nouvelles fonctions au control plane, sans effectuer de modifications dans le data plane.

### 1.5.1. Data Plane

Une infrastructure SDN, à l'instar d'un réseau traditionnel, est composée d'un ensemble d'équipements réseau (commutateurs, routeurs et Middleboxes). La principale différence réside dans le fait que ces dispositifs physiques traditionnels sont maintenant de simples éléments de transmission sans contrôle intégré ou logiciel pour prendre des décisions autonomes. L'intelligence réseau est retirée des dispositifs du plan de données vers un système de contrôle logiquement centralisé, c-à-d le NOS et les applications, comme le montre la Figure 3. Plus important encore, ces nouveaux réseaux sont construits sur des interfaces ouvertes et standard (par ex. OpenFlow), une approche cruciale pour assurer la compatibilité de la configuration et de la communication et

l'interopérabilité entre les différents dispositifs de data et plans de contrôle. En d'autres termes, ces interfaces ouvertes permettent aux entités contrôleur de programmer dynamiquement des dispositifs de transmission hétérogènes, ce qui est difficile dans les réseaux traditionnels, en raison de la grande variété d'interfaces propriétaires et les interfaces fermées et la nature distribuée de l'interface plan de contrôle.

Dans une architecture SDN/OpenFlow, il y a deux éléments principaux, les contrôleurs et les dispositifs de transmission, comme le montre la Figure 3. Un dispositif de plan de data est un élément matériel ou logiciel spécialisé dans la transmission de paquets, tandis qu'un contrôleur est une pile logicielle (le " cerveau réseau ") fonctionnant sur une plate-forme matérielle classique. Un dispositif de transmission compatible OpenFlow est basé sur un pipeline de tables de flux où chaque entrée d'une table de flux comporte trois parties : **(1)** une règle de correspondance ; **(2)** des actions à exécuter sur les paquets correspondants et **(3)** des compteurs qui tiennent des statistiques des paquets correspondants. Ce modèle de haut niveau et simplifié dérivé d'OpenFlow est actuellement la conception la plus répandue des dispositifs de plan de data SDN.

A l'intérieur d'un dispositif OpenFlow, un chemin à travers une séquence des tables de flux définit comment les paquets doivent être traités. Lorsqu'un nouveau flux arrive, le processus de recherche commence dans la première table et se termine soit par une correspondance dans l'une des tables du pipeline ou aucune correspondance (lorsqu'aucune règle n'est trouvée pour ce flux). Une règle de flux peut être définie en combinant différents champs d'adaptation, comme illustré à la Figure 4. S'il n'y a pas de règle par défaut, le flux sera rejeté. Cependant, le cas commun est d'installer une règle par défaut qui indique au commutateur d'envoyer le paquet au contrôleur (ou à la commande normale non-OpenFlow pipeline du commutateur). Le la priorité des règles suit le numéro d'ordre naturel des tables et de l'ordre des lignes dans la table de flux. Possibilité d'avoir les actions suivantes : **(1)** transférer le flux vers le(s) port(s) sortant(s) ; **(2)** l'encapsuler et l'envoyer au contrôleur ; **(3)** le rejeté ; **(4)** l'envoyer au pipeline de traitement normal et **(5)** l'envoyer à la table de flux suivante ou à des tables spéciales, telles que la table de conteur introduites dans la dernière version du protocole OpenFlow [1].

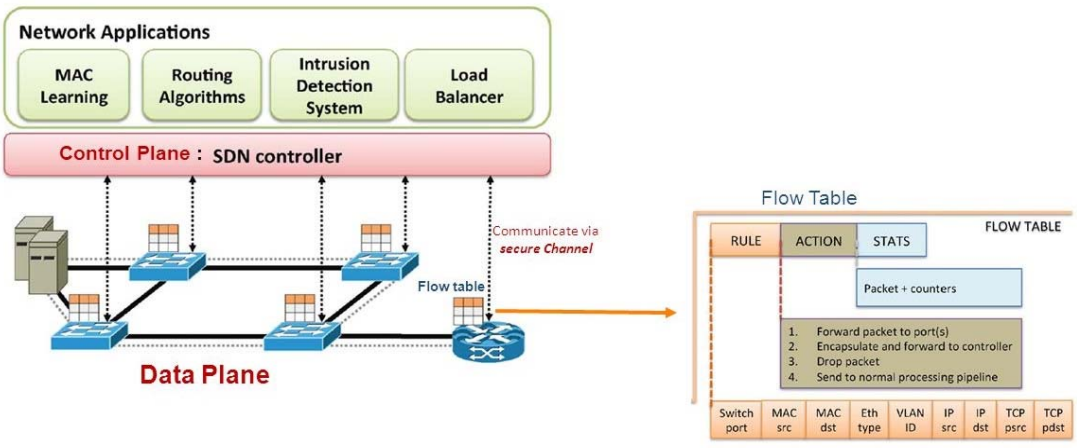



Figure 4 : OpenFlow activé sur les dispositifs SDN [25]

## 1.5.2. Southbound API

Southbound API est l'une des composantes les plus critiques d'un système SDN, qui fait le pont entre les dispositifs d'acheminement et le control plane. Il permet au contrôleur de contrôler le comportement du réseau en gérant les entrées du flux de tous les commutateurs sous-jacents. L'API Southbound fournit une interface commune pour les couches supérieures permettant au contrôleur d'utiliser les API vers le sud (par exemple OpenFlow, POF, OpFlex et OpenState) et les plugins de protocole pour gérer les périphériques physiques ou virtuels existants ou nouveaux (par exemple, SNMP, BGP et NetConf). Ceux-ci sont essentiels à la fois pour la compatibilité et l'hétérogénéité de l'arrière-plan. Par conséquent, sur le data plane, un mélange de dispositifs physiques, de dispositifs virtuels (par exemple, Open vSwitch, vRouter) et une variété d'interfaces de dispositifs (par exemple, OpenFlow, OVSDB [26], OF-Config, NetConf, et SNMP) peuvent coexister [27]. Actuellement, OpenFlow est le standard le plus accepté pour la norme de SBI [8].

### a. Protocole OpenFlow

OpenFlow est standardisé par l'Open Networking Foundation (ONF). OpenFlow est un protocole qui décrit l'interaction d'un ou plusieurs serveurs de contrôle avec les commutateurs compatibles OpenFlow. Un contrôleur OpenFlow installe les entrées de tables de flux dans les commutateurs, de sorte que ces commutateurs peuvent transférer le trafic en fonction de ces entrées. Ainsi, les commutateurs OpenFlow dépendent de la configuration des contrôleurs [8]. La Figure 5 représente la position schématique du protocole OpenFlow entre le plan de commande et le plan de données.

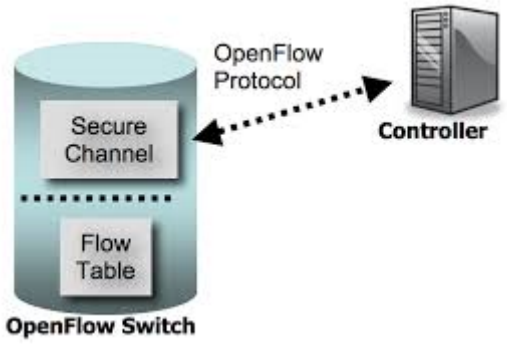



Figure 5 : Le protocole OpenFlow [28]

N. McKeown *et. al.* [1] expliquent comment la plupart des nouvelles idées du milieu de la recherche en réseau ne sont pas testées. C'est ainsi que l'infrastructure du réseau a stagné en raison de la réticence des opérateurs de réseau à expérimenter avec les données de production. Cet article présente les premiers arguments solides en faveur de la flexibilité du réseau et propose la première version du protocole OpenFlow et des commutateurs compatibles OpenFlow. Selon le document, cela permettrait d'innover dans les réseaux de campus du monde entier. L'article a été publié en 2008 et avec le recul de ces dernières années, l'affirmation s'est révélée vraie. Le routage basé sur IP

pose de nombreux problèmes, en particulier en ce qui concerne le nombre croissant d'appareils et l'épuisement de leurs adresses. La communauté de recherche sur les réseaux tente depuis longtemps de s'éloigner du routage IPv4.

OpenFlow peut acheminer des paquets sur la base de plusieurs critères, y compris IP, MAC, VLAN et d'autres adressages, et ainsi fournir un solide remplacement pour le routage sur IP. Le protocole OpenFlow a subi des améliorations rapides depuis la sortie de nombreuses nouvelles versions. Le Tableau 1 ci-dessous résume quelques-unes des améliorations significatives d'OpenFlow :

| Version OpenFlow | Support Description |
|---|---|
| **OF v1.1** | MPLS, VLAN, Multipath, Tables grouper, Tables Multi flux et Files d'attentes |
| **OF v1.2** | In-Match, Packet-In, En-têtes extensibles, Flux Flexible match et IPV6 |
| **OF v1.3** | Tunneling, Tables de compteurs et Champs supplémentaires dans les tables de flux (Priorité, Timeouts, Cookie) |
| **OF v1.4** | Gestion optique des ports, tables synchronisées, surveillance des flux |
| **OF v1.5** | Tables en sortie, statistiques d'entrée des flux, appariement des drapeaux TCP en fonction du type de paquet et du type de canalisation |

Tableau 1 : Le développement du protocole OpenFlow

## b. Dispositifs OpenFlow

Plusieurs dispositifs de transmission compatibles OpenFlow sont disponibles sur le marché, à la fois comme produits commerciaux et open source (voir Tableau 2). Il existe de nombreux commutateurs et routeurs OpenFlow prêts à l'emploi, prêts à être déployés, parmi d'autres appareils. La plupart des commutateurs disponibles sur le marché ont une mémoire relativement petite, avec jusqu'à 8000 entrées. Néanmoins, la situation évolue rapidement. Certains des derniers appareils commercialisés sur le marché vont bien au-delà de ce chiffre. Les commutateurs Gigabit Ethernet (GbE) à des fins commerciales communes prennent déjà en charge jusqu'à 32 000 flux de couche 2 (L2) + couche 3 (L3) ou 64 000 flux de correspondance exactement L2/L3. Les commutateurs Enterprise de classe 10GbE sont livrés avec plus de 80 000 entrées de flux de couche 2. D'autres dispositifs de commutation utilisant des puces à haute performance (par ex. EZchip NP-4) fournissent une mémoire optimisée qui supporte de 125 000 à 1 000 000 d'entrées de table de flux. C'est un signe clair que la taille des tables de flux augmente à un rythme visant à répondre aux besoins des futurs déploiements de SDN. Les fabricants de matériel de réseau ont produit divers types de dispositifs compatibles OpenFlow, comme le montre le Tableau 2. Ces appareils vont de l'équipement pour les petites entreprises (par ex., les commutateurs GbE) à l'équipement de « data center » de haute qualité.

Une observation intéressante est le nombre de petites entreprises en démarrage qui se consacrent au SDN, comme Big Switch, Pica8, Cyan, Plexxi et NoviFlow. Cela semble impliquer que le SDN est en train de créer un marché de réseau plus compétitif et plus ouvert, l'un de ses objectifs initiaux.

D'autres effets de cette ouverture déclenchée par le SDN comprennent l'émergence de ce qu'on appelle les " commutateurs à métal « bare » " ou " commutateurs « whitebox » ", où les logiciels et le matériel sont vendus séparément et l'utilisateur finale est libre de charger un système d'exploitation de son choix.

| Group | Product | Type | Maker/Developer | Version | Short description |
|---|---|---|---|---|---|
| Hardware | 8200zl and 5400zl | chassis | Hewlett-Packard | v1.0 | Data center class chassis (switch modules). |
| | Arista 7150 Series | switch | Arista Networks | v1.0 | Data centers hybrid Ethernet/OpenFlow switches. |
| | BlackDiamond X8 | switch | Extreme Networks | v1.0 | Cloud-scale hybrid Ethernet/OpenFlow switches. |
| | CX600 Series | router | Huawei | v1.0 | Carrier class MAN routers. |
| | EX9200 Ethernet | chassis | Juniper | v1.0 | Chassis based switches for cloud data centers. |
| | EZchip NP-4 | chip | EZchip Technologies | v1.1 | High performance 100-Gigabit network processors. |
| | MLX Series | router | Brocade | v1.0 | Service providers and enterprise class routers. |
| | NoviSwitch 1248 | switch | NoviFlow | v1.3 | High performance OpenFlow switch. |
| | NetFPGA | card | NetFPGA | v1.0 | 1G and 10G OpenFlow implementations. |
| | RackSwitch G8264 | switch | IBM | v1.0 | Data center switches supporting Virtual Fabric and OpenFlow. |
| | PF5240 and PF5820 | switch | NEC | v1.0 | Enterprise class hybrid Ethernet/OpenFlow switches. |
| | Pica8 3920 | switch | Pica8 | v1.0 | Hybrid Ethernet/OpenFlow switches. |
| | Plexxi Switch 1 | switch | Plexxi | v1.0 | Optical multiplexing interconnect for data centers. |
| | V330 Series | switch | Centec Networks | v1.0 | Hybrid Ethernet/OpenFlow switches. |
| | Z-Series | switch | Cyan | v1.0 | Family of packet-optical transport platforms. |
| Software | contrail-vrouter | vrouter | Juniper Networks | v1.0 | Data-plane function to interface with a VRF. |
| | LINC | switch | FlowForwarding | v1.3 | Erlang-based soft switch with OF-Config 1.1 support. |
| | ofsoftswitch13 | switch | Ericsson, CPqD | v1.3 | OF 1.3 compatible user-space software switch implementation. |
| | Open vSwitch | switch | Open Community | v1.0 | Switch platform designed for virtualized server environments. |
| | OpenFlow Reference | switch | Stanford | v1.0 | OF Switching capability to a Linux PC with multiple NICs. |
| | OpenFlowClick | vrouter | Yogesh Mundada | v1.0 | OpenFlow switching element for Click software routers. |
| | Switch Light | switch | Big Switch | v1.0 | Thin switching software platform for physical/virtual switches. |
| | Pantou/OpenWRT | switch | Stanford | v1.0 | Turns a wireless router into an OF-enabled switch. |
| | XorPlus | switch | Pica8 | v1.0 | Switching software for high performance ASICs. |

Tableau 2 : OpenFlow sur les dispositifs Hardware et Software

## c. Protocole OVSDB

OpenFlow est un protocole Southbound qui permet la communication entre le contrôleur et le commutateur. La gestion du switch lui-même n'est pas gérée par OpenFlow. Pour cela, deux protocoles différents peuvent être utilisés, OVSDB et OF-Config.

OF-Config est un protocole de gestion développé par l'ONF et est censé fonctionner avec n'importe quel périphérique compatible OpenFlow, tandis que le protocole OVSDB est spécifiquement développé pour OpenvSwitch. OVSDB fonctionne avec l'implémentation logicielle et matérielle d'OVS, comme Pica8 [29].

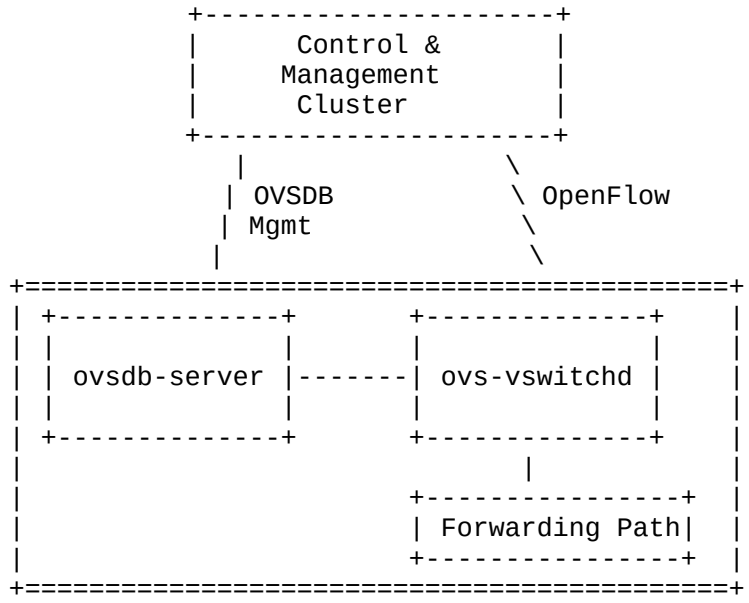



Figure 6 : L'architecture OVSDB [26]

OVSDB gère les opérations de commutation telles que la création d'interfaces, la définition de politiques QoS ou l'arrêt d'un port physique. La Figure 6 illustre l'architecture de l'OVSDB. Une instance OVS est constituée d'un serveur de base de données (ovsdb-server) et d'un démon vswitch (ovs-vswitchd). Le cluster de gestion et de contrôle est constitué des gestionnaires OVSDB et du contrôleur OpenFlow, qui peuvent être situés dans les mêmes appareils ou dans des appareils différents. Pendant que le contrôleur communique avec les commutateurs via le canal OpenFlow, le serveur OVSDB parle avec son gestionnaire via le protocole de gestion OVSDB. Le démon de commutation OVSDB, situé dans un commutateur, surveille la base de données pour les ajouts, les suppressions et les modifications apportées à ces informations. Tout changement dans la base de données est appliqué au commutateur. Le serveur OVSDB stocke les informations sur les commutateurs sous forme de base de données. La configuration dans OVSDB est permanente ; le switch ne perdra pas sa configuration en cas de redémarrage du commutateur.

L'OVSDB est formalisée dans la RFC7047 [26]. De nombreux contrôleurs OpenFlow, tels que OpenDaylight et Ryu ont intégré une API pour communiquer avec OVSDB. Cette prise en charge permet d'intégrer la gestion des commutateurs dans l'application OpenFlow.

La configuration des commutateurs dans OVSDB est stockée sous forme de base de données. La Figure 7 montre le schéma de la base de données. Chaque nœud représente une table dans la base de données, tandis que les bords représentent la relation entre les tables. Les tableaux qui font partie du "root set" sont représentés avec des bordures doubles. Root set est une table dont les entrées ne seront pas automatiquement supprimées lorsqu'elles ne sont pas accessibles depuis la table OpenvSwitch. Chaque bord du graphique part du tableau qui le contient et pointe vers le tableau que sa valeur représente. Les bords sont étiquetés avec leur nom de colonne. Les symboles à côté de l'étiquette indiquent le nombre de valeurs autorisées : ? pour zéro ou un, * pour zéro ou plusieurs, + pour un ou plusieurs.

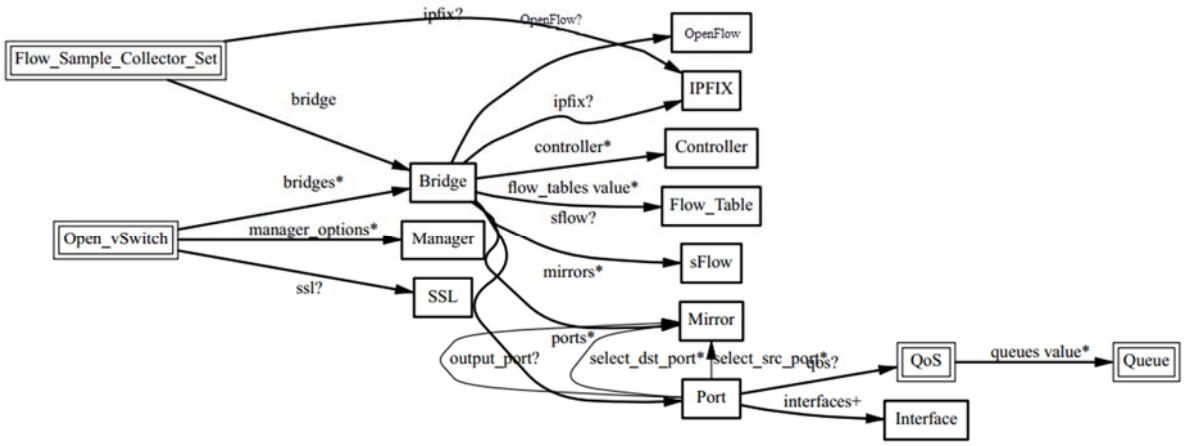



Figure 7 : Le schéma de base de données de l'OpenvSwitch [30]

## 1.5.3. Contrôle plane

Les systèmes d'exploitation traditionnels fournissent des abstractions (par ex. API de programmation de haut niveau) pour accéder aux périphériques de niveau inférieur, gérer l'accès simultané aux ressources sous-jacentes (par ex. disque dur, adaptateur réseau, CPU, mémoire) et fournir des mécanismes de protection. Ces fonctionnalités et ressources sont des outils clés pour accroître la productivité et faciliter la vie des développeurs de systèmes et d'applications. Leur utilisation généralisée a grandement contribué à l'évolution de divers écosystèmes (par ex., les langages de programmation) et au développement d'une variété d'applications.

En revanche, jusqu'à présent les réseaux ont été gérés et configurés à l'aide d'ensembles d'instructions de niveau inférieur spécifiques à l'appareil et de systèmes d'exploitation réseau propriétaires pour la plupart fermés (par ex. Cisco IOS et Juniper JunOS). De plus, l'idée de systèmes d'exploitation abstrayant les caractéristiques spécifiques des appareils et fournissant de manière transparente, des fonctionnalités communes est encore largement absente dans les réseaux. Par exemple, de nos jours les concepteurs de protocoles de routage doivent faire face à des algorithmes distribués complexes pour résoudre des problèmes de réseau. Les praticiens du réseau sont donc toujours en train de résoudre les mêmes problèmes encore et encore.

Le SDN est promis de faciliter la gestion du réseau et d'alléger le fardeau de la résolution des problèmes de réseau au moyen du contrôle logiquement centralisé offert par un système d'exploitation réseau (NOS) [7]. Comme avec les systèmes d'exploitation traditionnels, la valeur cruciale d'un NOS est de fournir des abstractions, des services essentiels et des interfaces de programmation d'applications (API) communes aux développeurs. Les fonctionnalités génériques telles que les informations sur l'état et la topologie du réseau, la découverte des périphériques et la distribution de la configuration du réseau peuvent être fournies en tant que services du NOS. Avec les NOS, pour définir les politiques réseau, un développeur n'a plus besoin de se soucier des détails de bas niveau de la distribution des données entre les éléments de routage, par exemple. On peut soutenir que de tels systèmes peuvent créer un nouvel environnement capable de favoriser

l'innovation à un rythme plus rapide en réduisant la complexité inhérente à la création de nouveaux protocoles et applications réseau.

Un NOS (ou contrôleur) est un élément critique dans une architecture SDN car il est la pièce maîtresse de la logique de contrôle (applications) pour générer la configuration réseau sur la base des politiques définies par l'opérateur réseau. Semblable à un système d'exploitation traditionnel, la plate-forme de contrôle résume les détails de niveau inférieur de connexion et d'interaction avec les dispositifs de transfert (c'est-à-dire de matérialisation des politiques réseau).

### 1.5.4. Northbound API

La Southbound API a déjà une proposition largement acceptée (OpenFlow), mais une Northbound API commune reste un problème ouvert. Pour l'instant, il est peut-être encore un peu trop tôt pour définir une Northbound API standard, car les cas d'utilisation sont encore en cours de développement [31]. Quoi qu'il en soit, il faut s'attendre à ce qu'une Northbound API commune apparaisse à mesure que le SDN évolue. Une abstraction qui permettrait aux applications réseau de ne pas dépendre d'implémentations spécifiques est importante pour explorer tout le potentiel du SDN.

Les contrôleurs existants tels que Floodlight, Trema, NOX, Onix et OpenDaylight proposent et définissent leurs propres Northbound API [32], [31]. Cependant, chacune d'entre elles a ses propres définitions spécifiques. Les langages de programmation tels que Frenetic [33], NetCore [34], Nettle [35], abstraient également les détails internes des fonctions du contrôleur et le comportement du plan de données des développeurs d'application. De plus, les langages de programmation peuvent fournir un large éventail d'abstractions et de mécanismes puissants tels que la composition des applications, la tolérance aux pannes dans le data plane et une variété d'éléments de base pour faciliter le développement des modules logiciels et applications.

D'autres propositions utilisent des approches différentes pour permettre aux applications d'interagir avec les contrôleurs. La plate-forme de contrôle yanc [36] explore cette idée en proposant une plate-forme de contrôle générale basée sur Linux et des abstractions telles que le système de fichiers virtuel (VFS). Cette approche simplifie le développement d'applications SDN car les programmeurs sont capables d'utiliser un concept traditionnel (fichiers) pour communiquer avec des appareils et sous-systèmes de niveau inférieur.

En fin de compte, il est peu probable qu'une seule Northbound API en sorte gagnante, car les exigences des différentes applications réseau sont très différentes. Les API pour les applications de sécurité sont susceptibles d'être différentes de celles pour les applications de routage. Le travail architectural de l'ONF [37] inclut la possibilité de fournir des APIs vers le nord pour permettre un contrôle dynamique et granulaire des ressources réseau à partir des applications clients, éventuellement à travers différentes frontières commerciales et organisationnelles.

### 1.5.5. Application plane

Comme le montre la Figure 3, la couche d'application se trouve au-dessus de la couche de contrôle. Grâce à la couche de contrôle, les applications SDN peuvent accéder facilement à une vue globale du réseau avec un état instantané par le biais de la Northbound, par exemple, le protocole

ALTO (Application Layer Traffic Optimization) [38]. Dotées de ces informations, les applications SDN peuvent mettre en œuvre par programmation des stratégies pour manipuler les réseaux physiques sous-jacents en utilisant un langage de haut niveau fourni par la couche de contrôle. Dans ce domaine, SDN propose le modèle "Platform as a Service" pour la mise en réseau [39].

## 1.6. Types de contrôleurs SDN

***Beacon* [40]:** Beacon est un contrôleur SDN qui a été introduit en 2010. Il a été utilisé dans plusieurs recherches projets. C'est un contrôleur basé sur Java. Il peut fonctionner sur de nombreuses plates-formes, y compris Linux multicœurs haut de gamme, et les téléphones Android.

***DISCO* [41]:** Disco est un contrôleur distribué. Il est principalement utilisé pour les réseaux WAN et les réseaux superposés. Chaque contrôleur est responsable d'un domaine réseau. Les régulateurs communiquent entre eux par l'intermédiaire d'un canal inter-contrôleur. DISCO peut s'adapter dynamiquement à différentes topologies de réseaux hétérogènes.

***IRIS* [42]:** IRIS est une plate-forme de contrôleur SDN pour contrôler la base OpenFlow. Il peut gérer un grand réseau. IRIS prend en charge les architectures qui sont évolutives horizontalement. Ainsi, les serveurs peuvent être ajoutés dynamiquement au cluster de contrôleurs. Cela augmente le facteur de performance du contrôle plane.

***Maestro* [43]:** Maestro est le premier système de contrôle OpenFlow qui exploite le parallélisme. Dans Maestro, les programmeurs peuvent modifier la fonctionnalité du data plane en écrivant des programmes simples à filetage unique. Maestro a son propre ensemble des conceptions et des techniques qui aident le protocole OpenFlow. Il s'agit d'un contrôleur basé sur Java et très portable pour différents systèmes d'exploitation et architectures.

***OpenDaylight* [44]:** OpenDaylight s'inspire de Beacon. Il s'agit d'un contrôleur basé de Java dérivé de Beacon. Il supporte OpenFlow et d'autres Southbound APIs. L'OpenDaylight est présent dans sa propre machine virtuelle Java (JVM).

***NOX* [7]:** NOX est la première plate-forme SDN OpenFlow Controller pour la construction d'applications de contrôle réseau. Il a été initialement développé par Nicira Networks, aux côtés d'OpenFlow. Plus tard, NOX a été donné à la communauté SDN. Les applications peuvent être en Python ou en C++ et peuvent être chargées dynamiquement.

***POX* [45]:** Pox est similaire au NOX Controller. POX est un contrôleur SDN qui permet le développement et le prototypage rapides du réseau. Il suit le protocole OpenFlow, et qui sert a joué le role d'un framework entre les commutateurs OpenFlow.

***Floodlight*** **[46]:** Floodlight est un contrôleur SDN Open source. Il s'agit d'un contrôleur OpenFlow de classe entreprise, basé sur Java. Il fonctionne à la fois avec les commutateurs physiques et les commutateurs virtuels qui utilisent le protocole OpenFlow.

***Ryu*** **[47]:** Ryu est un contrôleur SDN basé sur des composants. Ryu signifie "flux" en japonais. Ryu fournit de nombreux composants logiciels et API étendus pour créer et gérer les applications réseau. Ryu supporte les protocoles comme OpenFlow, Netconf, OF-config, etc. Ryu est complètement implémenté en langage Python. Il est sous licence Apache 2.0.

Le tableau 3 effectue une comparaison entre les propriétés des différents contrôleurs SDN.

| **Propriétés** | **Contrôleurs SDN** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Beacon | DISCO | IRIS | Maestro | OpenDaylight | NOX | POX | Floodlight | Ryu |
| **Support du Parallélisme** | Oui | Oui | Oui | Oui | Oui | Non | Non | Oui | Oui |
| **Protocole** | OpenFlow | OpenFlow | OpenFlow OVSDB | OpenFlow | OpenFlow OVSDB | OpenFlow | OpenFlow | OpenFlow | OpenFlow OVSDB |
| **L'architecture** | Centralisée | Distribuée | Centralisée | Centralisée | Distribuée | Centralisée | Centralisée | Centralisée | Centralisée |
| **Langage de programmation** | Java | Java | Java | Java | Java | C++ | Python | Java | Python |
| **API supportés** | Ad-hoc API | REST API | REST API | Ad-hoc API | REST API | Ad-hoc API | Ad-hoc API | REST API | Ad-hoc API |
| **Plateformes supportées** | Linux Windows | Linux Windows | Linux Windows | Linux Windows Mac OS | Linux | Linux Windows Mac OS | Linux Windows Mac OS | Linux Windows Mac OS | Linux |

Tableau 3 : Comparaison entre les propriétés des contrôleurs SDN

## 1.7. Conclusion

Au cours de ce chapitre, on a mis en évidence le Software Defined Network qui change tout le point de vue précédemment acquis sur le réseau traditionnel. On a commencé par introduire ces avantages principaux dans le réseau, puis on est passé à une comparaison entre le SDN et l'architecture traditionnelle, ensuite on a présenté son architecture en détail de bas en haut, en traitant les composants essentiels de cette solution pour appliquer ses concepts à notre contexte. A la fin on a présenté les contrôleurs SDN les plus répandus avec une comparaison entre chacun d'eux.

Dans le chapitre suivant on explore les méthodes de la qualité de service.

# Chapitre 2

# Qualité de Service

## 2.1. Introduction

La qualité de service (QoS) sur les réseaux est un domaine de recherche et d'amélioration continue depuis des décennies. Les réseaux traditionnels, qui n'avaient pas été initialement conçus en fonction de la qualité de service, ont ensuite été complétés par de nombreuses méthodes permettant d'obtenir l'accord de performances souhaités [48].

Dans ce chapitre on présente la qualité de service en détaillant ses niveaux et méthodes appliquées dans les réseaux.

## 2.2. Définition de la QoS

La qualité de service peut se définir comme étant « l'ensemble des phénomènes pouvant influencer les performances du service qui détermine le degré de satisfaction de l'utilisateur de ce service » [49].

La QoS est généralement considérée comme étant la capacité d'un réseau à fournir les services requis pour un trafic réseau sélectionné. L'objectif principal de la QoS est de fournir une priorité, en ce qui concerne les paramètres de la QoS ils sont limités aux critères de :

- **Bande passante :** La bande passante est la capacité d'une liaison de communication réseau à transmettre la quantité maximale de données d'un point à un autre en un temps donné. La QoS optimise le réseau en gérant la bande passante et en établissant des priorités pour les applications qui nécessitent plus de ressources que les autres [50].

- **Délai :** Le délai d'un réseau spécifie le temps qu'il faut pour qu'un bit de données circule sur le réseau d'un nœud à un autre. Elle est généralement mesurée en multiples ou en fractions de secondes. Le délai peut varier légèrement en fonction de l'emplacement de la paire de nœuds de communication spécifique. Bien que les utilisateurs ne se soucient que du délai total d'un réseau, les ingénieurs doivent effectuer des mesures précises [51].

- **Jitter :** Le jitter est le résultat de la congestion du réseau, de la dérive temporelle et des changements d'itinéraire. Pour ainsi dire un jitter trop instable peut dégrader la qualité de la communication vocale et vidéo [52].

- **Packet loss :** C'est un phénomène qui se produit lorsque les liaisons réseau deviennent encombrées et que les routeurs et les commutateurs commencent à rejeter des paquets. Lorsque des paquets sont rejetés pendant une communication en temps réel, comme des appels vocaux ou vidéo, ces sessions peuvent présenter des lacunes dans la transmission [53].

## 2.3. Niveaux de service

Le terme niveau de service définit le niveau d'exigence pour la capacité d'un réseau à fournir un service d'un nœud à un autre ou de bout en bout avec un trafic donné. Nous avons généralement trois niveaux de QoS :

- **Best effort :** Le Best effort ne fournis aucune différenciation entre plusieurs flux réseaux et ne permettant aucune garantie. Ce niveau de service est appelé « lack of QoS ».

- **Soft QoS :** La Soft QoS (ou DiffServ) permet de définir des niveaux de priorité aux différents flux réseau sans toutefois fournir une garantie stricte.

- **Hard QoS :** La Hard QoS (ou IntServ) consistant à réserver des ressources réseau pour certains types de flux. Le principal mécanisme utilisé pour obtenir un tel niveau de service est RSVP (Resource reSerVation Protocol).

### 2.3.1. IntServ

Le modèle Integrated Services (IntServ) est une architecture qui spécifie les éléments garantissant la QoS de bout en bout dans les réseaux. Il est proposé pour assurer la QoS au réseau sans perturber le bon fonctionnement du protocole IP [3].

Les composantes clés de cette architecture offrent deux nouvelles classes de services supplémentaires par rapport au service traditionnel du best-effort. Le premier est la charge contrôlée qui n'offre aucune garantie sur le délai de bout en bout et le « Packet loss » mais il garantit l'absence de congestion des trafics du réseau. Le second type est le service garanti la bande passante et le délai d'acheminement limité. Ce modèle repose sur le protocole de signalisation RSVP [54] d'acheminement des informations routeurs répondant aux demandes et aux exigences des flux (voir Figure 8). Les routeurs assurent les fonctionnalités de contrôle d'admission de flux et de réservation de ressources. RSVP est le protocole qu'utilisent les applications pour réserver des ressources du réseau.

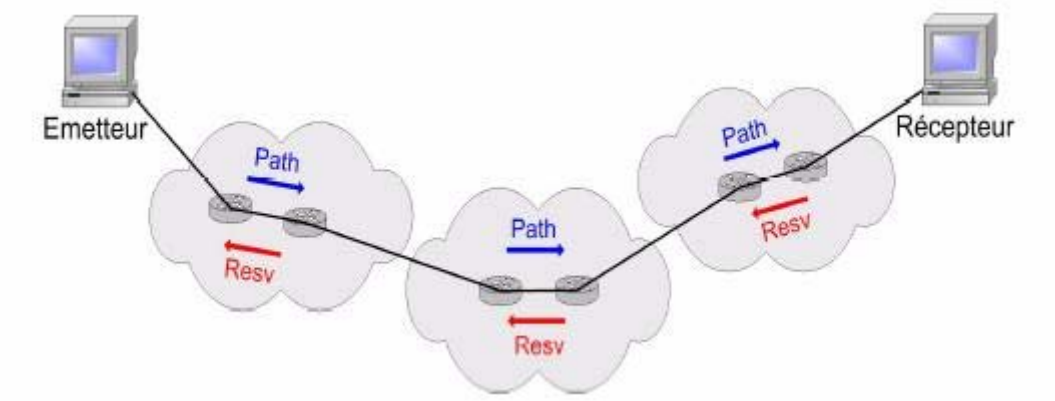



Figure 8 : Le protocole RSVP [55]

Bien qu'IntServ soit un moyen efficace de fournir une garantie Hard QoS, il n'a jamais décollé comme une solution tout-en-un pour les exigences QoS du réseau. Cette situation s'explique principalement par les inconvénients suivants d'IntServ :

- Tous les routeurs le long du Traffic doivent supporter IntServ.
- Chaque routeur le long du chemin a besoin de stocker de nombreux états qui deviennent très coûteux au fur et à mesure que le réseau s'étend.
- Le rafraîchissement périodique de la création d'une nouvelle QoS et le démantèlement de celles qui ne sont pas utilisées nécessitent la transmission d'un grand nombre de messages sur le réseau.

Pour ces raisons, IntServ n'a jamais été déployé sur les grands réseaux. Il n'est tout simplement pas évolutif. Toutefois, il a été largement utilisé dans les réseaux locaux et les réseaux de petites entreprises. Le principal avantage d'IntServ est qu'il offre une meilleure garantie et peut être utilisé pour fournir l'identification des classes de service.

### 2.3.2. DiffServ

La scalabilité était le principal obstacle à la croissance d'IntServ en tant que mécanisme de QoS [56]. DiffServ [57] a été introduit pour résoudre ce problème. Pendant qu'IntServ traitait le trafic de bout en bout, DiffServ appliquait des politiques à chaque saut. C'est ce qu'on appelle le « Per-Hop Behavior » (PHB). Contrairement à IntServ qui fonctionnait par flux, DiffServ fonctionne sur des flux agrégés. Les paquets peuvent être classés à l'aide des bits DSCP (Differentiated Services Code Points) successeur des bits Type Of Service (TOS) contenus dans l'en-tête du paquet. Tous les paquets avec le même en-tête reçoivent le même traitement, quels que soient les flux dont ils font partie. Contrairement à IntServ, DiffServ devient ainsi plus facile à mettre en œuvre et à maintenir sans trop de frais de transmission de messages et d'accord global. La Figure 9 représente le fonctionnement du protocole DiffServ.

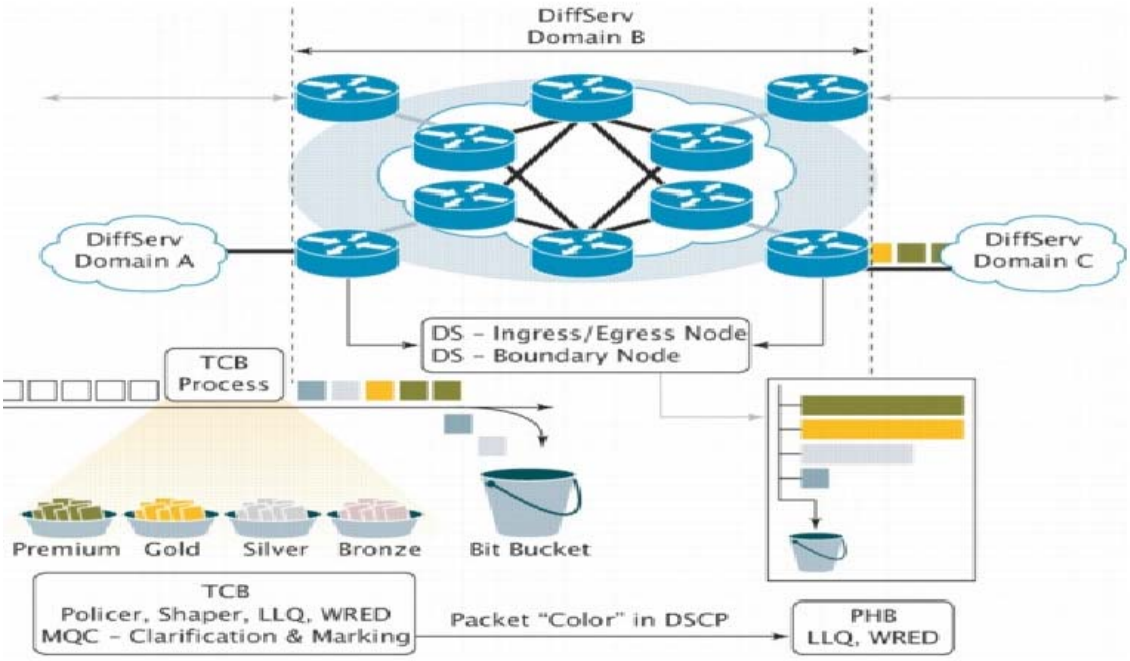



Figure 9 : Le protocole DiffServ [58]

Une fois que les paquets ont été classifiés, les PHB correspondants peuvent être appliqués à ces paquets. PHB fait référence au comportement de planification, de mise en file d'attente, de « policing » ou de mise en forme de paquets. Généralement, la plupart des réseaux utilisent les quatre types de PHB qui sont :

Default PHB : Il est utilisé pour les trafics du Best effort.

Expedited Forwarding (EF) [59] : Il est destiné à fournir un élément de base pour les services à faible retard, à faible jitter et à faibles pertes en garantissant que le trafic est servi à un certain débit configuré sur un intervalle défini de manière appropriée, indépendamment de la charge offerte de trafic non-EF vers cette interface.

Assured Forwarding (AF) [60] : Le groupe AF Per Hop Behavior (PHB) fournit la livraison de paquets IP dans quatre classes AF transmises indépendamment. Au sein de chaque classe AF, un paquet IP peut se voir attribuer l'un des trois niveaux de priorité d'abandon suivants : faible, moyen et élevé.

Class Selector (CS) [61] : Il est utilisé pour être compatible avec le champ de priorité IP dans l'octet TOS.

Bien qu'IntServ et DiffServ soient les idées générales de QoS sur le réseau, tout cela n'est pas encore disponible sur l'infrastructure SDN.

### 2.3.3. Traffic shaping

Le traffic shaping contrôle la fréquence des trames/paquets sortants pour permettre au taux de trafic de correspondre à celui de l'appareil en aval. Lorsque le trafic est transmis d'une liaison à grande vitesse vers une liaison à faible vitesse ou qu'une surcharge de trafic se produit, l'interface

entrante de la liaison à faible vitesse est sujette à de graves pertes de données. Pour éviter ce problème, le traffic shaping doit être configurée sur l'interface sortante de l'appareil se connectant à la liaison à faible vitesse, comme illustré à la Figure 10.

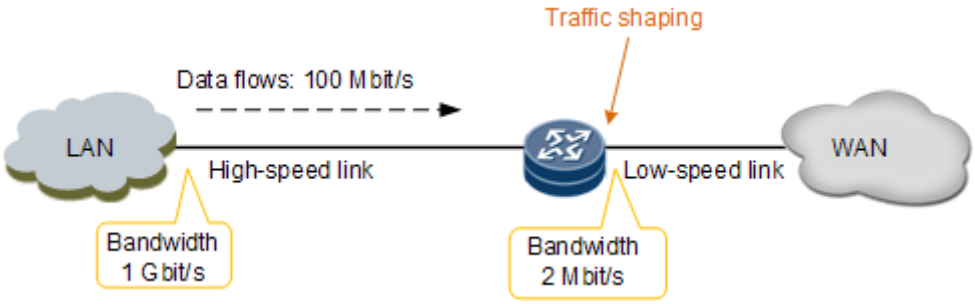



Figure 10 : Transmission de données d'une liaison à grande vitesse vers une liaison à faible vitesse [62]

Comme le montre la Figure 11, la mise en forme du trafic peut être configurée sur l'interface sortante d'un dispositif en amont pour rendre le trafic irrégulier transmis à un débit régulier, évitant ainsi la congestion du trafic sur le dispositif [63].

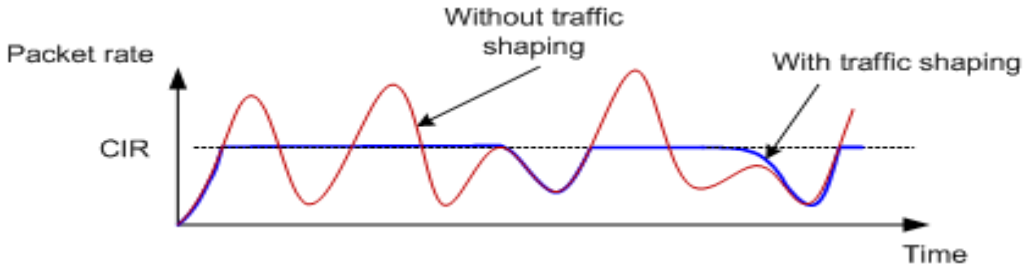



Figure 11 : Les effets du traffic shaping sur les flux [62]

## 2.4. Mécanismes de la QoS

Certains mécanismes de QoS peuvent gérer la qualité du trafic de données et maintenir les exigences de QoS spécifiées dans les SLA (Service-Level Agreement). Les mécanismes de QoS entrent dans des catégories spécifiques selon le rôle qu'ils jouent dans la gestion du réseau.

***Classification et marquage*** **[53]:** Classification et marquage différencient les applications et trient les paquets en différents types de trafic. Le marquage marquera chaque paquet comme un membre d'une classe réseau, ce qui permet aux périphériques du réseau de reconnaître la classe du paquet. La classification et le marquage sont mis en œuvre sur les périphériques réseau tels que les routeurs, les commutateurs et les points d'accès.

***Gestion de la congestion*** **[53]:** Les outils de gestion de la congestion utilisent la classification et le marquage des paquets pour déterminer dans quelle file d'attente placer les paquets. Ils comprennent les files d'attente prioritaires et FIFO.

***Congestion avoidance*** **[53]:** Congestion avoidance surveille la congestion du trafic réseau et rejette les paquets de faible priorité en cas de congestion. Ces outils comprennent la détection précoce aléatoire pondérée et la détection précoce aléatoire.

***Shaping*** **[53]:** Le Shaping manipule le trafic entrant sur le réseau et donnent la priorité aux applications en temps réel par rapport aux applications moins sensibles au temps, comme le courrier électronique et la messagerie. Les outils de mise en forme du trafic comprennent les tampons, le Traffic Shaping et le Frame Relay Traffic Shaping. Tout comme la mise en forme, les outils du « traffic policing » mettent l'accent sur la limitation du trafic excessive et le rejet des paquets.

***Link efficiency*** **[53]:** Link efficiency maximise l'utilisation de la bande passante et réduit les délais d'accès des paquets au réseau. Bien que ce ne soit pas exclusivement pour la QoS, les outils de link efficiency sont utilisés conjointement avec d'autres mécanismes de la QoS. Les outils de link efficiency comprennent la compression de l'en-tête du protocole de transport en temps réel, la compression de l'en-tête du protocole de contrôle de la transmission et la compression des liaisons.

## 2.5. Conclusion

On a présenté dans ce chapitre les recherches en cours dans le domaine de la qualité de service. Il nous a renseignés sur les divers types de techniques qui peuvent être utilisées pour mettre en œuvre des systèmes de QoS dans le SDN. Mais, pour que le SDN remplace l'architecture actuelle du réseau réel, il doit fournir un contrôle très négligé aux administrateurs du réseau pour contrôler la qualité des services.

Pour remplacer l'architecture réseau actuelle, SDN doit apporter des solutions à un certain nombre de problèmes. Cependant, les capacités de qualité de service de toutes les composantes différentes du SDN joueront également un rôle majeur dans l'adoption généralisée du SDN dans le réseau. Dans le chapitre suivant va définir chaque méthodes QoS choisies appropriées pour notre projet.

# Chapitre 3

# DiffServ et Traffic shaping dans SDN

## 3.1. Introduction

Ce chapitre traite l'architecture de diverses méthodes qui seront utilisées dans ce mémoire. En premier lieu, on analyse le support de la QoS « DiffServ » ainsi que ces différentes caractéristiques. On discutera par la suite les fonctionnalités du protocole OpenFlow par rapport à la QoS, en particulier ces files d'attente. Ensuite on traitera le mécanisme du Contrôle de trafic du kernel Linux utilisé pour implémenter le traffic shaping.

Enfin après de multiples recherches effectuées sur les deux méthodes QoS choisies par rapport au SDN, on présentera en donnant nos observations sur les travaux connexes lié à ces derniers.

## 3.2. Traffic mapping :

Il existe deux types du traffic mapping : TOS (Type Of Service) ou DSCP (Differentiated Services Code Point). Une seule méthode peut être utilisée à la fois, TOS étant la méthode par défaut.

### 3.2.1. Valeurs de priorité

Les flux ont un champ appelé le champ Type de service, aussi connu sous le nom d'octet TOS. L'idée originale derrière l'octet TOS était que nous pouvions spécifier une priorité et demander une route pour un débit élevé, un faible délai et un service fiable. L'octet TOS a été défini en 1981, mais la façon dont nous l'utilisons a changé au fil du temps. Cela rend la compréhension difficile car il y a beaucoup de terminologie et une partie de celle-ci n'est plus utilisée de nos jours. Nous allons maintenant discuter sur l'octet TOS, de la priorité et des valeurs DSCP.

### 3.2.2. TOS bits

TOS est un champ faisant partie d’une entête IP qui possède 4 bits comme le montre la Figure 12 :

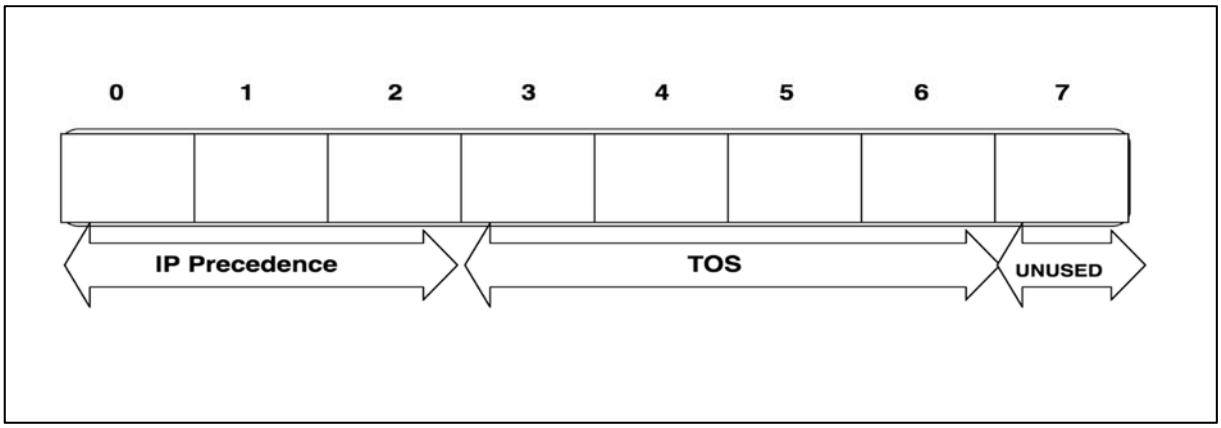



Figure 12 : Champ Type Of Service [64]

- Les 3 premiers bits servent à définir une priorité. Plus la valeur est élevée, plus le flux est important, en cas d'encombrement, le commutateur laisserait tomber en premier les flux de faible priorité.
- Les 4 bits suivants représentent les bits TOS utilisés pour attribuer les valeurs de délai, de débit et de fiabilité ainsi que le cout que nous voulons.
- Le dernier bit n'est pas utilisé.
- Généralement seuls les 3 premiers bits (ceux qui définissent la priorité) sont utilisés, ils ont comme rôle la priorisation du trafic (Voir le Tableau 4). Les commutateurs peuvent choisir d'utiliser ce champ pour accorder un traitement préférentiel à certains types de trafic [65].

| Precedence Value | Meaning |
|---|---|
| 000 (0) | Routine or Best Effort |
| 001 (1) | Priority |
| 010 (2) | Immediate |
| 011 (3) | Flash |
| 100 (4) | Flash Override |
| 101 (5) | Critical |
| 110 (6) | Internetwork Control |
| 111 (7) | Network Control |

Tableau 4 : Valeurs de priorité [64]

Six années sont écoulées depuis les dernières modifications apportées à l'octet TOS. La RFC 2474 est créée et décrit un octet TOS différent, avec un nouveau nom « le champs DS (Differentiated Services) », en plus des changements au niveau des bits.

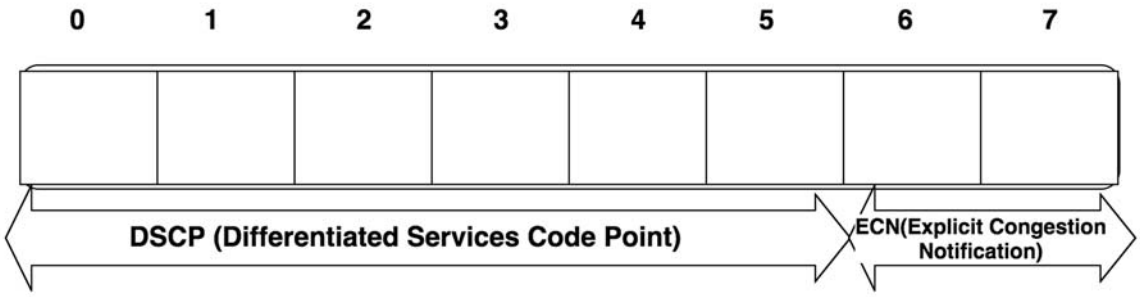



Figure 13 : Champs DSCP et ECN [64]

Les valeurs DSCP sont utilisées pour l'unique raison d'atteindre différents et plusieurs niveaux prioritaires de flux (Voir le Tableau 5). Le champs DS comme illustré sur la Figure 13 se compose de 8 bits dont 6 bits sont réservés au DSCP, leur but est de classifier les flux prioritaires, les 2 bits restants sont l'ECN (Explicit Congestion Notification) [66]. L'ECN a changé comment la congestion réseau est signalée. Il la signale en modifiant la valeur des 2 bits qui lui sont accordé dans le champ DS.

| DSCP | Binary |
|---|---|
| AF 11 | 0 0 1 0 1 0 - Low Drop Precedence/ CS1 |
| AF 12 | 0 0 1 1 0 0 - Medium Drop Precedence/ CS1 |
| AF 13 | 0 0 1 1 1 0 - High Drop Precedence/ CS1 |
| AF 21 | 0 1 0 0 1 0 - Low Drop Precedence / CS2 |
| AF 22 | 0 1 0 1 0 0 - Medium Drop Precedence/CS2 |
| AF 23 | 0 1 0 1 1 0 - High Drop Precedence/CS2 |
| AF 31 | 0 1 1 0 1 0 - Low Drop Precedence/CS3 |
| AF 32 | 0 1 1 1 0 0 - Medium Drop Precedence/CS3 |
| AF 33 | 0 1 1 1 1 0 - High Drop Precedence/CS3 |
| AF 41 | 1 0 0 0 1 0 - Low Drop Precedence/CS4 |
| AF 42 | 1 0 0 1 0 0 - Medium Drop Precedence/CS4 |
| AF 43 | 1 0 0 1 1 0 - High Drop Precedence/CS4 |

Tableau 5 : Valeurs de priorité DSCP [67]

## 3.3. OpenFlow et la QoS

OpenFlow a pris en compte une notion de QoS depuis ses premières versions. Toutefois, le soutient a été limité. Les premières versions d'OpenFlow OF1.0 - OF1.1 supportaient les files d'attente avec des tarifs minimums. Il est encore amélioré dans OF1.2 suite à l'implémentation d'une file d'attente avec un mécanisme de limite de débit maximum. Plus tard, dans OF1.3, une fonctionnalité similaire de limitation de débit à travers les tables de compteurs a été introduite. Le commutateur OpenFlow « OpenvSwitch » a également la possibilité de lire et d'écrire des bits de type de service (ToS) dans une en-tête IP [68].

### 3.3.1. Les files d'attente

Depuis le début d'OpenFlow dans OF1.0 les files d'attente ont supporté l'implémentation des flux à débit limité sortant d'un port de commutateur. Les files d'attente sont conçues pour garantir le débit des flux. Elles peuvent être utilisée pour donner la priorité au trafic "spécial" sur le trafic "ordinaire" (ou pour limiter le trafic "gourmand" à titre de sanction). OpenFlow 1.0 et 1.1 supportent les files d'attente avec des taux minimums garantis, tandis qu'OpenFlow 1.2+ supporte les taux minimum et maximum pour une file d'attente donné.

OpenFlow 1.3 introduit les « Meters » (compteurs) au protocole OpenFlow, qui complètent le Framework de file d'attente déjà en place en permettant la surveillance du débit du trafic avant la sortie. Noter donc que les compteurs et les files d'attente sont complémentaires et ne sont pas des implémentations différentes. On croit souvent à tort que les compteurs remplacent les files d'attente, ce qui n'est pas vrai.

Les files d'attente OpenFlow sont largement supportées. Ils sont pris en charge sur la plupart des implémentations de commutateurs SDN (par exemple : OpenvSwitch (OVS), ofsoftswitch, etc.), aussi de nombreux fournisseurs de commutateurs matériels prennent également en charge les files d'attente, avec bien sûr des limitations.

Les files d'attente, bien que très utiles, sont définies en dehors du protocole OpenFlow. OpenFlow ne fait qu'entourer les mécanismes de files d'attente de commutateurs existants afin d'en informer le contrôleur.

Dans [69], les auteurs utilisent les files d'attente OpenFlow pour fournir des garanties de bande passante. Pour chaque flux une file d'attente est créée dans le commutateur d'entrée et les commutateurs intermédiaires avec un débit minimal et maximal égal à la bande passante garantie. Bien que ce soit correct, le système n'évolue pas, puisque le nombre de files d'attente augmente avec le nombre de flux.

### 3.3.2. Traffic Control

Le Traffic Control (TC) est utilisé pour configurer le contrôle du trafic dans le kernel Linux. Le Traffic Control se compose des éléments suivants [70]:

- **Planification :** En planifiant la transmission des paquets, il est possible d'améliorer l'interactivité pour le trafic qui en a besoin, tout en garantissant la bande passante pour les transferts en masse. Le réordonnancement est également appelé priorisation et n'a lieu qu'au moment de l'évacuation des flux.

- **Policing :** La mise en forme porte sur la transmission du trafic alors que le « policing » concerne le trafic à l'arrivée. Le « policing » se produit donc lors de l'entrée.

- **Dropping :** Le trafic dépassant une bande passante définie peut être immédiatement rejeter, tant à l'entrée qu'à la sortie.

Le traitement du trafic est contrôlé par trois types d'objets : Qdiscs, classes et filtres.

### a. Qdiscs

Qdisc est l'abréviation de « queueing discipline » et il est élémentaire pour comprendre le contrôle du trafic. Chaque fois que le kernel a besoin d'envoyer un flux à une interface, il est demandé au Qdisc d'enfiler à cette interface. Immédiatement après, le kernel essaie d'obtenir autant de flux que possible du Qdisc, pour les donner au pilote de l'adaptateur réseau.

Un Qdisc simple est le « FIFO », qui ne fait aucun traitement du tout et qui est une file d'attente « First In, First Out ».

### b. Classes

Certains Qdisc peuvent contenir des classes, qui contiennent d'autres Qdisc, le trafic peut alors être enfiler dans n'importe quel Qdisc internes, qui sont dans les classes. Quand le kernel essaie de défiler un flux d'un Qdisc de classe, il peut venir de n'importe quelle classe. Un Qdisc peut par exemple donner la priorité à certains types de trafic en essayant de défiler certaines classes avant d'autres.

### c. Filtres

Un filtre est utilisé par un Qdisc de classe pour déterminer dans quelle classe un flux sera enfiler. Tous les filtres attachés à la classe sont appelés, jusqu'à ce que l'un d'eux revienne avec un verdict. Si aucun verdict n'a été rendu, d'autres critères peuvent être pris en charge. Ceci diffère par Qdisc.

Il est important de noter que les filtres résident dans les Qdiscs, ils ne sont pas maîtres de ce qui se passe.

## 3.4. Token Bucket hiérarchique

HTB (Hierarchical Token Bucket) est conçu comme un remplacement plus compréhensible et intuitif pour le Qdisc sous Linux. Qdisc et HTB aident à contrôler l'utilisation de la bande passante

sortante sur une liaison donnée. Les deux permettent d'utiliser un lien physique pour simuler plusieurs liens plus lents et d'envoyer différents types de trafic sur différents liens simulés. Dans les deux cas, il faudra spécifier comment diviser le lien physique en liens simulés et comment décider quel lien simulé utiliser pour un flux donné à envoyer.

HTB utilise les concepts de jetons et de seaux (buckets) ainsi que le système hiérarchique et les filtres basés sur les classes pour permettre un contrôle complexe et granulaire des flux. HTB permet à l'utilisateur de définir les caractéristiques des jetons et des seaux utilisés, aussi d'imbriquer ces seaux d'une manière arbitraire avec le taux et le plafond (ceiling). Le taux définit le débit de données assuré, tandis que le plafond définit le débit de données maximal autorisé pour le flux [71].

Contrairement au Qdisc, HTB forme le trafic basé sur l'algorithme « Token Bucket Filter » qui ne dépend pas des caractéristiques de l'interface et n'a donc pas besoin de connaître la bande passante sous-jacente de l'interface sortante [72].

Les classes HTB disposent d'une multitude de paramètres pour configurer leur fonctionnement :

- **Taux (rate)** [72]**:** Le taux maximum de cette classe et de tous ses enfants est garanti.

- **Prioriser le partage de la bande passante** [73]**:** La priorisation du trafic a deux côtés. Tout d'abord, cela affecte la façon dont la bande passante excédentaire est répartie entre les frères et sœurs. Il y a aussi une deuxième facette du problème. C'est le délai du flux. Il est relativement difficile de le mesurer sur Ethernet qui est trop rapide (le délai est tellement négligeable). Mais il existe une méthode simple, Ou une simple HTB est ajoutée avec un taux de classe limité à moins de 100 kbps et ajouter une seconde HTB (celle que nous mesurons) comme enfant. Ensuite, nous pouvons simuler une liaison plus lente avec des délais plus large.

- **Ceiling** [73]**:** L'argument plafond spécifie la bande passante maximale qu'une classe peut utiliser. Cela limite la bande passante que cette classe peut emprunter. Le plafond par défaut est le même que le taux.

Les files d'attente sont créées à l'aide de la commande « ovs-vsctl ». Cette commande crée une entrée dans OVSDB, puis l'implémente dans le commutateur sous Linux TC (Traffic Control).

## 3.5. Table de flux

Une table de flux est constitué d'entrées de flux [28].

| Champs de correspondance | Priorité | Compteurs | Instructions | Temps morts | Cookie |
|---|---|---|---|---|---|

Tableau 6 : Composants principaux d'une entrée de flux dans une table de flux [28]

Chaque entrée de table de flux (Tableau 6) contient :

- **Champs de correspondance :** Il s'agit du port d'entrée et des en-têtes de flux, et éventuellement des métadonnées spécifiées par une table précédente.
- **Priorité :** Priorité correspondante de l'entrée de flux.
- **Compteurs :** Mis à jour lorsque les paquets sont trouvés.
- **Instructions :** Pour modifier l'ensemble d'actions ou le traitement en pipeline.
- **Temps morts :** Le temps maximum ou le temps d'inactivité avant que le flux ne soit expiré par le commutateur.
- **Cookie :** Valeur de données choisie par le contrôleur. Peut être utilisé par ce dernier pour filtrer les statistiques, la modulation de débit et sa suppression. Non utilisée lors du traitement des flux.

Une entrée de table de flux est identifiée par ses champs de correspondance et sa priorité : les champs de correspondance et la priorité combinés identifient une entrée de flux unique dans la table de flux. L'entrée de flux qui caractérise tous les champs et a une priorité qui égale à 0 est appelée entrée de flux manquée dans la table de flux. Le diagramme de la Figure 14 explique en détails le fonctionnement de la table de flux.

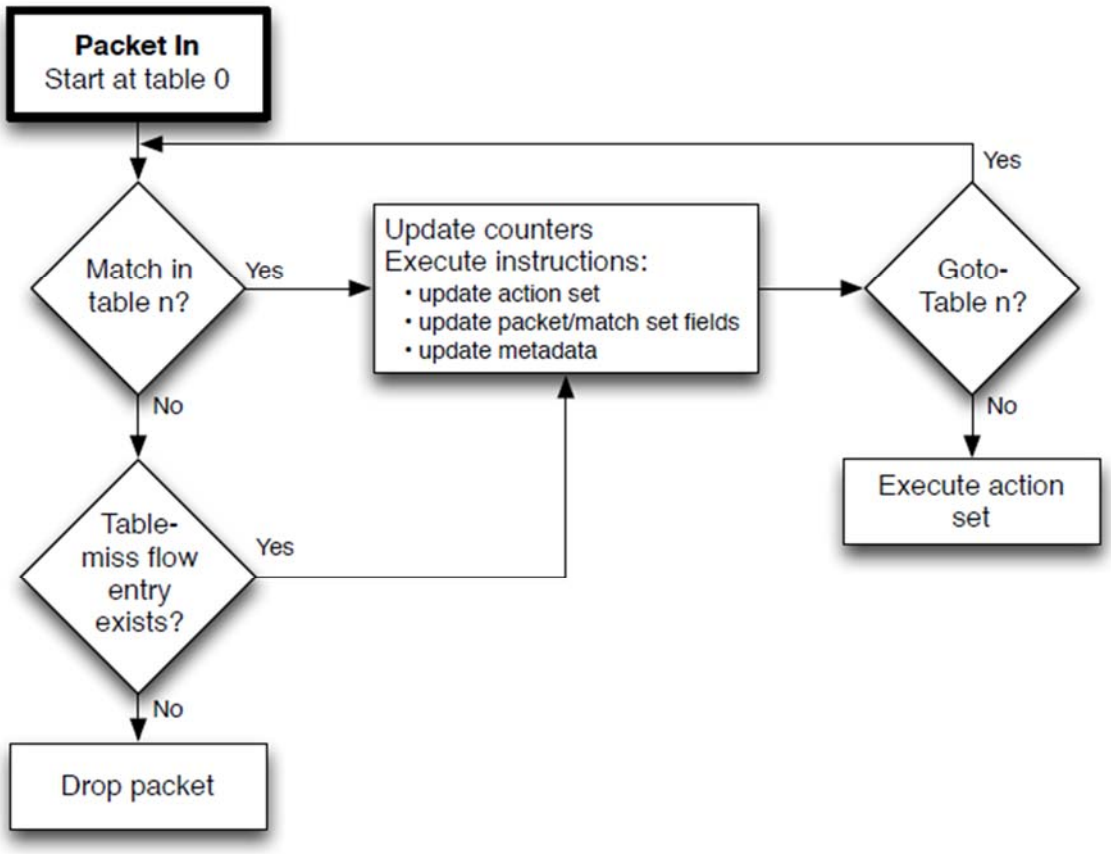



Figure 14 : Organigramme détaillant le flux à travers un commutateur OpenFlow [28]

## 3.6. Table de compteurs

Elle est constituée d'entrées de compteurs. Les compteurs par flux permettent à OpenFlow d'implémenter des diverses opérations simples de QoS, telles que la limitation du débit, et peuvent être combinés avec des files d'attente pour implémenter des Frameworks QoS complexes, tels que DiffServ.

Un compteur mesure la fréquence des paquets qui lui sont assignés et permet de contrôler leurs fréquences. Les compteurs sont fixés directement aux entrées de flux (par opposition aux files d'attente qui sont attachées aux ports). Toute entrée de flux peut spécifier un compteur : qui mesure et contrôle le débit de l'ensemble des entrées de flux auxquelles il est raccordé. Plusieurs compteurs peuvent être utilisés dans la même table en utilisant plusieurs entrées de flux si le commutateur le soutient.

| Identificateur du compteur | Bandes de mesure | Compteurs |
|---|---|---|

Tableau 7 : Composants principaux de la table des compteurs [74]

Chaque entrée de compteur est identifiée par son identificateur (Tableau 7) et contient [74] :

- **Identificateur du compteur :** Un nombre entier de 32 bits identifiant de façon unique le compteur.
- **Bandes de mesure :** Une liste non ordonnée de bandes de mesure, où chaque bande a un débit spécifique et une façon de traiter le flux.
- **Compteurs :** Mis à jour lorsque les flux sont traités par ce dernier.

Il y a deux types de bandes pour définir comment un flux serait traité, il s'agit de drop et DSCP. Les bandes fonctionne sur le trafic qui dépasse le taux défini. La bande de drop rejette les flux qui dépassent le taux spécifié. Il peut être utilisé pour définir une bande de limitation de débit. La bande de DSCP, d'autre part, est utilisée pour augmenter la priorité de drop du champ DSCP dans l'en-tête IP. Il peut être utilisé pour implémenter DiffServ.

## 3.7. Travaux connexes

Le paradigme SDN a émergé en réponse aux limites des architectures réseau traditionnelles. Ses principaux avantages sont : la vue centralisée et globale du réseau, la programmabilité et la séparation du Data plane et du Control plane. Ces caractéristiques ont attiré l'attention des chercheurs afin d'améliorer la QoS des diverses applications réseau d'aujourd'hui.

La qualité de service d'OpenFlow est dite limitée puisqu'elle n'est réalisée qu'avec deux caractéristiques, c'est-à-dire la file d'attente et la table de compteurs. Il n'y a pas de standards définis comme IntServ et DiffServ dans OpenFlow. D'une part, elle peut être considérée comme une flexibilité pour les administrateurs réseau pour implémenter leur propre algorithme de QoS, mais d'autre part, elle ajoute de la difficulté à implémenter la QoS dans OpenFlow.

### 3.7.1. DiffServ dans SDN

Dans HiQoS : Une solution de QoS multipath basée sur SDN [75], la QoS est obtenue grâce à une garantie de bande passante spécifique à la classe. HiQoS divise le trafic en trois classes différentes. Streaming vidéo, multimédia interactif et streaming best effort. Ces trois classes sont transférées dans trois files d'attente différentes avec des taux préconfigurés, ce qui signifie que chaque file d'attente a une bande passante minimale et maximale fixe des classes. De cette façon, la qualité de service est atteinte entre les trois classes, mais dans le cas d'une trop grande bande passante d'une seule classe, il y aura congestion pour cette file d'attente. Ainsi, l'évolutivité est limitée car trop de trafic à l'intérieur d'une file d'attente causera de la congestion, et le nombre de files d'attente doit être préconfiguré.

Dans un autre article [76] la garantie de bande passante par flux est utilisée. Ici, le type de trafic dépend des bits DSCP/TOS. Le trafic est divisé en deux files d'attente distinctes. Une file d'attente est pour le trafic avec les bits DSCP/TOS activés, et l'autre file d'attente est pour le trafic avec les bits DSCP/TOS désactivés, en d'autres termes pour le trafic normal. Bien qu'il puisse seulement diviser en deux classes avec les bits DSCP/TOS, trois différentes files d'attente existent. La troisième file d'attente est destinée au trafic de contrôle.

Via l'API Northbound d'un contrôleur SDN, il est possible de définir dynamiquement les SLAs et de les appliquer sur le plan de transfert sous-jacent via l'API Southbound. L'une des solutions proposées pour SDN est PolicyCop [77], basée sur le contrôleur Floodlight. Elle diffère des autres approches DiffServ existantes car elle n'est pas limitée par les classes de trafic statiques. Le Framework est facilement extensible, car il a une architecture en couches. Chaque couche communique avec les autres via des API JSON RESTful, d'où la possibilité d'ajouter facilement de nouvelles couches ou changer les anciennes. Il permet en outre de créer dynamiquement de nouvelles classes de flux, contrairement aux classes statiques dans DiffServ [77]. La solution est flexible, car elle ne nécessite pas de ressources spéciales, à l'exception de commutateurs avec OpenFlow.

### 3.7.2. Traffic shaping dans le SDN

Dans une étude faite sur le traffic shaping [78], l'auteur discute des techniques et des moyens pour mettre en œuvre la QoS dans OVS (OpenvSwitch) par le traffic shaping. OVS démontre une prise en charge substantielle de la QoS sous la forme de qdisc (Ingress Classless Queuing Disciplines), HFSC (Hierarchical fair-service curve) et HTB (Hierarchy Token Bucket). Il prend en charge l'ajustement dynamique de la bande passante. Un Framework est fourni pour les tests et l'analyse des réseaux à effectuer afin de démontrer la QoS.

Un exemple de Framework visant le traffic shaping est FlowQoS [79]. Ce prototype cible les réseaux à petite échelle et utilise le SDN pour reconfigurer les routeurs domestiques selon un ensemble de polices de mise en forme du débit définies par l'utilisateur. La classification des différents flux se fait par deux modules distincts. Le premier module effectue la classification initiale du trafic Web (HTTP et HTTPS), déterminée par les numéros de port respectifs (80 et 443). Ensuite, une classification plus poussée de ce trafic Web est effectuée sur la base des réponses DNS. Le second module traite de la classification des autres flux (non HTTP/HTTPS). Une fois que les flux ont été classifiés avec succès, le traffic shaping a lieu. Pour pallier l'absence d'une

implémentation OpenFlow de mise en forme du débit par débit, le Framework introduit deux commutateurs "virtuels" à l'intérieur du routeur domestique. Les différentes connexions entre ces deux commutateurs internes sont ensuite configurées via l'utilitaire Tc (Traffic Control) de Linux et affectées à différents taux spécifiés par le contrôleur (prédéfinis par l'utilisateur).

Le Tableau 8 résume les travaux connexes ainsi que quelques observations.

| Titre | Auteur | Solution | Catégorie | Observations |
|---|---|---|---|---|
| HiQoS : An SDN-based multipath QoS solution | Jinyao Yan, Hailong Zhang, Qianjun Shuai, Bo Liu, Xiao Guo | HiQoS | DiffServ | • Contrôleur : Floodlight.<br>• La bande passante des liens est trop basse elle est définie à 4mb/s pour un totale de deux files d'attentes. |
| Implementing Quality of Service for the Software Defined Networking Enabled Future Internet | Sachin Sharma, Dimitri Staessens, Didier Colle, David Palma | QoS Framework pour un système autonome dans l'internet | DiffServ | • Contrôleur : Floodlight.<br>• Pour un totale de 16 serveurs et clients et une limite de bande passante de 100mb/s le trafic généré pour chaque client est à 0.56mb/s pour Best effort et 0.24mb/s représentant le trafic prioritaire. |
| PolicyCop : An Autonomic QoS Policy Enforcement Framework for Software Defined Networks | Md.Faizul Bari, Shihabur Rahman, Reaz Ahmed, Raouf Boutaba | PolicyCop | DiffServ | • Contrôleur : Floodlight.<br>• Topologie trop petite contenant 4 OpenVswitch ainsi que 4 clients avec une bande passante de 150mb/s entre chaque lien. |
| Traffic shaping with OVS and SDN | Ramiro Voicu Caltech | OVS Limite du taux de flux entrant | Traffic shaping | • Contrôleur : Onos.<br>• L'auteur mentionne que les CPU des OVS n'ont subie aucune surcharge cela doit être du fait que la congestion n'étais pas présente vue que les liens sont limités a 10gb/s. |
| FlowQoS : QoS for the rest of us | M. Said Seddiki, Muhammad Shahbaz, Sarthak Grover | FlowQoS | Traffic shaping | • Contrôleur : POX (dans une Rasberypi).<br>• Topologie trop petite contenant 2 OpenVswitch ainsi que 2 clients avec une bande passante de 150mb/s entre chaque lien. |

Tableau 8 : Tableau comparatif entre les solutions citées

D'après toutes nos recherches effectuées on a remarqué qu'aucun chercheur n'a comparer ou mentionné de comparaison entre les deux méthodes QoS DiffServ et traffic shaping dans le SDN.

## 2.8. Conclusion

Ce présent chapitre décrit dans un premier temps les solutions choisies pour implémenter la QoS dans le SDN. Par la suite on a introduit l'accouplement des files d'attente et du protocole OpenFlow avec la QoS, avec une définition sur leurs éléments. Ensuite on a analysé la technologie utilisée pour implémenter le traffic shaping. Enfin on a présenté travaux connexes sur le DiffServ et le traffic shaping dans le SDN, ils nous ont renseignés sur les divers types de techniques qui peuvent être utilisées pour mettre en œuvre les systèmes de QoS dans le SDN, sur les divers défis auxquels fait face la communauté du SDN et nous a suggéré des solutions plausibles à certains des problèmes. Bien que la plupart de ces recherches aient fourni l'une ou l'autre forme pour atteindre la QoS, elles ont également démontré l'absence de la QoS dans l'infrastructure actuelle du SDN. Pour que le SDN remplace l'architecture actuelle du réseau réel, il doit fournir un contrôle très détailler aux administrateurs du réseau pour contrôler la QoS.

Dans le chapitre suivant on va explorer les outils SDN utilisés pour l'implémentation et l'étude de performance de la QoS.

# Chapitre 4

# Les outils SDN pour l'implémentation de la QoS

## 4.1. Introduction

Les SDN ont pour but pratique de rendre les réseaux programmables par le biais d'un contrôleur centralisé. Un contrôleur central omniscient permet aux ingénieurs de réseau de mettre en œuvre des politiques de transfert uniques et flexibles dont les seules limitations sont liées à la capacité du logiciel faisant fonctionner le contrôleur.

Dans ce chapitre en premier lieu on va effectuer un choix du contrôleur SDN. Ensuite une exploration du contrôleur SDN sera effectuée avec ses différentes versions en se basant sur la version utilisée dans le projet. Par la suite on va expliquer le choix du OpenVswitch comme commutateur SDN, et nous parlerons de l'émulateur réseau Mininet et du workload afin d'examiner l'exactitude de mesure par le biais du iPerf.

## 4.2. Le contrôleur SDN

Avec tous les outils nécessaires pour un réseau, un contrôleur SDN doit être sélectionné pour le gérer. De nombreux facteurs doivent être pris en compte lors du choix d'un contrôleur.

Pour atteindre la QoS le DS sera utilisé, à son tour les compteurs mettront en œuvre le marquage DSCP. Pour que les compteurs soient activés, les commutateurs doivent prendre en charge OF 1.3. Ainsi, le contrôleur doit également prendre en charge OF 1.3, sinon ils seront incompatibles entre eux.

D'après l'étude réalisé, on a constaté que l'OpenDaylight est le contrôleur le plus complet en termes de fonctionnalités. La prise en charge d'un large éventail d'applications avec un bon écosystème lui confère une réelle chance de devenir le contrôleur adéquat à l'utilisation future. Aussi selon Khondoker, Zaalouk *et. al.* [80], OpenDaylight est l'un des 3 top meilleurs contrôleurs pour le support OF. De plus selon Mamadou T.BAH ; Azzouni *et. al.* [81], les mesures du temps de découverte de topologie ont été effectuées à la fois en mode unique et en mode cluster et montrent qu'OpenDaylight obtient de meilleures performances. Et pour finir selon Liehuang Zhu, Md Monjurul Karim *et. al.* [82], OpenDaylight fait partie des contrôleurs distribués qui fonctionnent mieux que les contrôleurs centralisés.

## 4.3. Contrôleur Opendaylight

Annoncé en 2014 et hébergé par la Linux Fondation, le projet OpenDaylight est un projet collaboratif open source qui vise à accélérer l'adoption de SDN.

Il s'agit d'une plate-forme pour le Software Defined Network (SDN). Son rôle est de fournir le contrôle centralisé et programmable ainsi que les protocoles open-source de surveillance des périphériques réseau.

En raison de la nature open-source du contrôleur, il permet aux utilisateurs de minimiser la complexité opérationnelle et donc de prolonger la durée de vie de leur infrastructure existante [83]. Comme beaucoup d'autres contrôleurs SDN, tels que Ryu, Pox…etc. OpenDaylight prend également en charge OpenFlow et offre des solutions réseau prêtes à installer dans le Framework de sa plate-forme [28].

La communauté OpenDaylight est en train de développer une architecture SDN qui prend en charge un large éventail de protocoles et peut évoluer rapidement dans la direction du SDN, sans se baser sur les objectifs d'un seul fournisseur.

Les projets ont été réalisés par Cisco, ConteXtream, Ericsson, IBM, Industrial Technology Research Institute (ITRI), NEC, Pantheon, Plexxi, Radware et les développeurs Brent Salisbury et Evan Zeller de l'Université du Kentucky.

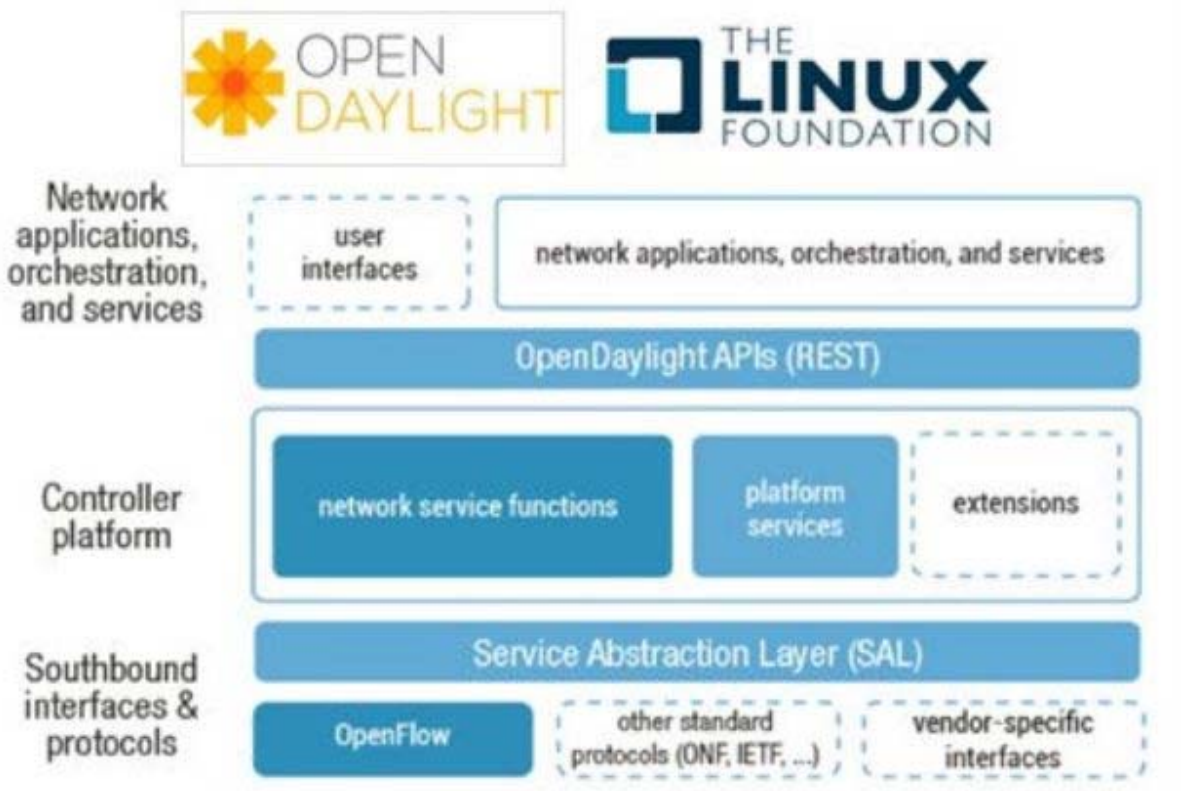



Figure 15 : Architecture OpenDaylight [84]

Comme le montre la Figure 15, l'architecture d'OpenDaylight est une architecture multicouche :

- La couche principale est une plate-forme de contrôleur ou il y réside, et agit comme le cerveau du réseau, il gère le flux de trafic des commutateurs en utilisant des tables de flux [85].

- La couche SAL (Service Abstraction Layer) est située au cœur de la conception modulaire du contrôleur comme le montre la Figure 15 de l'architecture OpenDaylight, elle peut prendre en charge plusieurs protocoles Southbound et fournit des services pour les modules et les applications.

- Il y a quelques modules dynamiquement ajouter qui sont responsables de l'exécution des tâches réseau et qui sont contenus dans le contrôleur lui-même. Il est également possible d'y insérer d'autres services et extensions pour améliorer la fonctionnalité SDN. Tous ces modules sont reliés au SAL.

- Pour fournir les services demandés indépendamment du protocole sous-jacent utilisé et des périphériques réseau, la couche d'infrastructure est exposée par le SAL aux applications situées au nord de celui-ci.

Lorsqu'il s'agit de programmer des applications pour Opendaylight, il existe deux approches différentes du SAL qui peuvent être prises en compte [44] :

1. **L'API-Driven SAL (AD-SAL).**
2. **Le Model-Driven SAL (MD-SAL).**

### 4.3.1. AD-SAL

L'approche AD-SAL [85] présente les principales caractéristiques suivantes :

- Il peut être utilisé avec les pluggins southbound et northbound.
- Les applications sont programmées dans le contrôleur sous forme de bundles OSGi (Open Services Gateway initiative).
- La programmation du flux est réactive, en recevant les événements du réseau.

### 4.3.2. MD-SAL

Cette approche présente les caractéristiques suivantes (Voir la Figure 16) :

- Il dispose d'une API REST commune à tous les modules.
- C'est un modèle agnostique. Il prend en charge tous les modèles d'appareils ou de services.
- Les applications sont programmées en dehors du contrôleur.

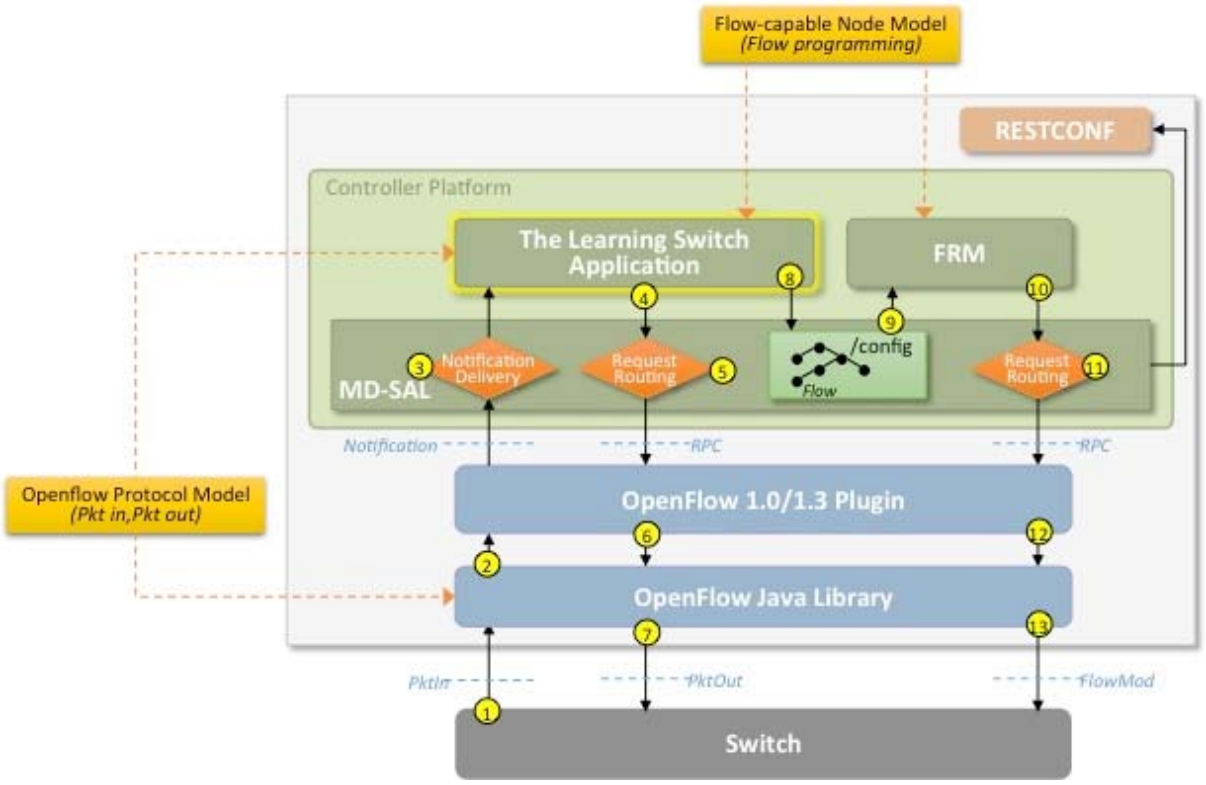



Figure 16 : Modèle de MD-SAL [86]

L'API REST est utilisée par des applications fonctionnant en dehors du contrôleur lui-même, et même dans différentes machines.

Depuis la création du projet, il y a eu ces versions majeures du contrôleur OpenDaylight [87] :

- Hydrogène (février 2014)
- Hélium (septembre 2014)
- Lithium (août 2015)
- Béryllium (mars 2016)
- Bore (décembre 2016)
- Carbone (version actuelle)

Dans notre projet on a eu recours à la version Béryllium.

### 4.3.3. Béryllium

Béryllium (Be) est la quatrième version d'OpenDaylight (ODL) qui dirige la plate-forme open source pour les réseaux programmables et définis par logiciel. ODL est la plate-forme SDN de l'industrie, qui prend en charge un large éventail de cas d'utilisation et fournit ainsi la base des réseaux de l'avenir. Pour résoudre de nombreux défis réseau clés liés à l'optimisation des ressources réseau, au Cloud et au NFV, à la recherche…etc. toutes les entreprises utilisent OpenDaylight [85].

Pour renforcer l'architecture d'ODL, il utilise le modèle basé sur MD-SAL et permet d'intégrer facilement de nouvelles applications ainsi que des protocoles [85]. L'architecture BE est illustrée à la Figure 17.

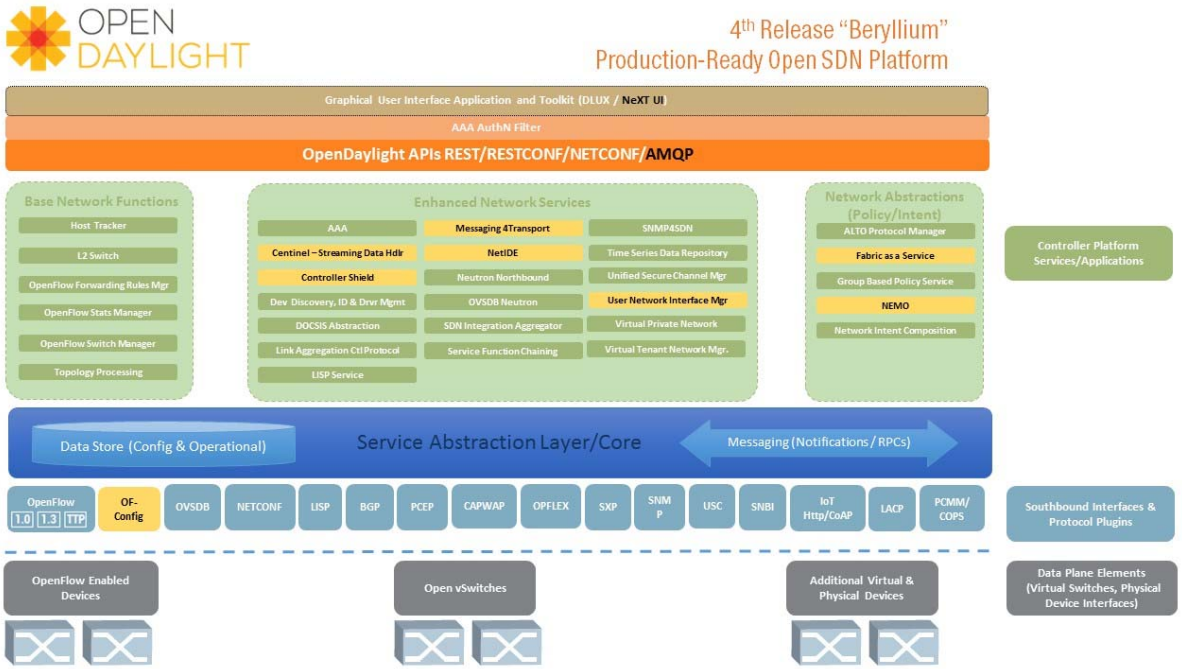



Figure 17 : Béryllium Architecture [88]

## 4.4. OpenvSwitch

Bien que la spécification OpenFlow ait progressé vers la version 1.5.1 en août 2017, les commutateurs OpenFlow les plus populaires ne prennent pas encore en charge toutes les dernières fonctionnalités définies dans la spécification. Ceci est dû au fait que la spécification OpenFlow dresse une liste des fonctionnalités qui sont obligatoires pour qu'un commutateur à implémenter soit appelé un commutateur compatible OF. Cependant, la spécification laisse certaines caractéristiques en option. Ces caractéristiques sont laissées à la décision du fabricant du commutateur pour les implantées. Ceci a conduit à un état de l'art peu clair dans le domaine de la prise en charge de la QoS dans les commutateurs OpenFlow.

La plupart des commutateurs matériels produits commercialement et prêts pour la production prennent en charge la majorité des fonctions de QoS. C'est compréhensible, car on s'attend à ce qu'ils soient déployés dans des réseaux du monde réel qui exigent une qualité de service optimale. Les commutateurs logiciels, d'autre part, sont pour la plupart des implémentations open-source et sont en retard dans la mise en œuvre de toutes ces fonctionnalités. L'équipe de contrôleurs Ryu SDN teste régulièrement la compatibilité OpenFlow de divers commutateurs populaires et délivre une certification. La liste peut être consultée à l'adresse [89]. Nous avons choisi d'utiliser l'OpenvSwitch pour notre projet étant l'un des meilleurs commutateur SDN du moment.

## 4.5. Mininet

Mininet est un émulateur de réseau open-source avec la particularité d'inclure des contrôleurs SDN avec support OF [19], Mininet est donc un outil pour émuler des réseaux SDN. Il fonctionne sur le noyau de Linux et utilise Python pour son API. Il en résulte un outil qui peut être utilisé pour émuler une configuration réseau complète [90].

Bien que Mininet permet la création d'un réseau personnalisé de choix, il impose certaines limites par rapport à un réseau physique réel. Mininet n'est pas adapté pour des expériences allant jusqu'à 10Gbps. Dans de tels cas, les réseaux physiques offrent généralement une meilleure expérience.

Mininet offre l'avantage de la personnalisation, et la création des topologies basée sur le choix de l'utilisateur, au lieu d'être limité par exemple par le paramètre du nombre de commutateurs achetés ou espace physique [91].

Mininet a été testé sous différents scénarios et s'est avéré être un émulateur très performant [92]. Il utilise les techniques de virtualisation réseau de Linux comme l'espace de noms réseau pour émuler plusieurs hôtes et peut créer des réseaux complexes dans une seule machine. Il est compatible avec la plupart des commutateurs logiciels courants.

## 4.6. Workload

Pour ce projet, l'exactitude de la mesure est un objectif obligatoire. Le besoin est de généré un trafic de flux marqué d'une valeur DSCP. Par conséquent, les outils suivants ont été sélectionnés pour la mesure de la bande passante et le jitter, ainsi que l'analyse du trafic sous différents paramètres de QoS : iPerf3 et JPerf [93].

### 4.6.1. iPerf

iPerf est un outil de mesure en direct sur les réseaux, dont l'objectif principal est de déterminer le débit maximal réalisable sur le réseau. Il peut être utilisé pour d'autres mesures comme le jitter et les pertes de trames/paquets. Il est personnalisable pour générer des paquets UDP et TCP de MTU différents avec un débit et un intervalle de transmission spécifiés.

Cet outil acquiert ses statistiques en fonction du nombre de paquets transmis pendant le temps et l'intervalle de transmission. Il comprend deux éléments, un client et un serveur. Le client génère le trafic vers le serveur selon les spécifications du flux souhaitées, et détermine une bande passante moyenne transmise, tandis que le serveur reçoit le trafic généré, effectue les statistiques et renvoie les résultats au client. iPerf fonctionne au niveau de l'application, ce qui signifie que le débit mesuré n'est pas exclusif de la performance des liens, mais aussi du traitement des paquets via le modèle TCP/IP. De plus, iPerf est exécuté au niveau de l'espace utilisateur du système d'exploitation Linux, travaillant au sommet de l'architecture système et utilisant les appels système pour utiliser les ressources.

### 4.6.2. JPerf

JPerf est une interface graphique pour le populaire outil de test réseau iPerf. En utilisant JPerf, on peut rapidement tester une connexion WAN ou LAN pour déterminer le débit maximal du réseau. Les résultats du test sont automatiquement représentés graphiquement et présentés dans un format facile à lire. JPerf peut également être utilisé pour détecter la perte de trames/paquets, le délai, le jitter et d'autres problèmes réseau courants.

## 4.7. Conclusion

Dans ce chapitre on a analysé en détaille l'architecture du contrôleur OpenDaylight en prenant parti sur l'une de ces versions. Ensuite on a opté pour l'OpenvSwitch comme commutateur SDN qui est le plus aptes pour notre projet. Enfin une présentation est établie de l'émulateur SDN ainsi que le workload choisi.

OpenDaylight est l'un des principaux contrôleurs prêts à la production pour SDN. Ce sera une étude intéressante pour explorer les capacités de l'OpenDaylight en matière de QoS et pour comprendre dans quelle mesure il est prêt à remplacer l'infrastructure réseau vieille de plusieurs années et éprouvée. Le prochain chapitre sera consacré à l'implémentation de nos techniques QoS en utilisant le contrôleur OpenDaylight.

# Chapitre 5

# Implémentation et étude des performances de la QoS dans le SDN

## 5.1. Introduction

Les chapitres précédents ont établi le contexte du travail présenté dans ce projet, qui consiste à tester l'OpenDaylight avec différents composants de l'architecture SDN en vue d'éventuelles implémentations et comparaisons QoS. Pour tester les capacités de QoS de l'OpenDaylight, on a réalisé deux expériences différentes où nous avons implémenter les fonctionnalités OpenFlow avec la QoS.

Les deux aspects fondamentaux de modèles QoS, le processus du traffic shaping et le Differentiated Services (DS) sont simulés dans ce chapitre. Tout d'abord, on a recours à une expérience suivie de tests pour voir comment le contrôle de bande passante par le traffic shaping réagit et comment le DS œuvre pour garantir la QoS souhaitée. Enfin après chaque discussion sur les tests nous donnerons notre point de vue sur la meilleur méthode QoS a implémenté sur le réseau réel de l'organisme d'accueil.

## 5.2. Présentation de l'architecture traditionnelle de l'organisme d'accueil

Comme on peut le constater d'après la Figure 18, l'architecture traditionnelle de notre LAN est composée de deux couches essentielles core/accès aussi appelé « Collapsed Distribution and Core layer ».

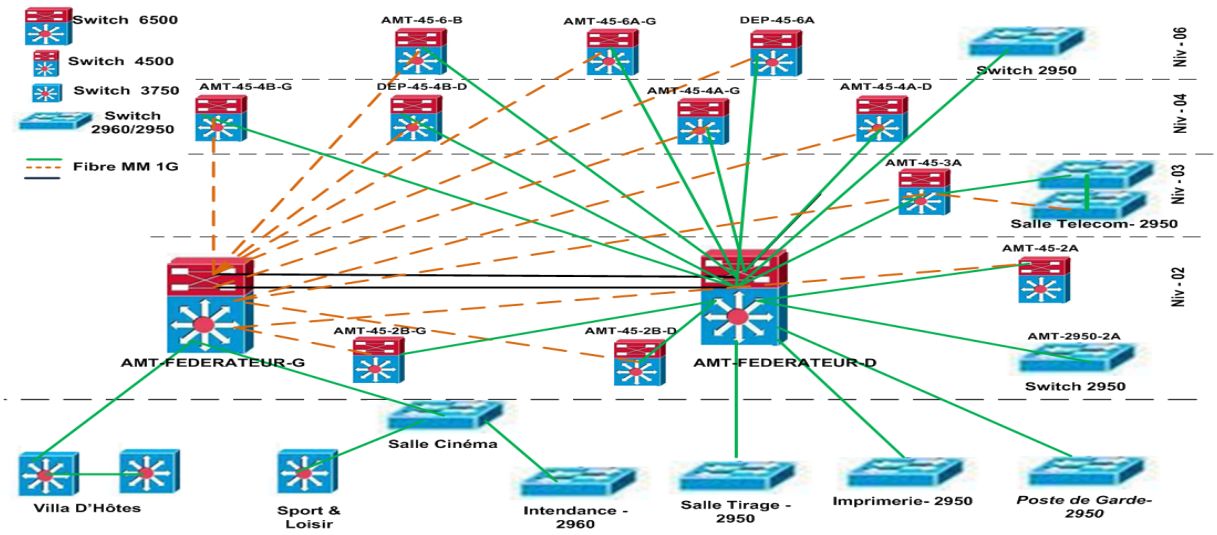



Figure 18 : Architecture traditionnelle du LAN de l'organisme d'accueil

### 5.2.1. Couche Core

La couche core se situe dans l'étage 2 du bâtiment de l'organisme d'accueil, elle se compose de 2 commutateurs fédérateur de couche 3 « Cisco Catalyst 6500 Series » connectés entre eux par 2 câbles en fibre configurés en Etherchannel, les 2 commutateurs core sont connectés avec le reste des autres commutateurs de couche accès en fibre optique. Le protocole HSRP (Hot Standby Router Protocol) a lui aussi été mise en œuvre pour assurer la redondance entre les 2 commutateurs afin qu'il y ait toujours un chemin accessible pour sortir du LAN.

### 5.2.2. Couche Accès

La couche accès se compose de 9 commutateurs de couche 2 « Cisco Catalyst 2950 Series » et 14 commutateurs de couche 3 « Cisco Catalyst 4500 Series ». La couche accès est distribués en 5 niveaux, sur chaque commutateur le protocole STP (Spanning Tree Protocol) a été activé pour ne pas avoir de boucles entre les commutateurs de couche accès et core. Les câbles en vert signifient que la ligne est active et pour orange sa signifie que la ligne est bloquée par le protocole STP.

## 5.3. Conception de l'architecture SD-LAN

La Figure 19 représente la maquette du LAN de l'organisme d'accueil (au niveau DP) sur laquelle on effectuera l'implémentation et les tests lié au traffic shaping et le DSCP, le SD-LAN est une réplique du LAN traditionnel de l'organisme d'accueil, ou on retrouve les deux couches essentielles core/accès, les 2 commutateurs traditionnel appartenant au core n'ont pas été remplacer due à leur grande bande passante pouvant aller jusqu'à 1Gb/s.

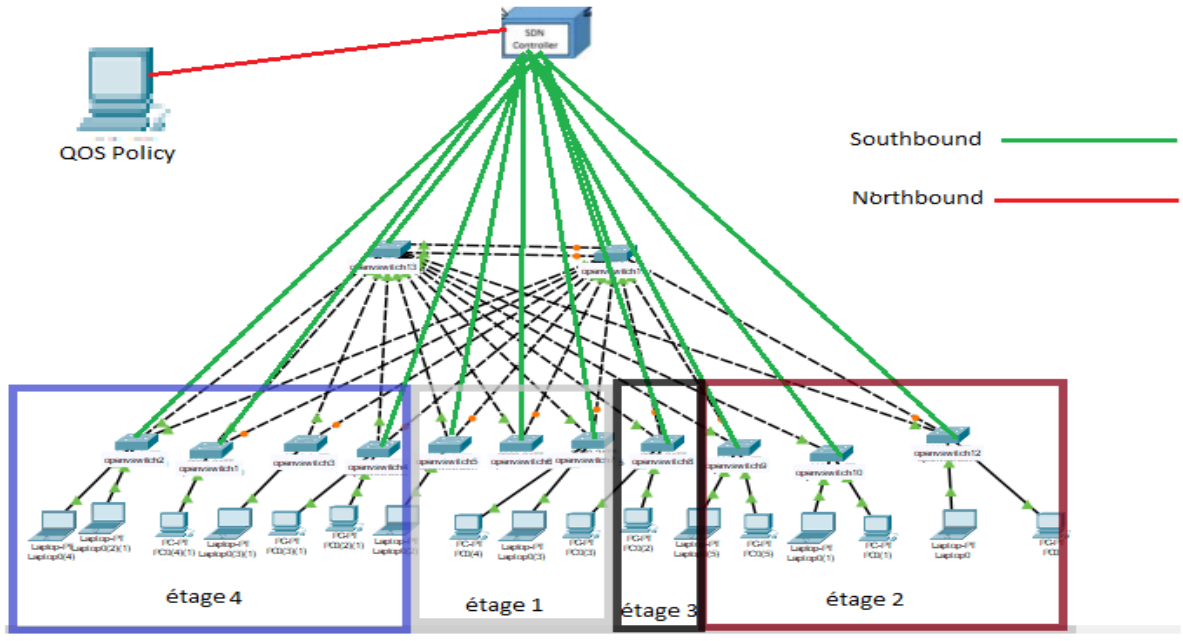



Figure 19 : L'architecture SDN du LAN de l'organisme d'accueil

Le reste des commutateurs de couche 3 se trouvant dans la partie accès ont été remplacés par les commutateurs SDN « OpenvSwitch » ou notre QoS sera appliquées, ils sont répartis en 4 étages comme suit :

- **Etage 1 :** cet étage regroupe 3 OpenvSwitch, 2 d'entre eux remplacent les 2 commutateurs appartenant aux villas D'hôtes tandis que le 3ème remplace celui de la Salle de cinéma.

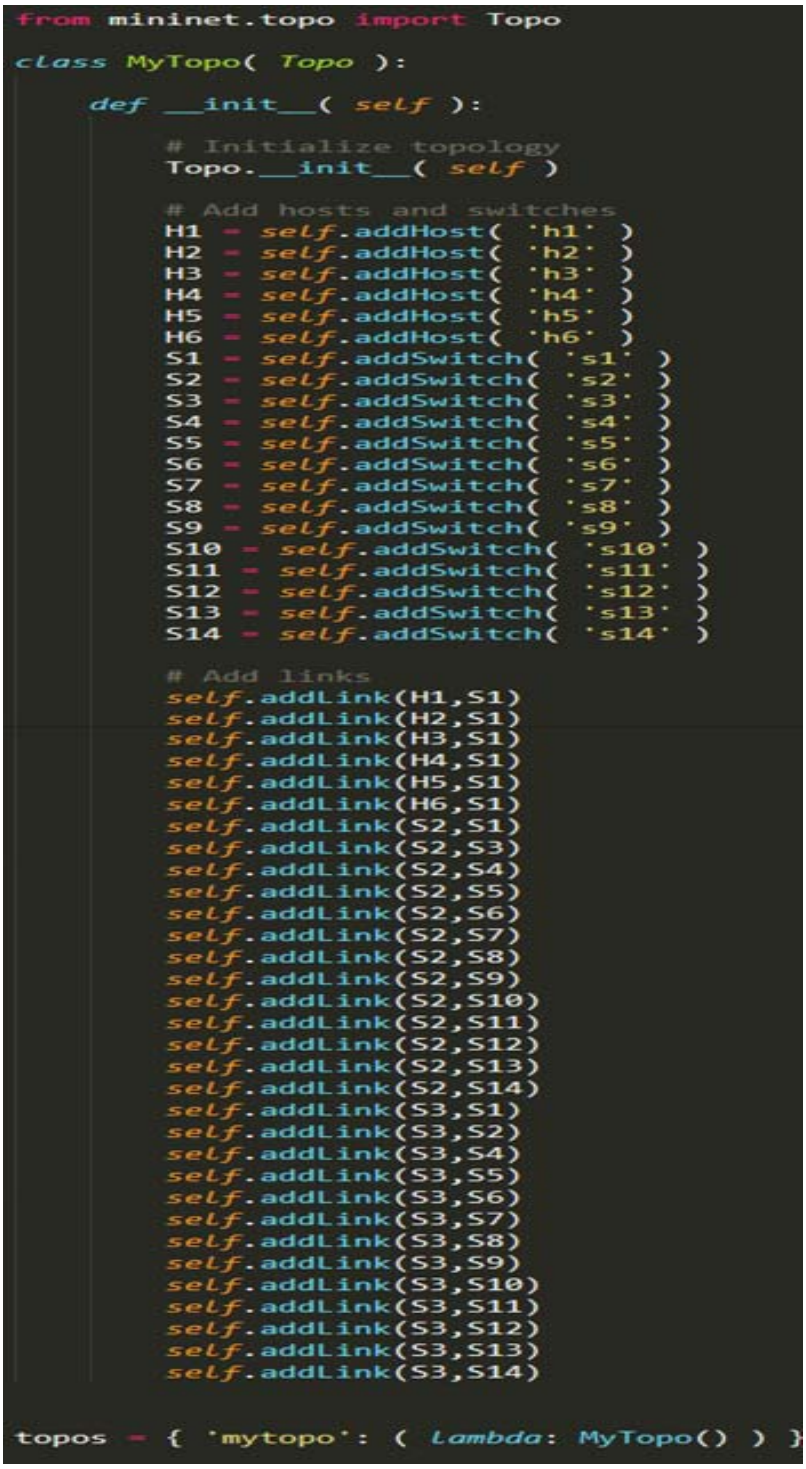
- **Etage 2 :** cet étage regroupe 3 OpenvSwitch, 2 d'entre eux remplacent les commutateurs ATM-45-2B-D et ATM-45-2B-G tandis que le 3ème remplace celui de l'AMT-45-2A.

- **Etage 3 :** cet étage contient qu'un seul OpenvSwitch référent au commutateur AMT-45-3A.

- **Etage 4 :** cet étage regroupe 4 OpenvSwitch, 2 d'entre eux remplacent les 2 commutateurs AMT-45-4A-G et AMT-45-4A-D, ainsi que 2 autres remplaçant les 2 commutateurs AMT-45-4B-G et AMT-45-4B-D.

Les liens sortants des nœuds OpenvSwitch détiennent une bande passante de 100Mb/s. Chaque OpenvSwitch est connecté avec le contrôleur OpenDaylight via leur interface de management (eth-0) dédié uniquement pour les communications Southbound avec le contrôleur. OpenDaylight est connecté avec une machine qui surveille le réseau et injecte les règles QoS en utilisant le NBI.

La topologie a été implémenter sous l'émulateur Mininet en utilisant le scripte en langage Python qui suit :



```
from mininet.topo import Topo

class MyTopo( Topo ):

    def __init__( self ):

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        H1 = self.addHost( 'h1' )
        H2 = self.addHost( 'h2' )
        H3 = self.addHost( 'h3' )
        H4 = self.addHost( 'h4' )
        H5 = self.addHost( 'h5' )
        H6 = self.addHost( 'h6' )
        S1 = self.addSwitch( 's1' )
        S2 = self.addSwitch( 's2' )
        S3 = self.addSwitch( 's3' )
        S4 = self.addSwitch( 's4' )
        S5 = self.addSwitch( 's5' )
        S6 = self.addSwitch( 's6' )
        S7 = self.addSwitch( 's7' )
        S8 = self.addSwitch( 's8' )
        S9 = self.addSwitch( 's9' )
        S10 = self.addSwitch( 's10' )
        S11 = self.addSwitch( 's11' )
        S12 = self.addSwitch( 's12' )
        S13 = self.addSwitch( 's13' )
        S14 = self.addSwitch( 's14' )

        # Add links
        self.addLink(H1,S1)
        self.addLink(H2,S1)
        self.addLink(H3,S1)
        self.addLink(H4,S1)
        self.addLink(H5,S1)
        self.addLink(H6,S1)
        self.addLink(S2,S1)
        self.addLink(S2,S3)
        self.addLink(S2,S4)
        self.addLink(S2,S5)
        self.addLink(S2,S6)
        self.addLink(S2,S7)
        self.addLink(S2,S8)
        self.addLink(S2,S9)
        self.addLink(S2,S10)
        self.addLink(S2,S11)
        self.addLink(S2,S12)
        self.addLink(S2,S13)
        self.addLink(S2,S14)
        self.addLink(S3,S1)
        self.addLink(S3,S2)
        self.addLink(S3,S4)
        self.addLink(S3,S5)
        self.addLink(S3,S6)
        self.addLink(S3,S7)
        self.addLink(S3,S8)
        self.addLink(S3,S9)
        self.addLink(S3,S10)
        self.addLink(S3,S11)
        self.addLink(S3,S12)
        self.addLink(S3,S13)
        self.addLink(S3,S14)


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

Figure 20 : Code source de la topologie SD-LAN en Python

La commande a exécuté sur Mininet pour créer la topologie à partir du scripte Python :

- « sudo mn --custom topo-2sw-2host.py --topo mytopo –controller = remote,ip=192.168.209.159,port=6633 --switch ovsk, --link tc,bw=100 »

Comme on peut la commande se sert de notre code scripte Python « topo-2sw-2host.py » pour implémenter notre topologie en utilisant notre classe mytopo comme on l'avais constaté sur la Figure 20, tous les OpenvSwitch sont connectés avec notre contrôleur OpenDaylight qui a pour

adresse ip facultatif « 192.168.209.159 » et numéro de port « 6633 ». Enfin on remarque la bande passante entre tous les liens est défini à 100mb/s « --link tc,bw=100 ».

## 5.4. Expérimentations

La section des expériences est organisée de la manière suivante. Chaque première section présente l'énoncé de l'expérience, en précisant ce que nous essayons d'accomplir. Ensuite, la mise en œuvre de cette expérience est expliquée en détail. Les résultats de l'expérience sont ensuite présentés. Chaque section se termine par une note sur les observations de l'expérience accomplit.

Dans toutes les expériences voici les valeurs du workload utilisées :

- Les flux de données ont une moyenne de 100mb/s.
- Maximum TCP MTU 1500 octect.
- Les premiers tests ont été refait 2 fois de suite pour montrer que le Seed du workload (iPerf) change à chaque envoi de flux.
- Commende additionnel pour envoyer le trafic marquer 0x(valeur TOS)
- Numéro de port utiliser 5001.
- Modifier l'intervalle de capture des flux -i (temps de capture).
- Modifier le temps du test -t (valeur de temps en secondes).

### 5.4.1. Expérience 1 : Implémentation du traffic shaping

Dans cette première expérience nous allons implémenter la technique du traffic shaping en format hybride entre le réseau traditionnel et SDN. Pour cela, on doit d'abord configurer les OpenvSwitch avec une politique QoS et la(s) file(s) d'attente(s) avec différentes bandes passantes. Ensuite nous devons matcher les files d'attentes à la politique QoS, puis appliquer la QoS au port choisi. A la fin il nous faut orienter le flux vers la file d'attente configurer avec la QoS, pour cela il nous faudra utiliser l'api Northbound REST pour envoyer le code source au contrôleur OpenDaylight.

Tout d'abord on configure les OpenvSwitch en utilisant le code source qui suit sur CLI (Commande Line Interface) :

**Code Source :**

```
mininet> sh sudo ovs-vsctl set port s1-eth1 qos=@newqos -- --id=@newqos create q
os  type=linux-htb other-config:max-rate=50000000 queues=1=@q1 -- --id=@q1 creat
e queue other-config:max-rate=50000000 other-config:min-rate=15000000
13cac1b7-99ca-4a3f-9a2c-0bd3dcdf1916
74ab3223-3e44-4d22-b708-4aa87c8e68ef
```

Figure 21 : Code source de la création d'une QoS et file d'attente sur le port s1-eth1 du OpenvSwitch

Dans la Figure 21, on crée une politique de QoS de type linux-htb avec une bande passante maximum de 50 mb/s. Ensuite une file d'attente q1 est créée, avec ID de file d'attente = 1, on ajoute par la suite la file d'attente dans la politique de QoS. La file d'attente q1 a un taux de transfert minimum de 15 mb/s et un taux de transfert maximum de 50 mb/s.

Il faut savoir que la politique de QoS n'est appliquée qu'à la sortie du port, ce qui signifie que les taux seront applicables lorsque les flux sont transférés à la sortie du port seulement.

Dans OVSDB, elle se manifeste sous la forme de nouvelles entrées dans la table de QoS et la table des files d'attente. OVSDB met alors une relation entre l'entrée s1-eth1 dans la table bridge et l'entrée QoS nouvellement créée. Cette relation indique que s1-eth1 doit se comporter selon les règles énoncées dans cette QoS. Au cours de ce processus, le commutateur appelle l'application TC (Traffic Control) pour créer Qdisc et des classes en arrière-plan.

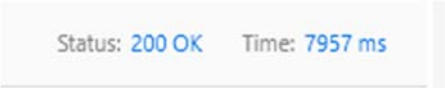
Maintenant on va appliquer le mappage du flux avec notre QoS en se servant du logiciel Postman, qui nous permet d'utiliser l'API Northbound REST pour envoyer notre code en langage « XML » comme suit :

**Code Source en XML :**

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <flow-name>Troll</flow-name>
    <id>1</id>
    <cookie>103</cookie>
    <cookie_mask>255</cookie_mask>
    <table_id>0</table_id>
    <priority>1000</priority>
    <hard-timeout>1800</hard-timeout>
    <idle-timeout>3600</idle-timeout>
    <installHw>false</installHw>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <set-queue-action>
                      <queue-id>1</queue-id>
                    </set-queue-action>
                      <output-action>
                <output-node-connector>NORMAL</output-node-connector>
                      <max-length>65535</max-length>
                      </output-action>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <match>
```

```
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
        </ethernet-match>
        <ipv4-source>10.0.0.0/8</ipv4-source>
        <ipv4-destination>10.0.0.1/32</ipv4-destination>
        <ip-match>
            <ip-protocol>4</ip-protocol>
        </ip-match>
        <tcp-destination-port>5001</tcp-destination-port>
    </match>
</flow>
```

A la fin Postman doit afficher un message comme ceci :




Une fois la QoS implémenté pour le trafic allant au host1, on va répéter les mêmes étapes pour le trafic allant au host2 avec une valeur différente du traffic shaping :

**Code Source :**

```
mininet> sh sudo ovs-vsctl set port s1-eth2 qos=@newqos2 -- --id=@newqos2 create
 qos  type=linux-htb other-config:max-rate=20000000 queues=2=@q2 -- --id=@q2 cre
ate queue other-config:max-rate=20000000 other-config:min-rate=5000000
c0a96974-7f85-4bc6-9012-b0fc732b7e97
b6132d39-02b3-4d2b-a6df-4938a08831b6
```

Figure 22 : Code Source de la création d'une QoS et file d'attente sur le port s1-eth2 du OpenvSwitch

Dans la Figure 22, on crée une politique de QoS de type « linux-htb » avec une bande passante maximum de 20 mb/s. Ensuite une file d'attente q2 est créée avec ID de file d'attente = 2 et on ajoute la file d'attente dans la politique de QoS. La file d'attente q2 a un taux de transfert minimum de 5 mb/s et un taux de transfert maximum de 20 mb/s.

Maintenant on applique le mappage du flux avec notre QoS en utilisant l'API REST, on envoie le code source en « XML » suivant :

**Code Source en XML :**

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <flow-name>Troll</flow-name>
    <id>2</id>
    <cookie>103</cookie>
    <cookie_mask>255</cookie_mask>
    <table_id>0</table_id>
    <priority>1000</priority>
    <hard-timeout>1800</hard-timeout>
    <idle-timeout>3600</idle-timeout>
    <installHw>false</installHw>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <set-queue-action>
                      <queue-id>2</queue-id>
                    </set-queue-action>
                      <output-action>
                  <output-node-connector>NORMAL</output-node-connector>
                        <max-length>65535</max-length>
                        </output-action>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <match>
       <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
        </ethernet-match>
        <ipv4-source>10.0.0.0/8</ipv4-source>
        <ipv4-destination>10.0.0.2/32</ipv4-destination>
        <ip-match>
            <ip-protocol>4</ip-protocol>
        </ip-match>
        <tcp-destination-port>5001</tcp-destination-port>
    </match>
</flow>
```

Pour envoyer les codes en « XML » on doit utiliser l'adresse web qui suit dans Postman avec les paramètres adéquats :

- Méthode d'envoi de la requête vers le contrôleur = « PUT ».
- Type du contenu = « application/xml ».
- L'Autorisation : « Basic Auth » (nom d'utilisateur/mot de passe : admin/admin, par défauts).
- L'adresse web : http://192.168.209.159:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/1

Dans l'adresse web utiliser les deux valeurs essentielles dont il faut toujours s'assurer qu'ils sont à l'identique avec celles de nos codes XML sont :

- Identifiant de la table « table/0 » dans l'adresse web et « <table_id>0</table_id> » dans nos codes XML. On a choisi d'utiliser la table de flux numéro 0, c'est la table principale créé par défauts par le contrôleur.

- Identifiant du flux « flow/1 » dans l'adresse web et « <id>1</id> » dans le premier code XML.

- Identifiant du flux « flow/2 » dans l'adresse web et « <id>2</id> » dans le deuxième code XML.

Comme on peut le remarquer dans nos codes XML on a défini plusieurs éléments essentiels à commencer par la priorité de notre flux mise a « 1000 », cela est très important car dans la table de flux lorsque le commutateur voudra exécuter une action il y a une hiérarchie défini par une priorité du flux, par défaut les flux ont une priorité de « 2 », donc le commutateur commencera par notre code vue qu'il a une plus grande priorité, s'il y a un match avec le reste des commandes alors il exécutera notre action.

Maintenant qu'on a appliqué notre QoS avec succès, on peut vérifier sont applicabilité sur les OpenvSwitch en utilisant la commande « ovs-vsctl list qos » sur Mininet, voici le résultat obtenu :

**Output de l'OVS :**

```
mininet> sh ovs-vsctl list qos
_uuid               : c0a96974-7f85-4bc6-9012-b0fc732b7e97
external_ids        : {}
other_config        : {max-rate="20000000"}
queues              : {2=b6132d39-02b3-4d2b-a6df-4938a08831b6}
type                : linux-htb

_uuid               : 13cac1b7-99ca-4a3f-9a2c-0bd3dcdf1916
external_ids        : {}
other_config        : {max-rate="50000000"}
queues              : {1=74ab3223-3e44-4d22-b708-4aa87c8e68ef}
type                : linux-htb
```

Figure 23 : Résultat après la création des QoS sur le port s1-eth1 et s1-eth2 du OpenvSwitch

Comment on peut le constater sur la Figure 23 la première QoS a bien été appliquer avec une bande passante maximale de 50 mb/s, on remarque aussi la file d'attente q1 attacher à cette dernière.

On constate aussi que la deuxième QoS a été appliquer avec une bande passante maximale de 20 mb/s, on remarque que la file d'attente q2 a été attacher à elle.

On vérifie la bande passante attribué à la file d'attente ainsi que ces taux maximum et minimum avec la commande « ovs-vsctl list queue » sur Mininet, voici le résultat obtenu sur nos commutateurs OpenvSwitch :

**Output de l'OVS :**

```
mininet> sh ovs-vsctl list queue
_uuid               : 31007265-8787-4b14-b165-5630f1f2cac9
dscp                : []
external_ids        : {}
other_config        : {max-rate="50000000", min-rate="15000000"}

_uuid               : 242e25f5-a758-4ab1-b049-88a4b9fbd94f
dscp                : []
external_ids        : {}
other_config        : {max-rate="20000000", min-rate="5000000"}
```

Figure 24 : Résultat après la création des files d'attente dans l'OpenvSwitch

Comme on peut le constater sur la Figure 24, pour la file d'attente q1 le taux de transfert maximum est bien en règle à 50 mb/s, ainsi que le taux de transfert minimum qui est à 15 mb/s. Pour la file d'attente q2 le taux de transfert maximum est bien en règle à 20 mb/s, et le transfert minimum à 5 mb/s.

## a. 1er Test : Avant l'implémentation du traffic shaping

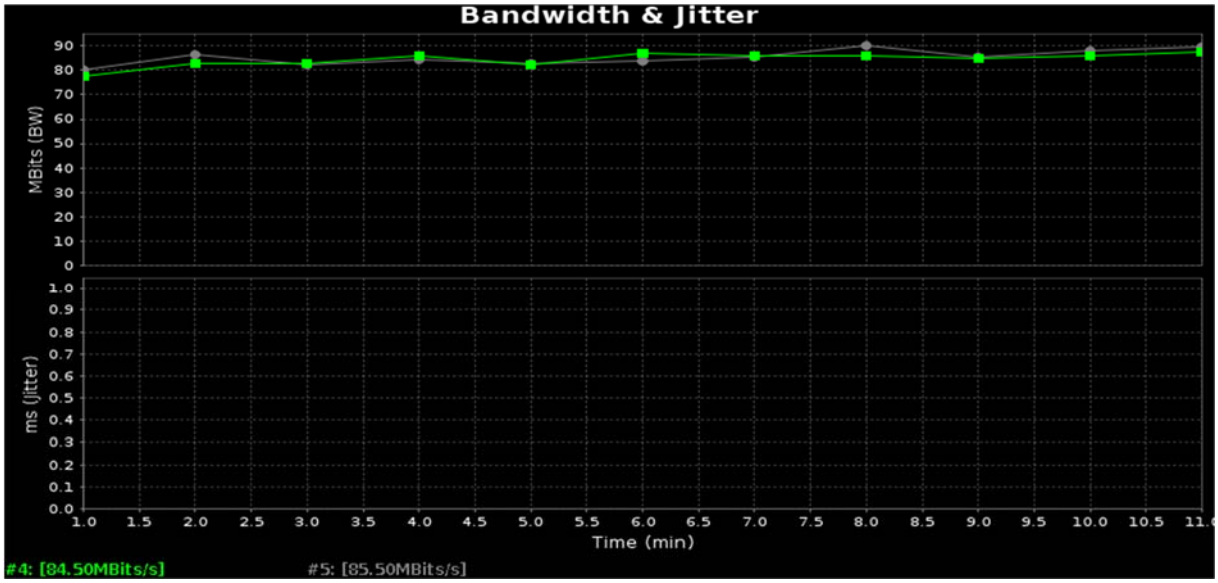



Figure 25 : Graphes de test TCP entre 2 PC représentants les résultats avant l'implémentation du traffic shaping

Pour les graphes de la Figure 25 nous avons exécuté le test à deux reprises, comme on le remarque on a obtenu un graphe vert pour le premier trafic généré et blanc représentant le deuxième trafic généré.

Les deux graphes obtenus représentent la bande passante en fonction du temps (capturée à chaque minute) suite à la génération de deux trafics TCP allant du PC1 vers le PC2, ou le lien entre les deux PC est 100 mb/s en utilisant le numéro de port 5001 et ceci sans application d'une politique QoS.

**Constat :**

Les deux tests ont duré 10 minutes. Le graphe en vert et le graphe en blanc qui représentent le premier et deuxième trafic TCP généré asynchrones allant tout deux du PC1 vers le PC2, on a constaté rien d'anormale, les graphes sont stables avec une bande passante entre 80 mb/s et 90 mb/s.

On a obtenu une moyenne de la bande passante estimée à 84.50 mb/s pour le premier test et 85.50 mb/s pour le deuxième test.

## b. 2ème Tests : Après l'implémentation du traffic shaping

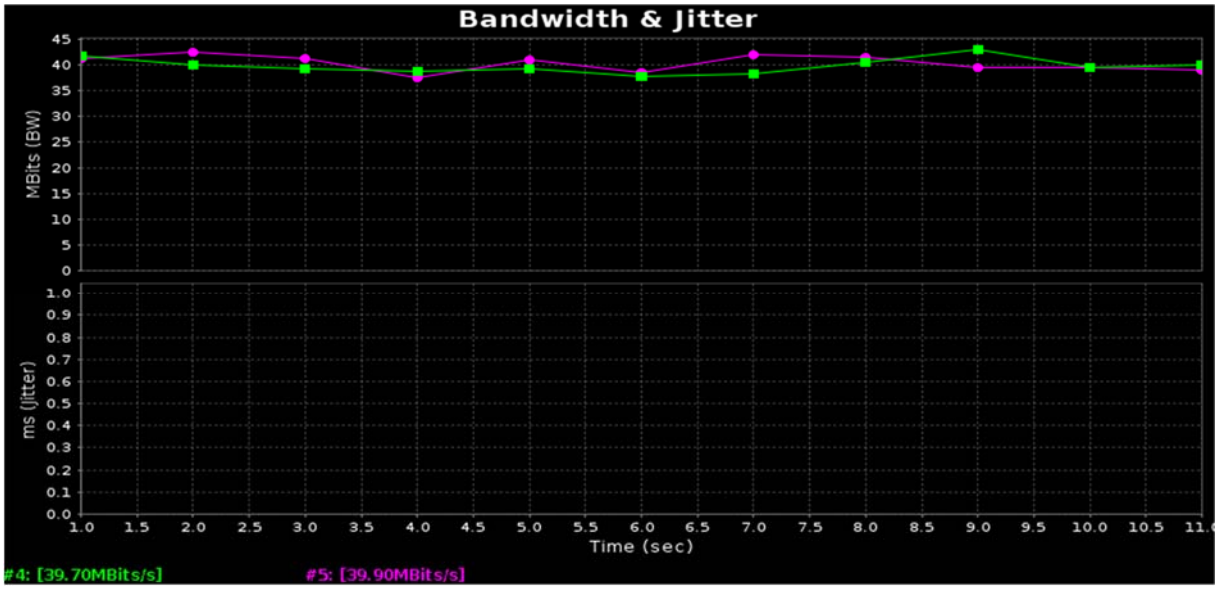



Figure 26 : Graphes de test TCP entre 2 PC représentants les résultats après l'implémentation du traffic shaping dans le port s1-eth1

Pour les graphes de la Figure 26 nous avons le test exécuté à deux reprises, comme on le remarque on a obtenu un graphe vert pour le premier trafic généré et rose représentant le deuxième trafic généré.

Les deux graphes obtenus représentent la bande passante en fonction du temps (capturée à chaque seconde), on a généré un trafic TCP allant du PC2 vers le PC1, en appliquant la politique du traffic shaping avec une bande passante minimisé à 50 mb/s pour le trafic allant au PC1.

**Constat :**

Les deux tests ont duré 10 secondes. Le graphe en vert et le graphe en rose qui représentent le premier et deuxième trafic TCP généré asynchrones allant tout deux du PC2 vers le PC1, on a constaté rien d'anormale, les graphes sont stables avec une bande passante entre 37 mb/s et 42 mb/s.

Il n'y a pas eu d'excès sur la limite de la bande passante implémenté, on a obtenu une moyenne de la bande passante estimée à 39.70 mb/s pour le premier test et 39.90 mb/s pour le deuxième test.

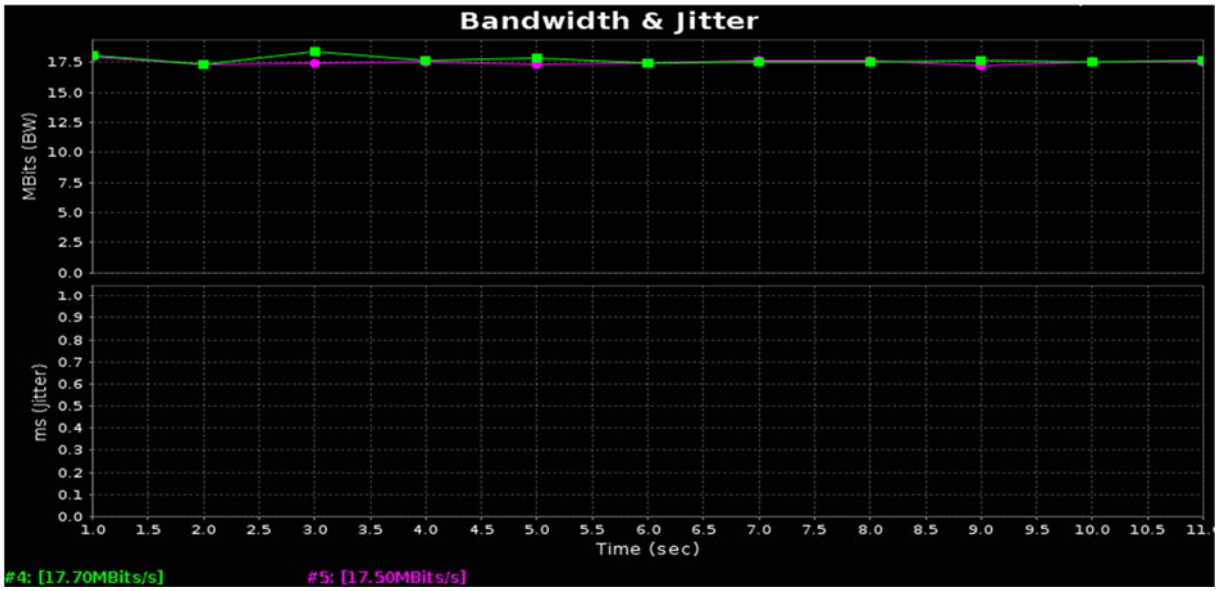



Figure 27 : Graphes de test TCP entre 2 PC représentants les résultats après l'implémentation du traffic shaping dans le port s1-eth2

Pour les graphes de la Figure 27 nous avons exécuté le test à deux reprises, comme on le remarque on a obtenu un graphe vert pour le premier trafic généré et mauve représentant le deuxième trafic généré.

Les deux graphes obtenus représentent la bande passante en fonction du temps (capturée à chaque seconde), on a généré un trafic TCP allant du PC1 vers le PC2, en appliquant la politique du traffic shaping avec une bande passante minimisé à 20 mb/s pour le trafic allant au PC2.

**Constat :**

Les deux tests ont duré 10 secondes. Le graphe en vert et le graphe en mauve qui représentent le premier et deuxième trafic TCP généré asynchrones allant tout deux du PC1 vers le PC2, on a constaté rien d'anormale, les graphes sont stables avec une bande passante entre 16 mb/s et 18 mb/s.

Il n'y a pas eu d'excès sur la limite de la bande passante implémenté, on a obtenu une moyenne de la bande passante estimée à 17.70 mb/s pour le premier test et 17.50 mb/s pour le deuxième test.

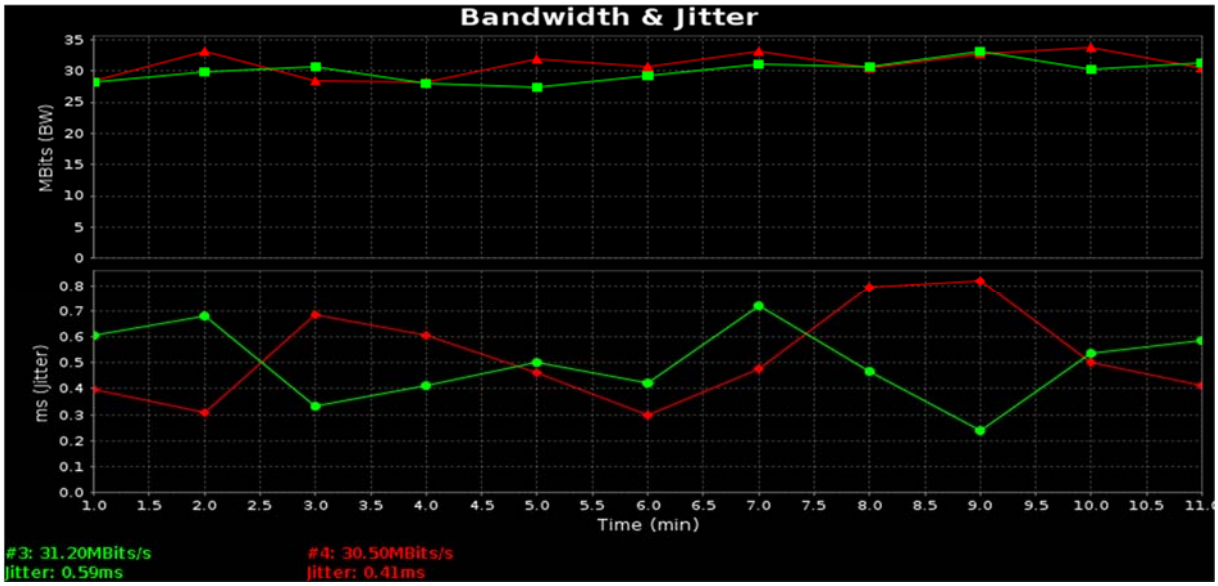



Figure 28 : Graphes de test UDP entre 2 PC représentants les résultats après l'implémentation du traffic shaping dans le port s1-eth1

Pour les graphes de la Figure 28 nous avons exécuté le test à deux reprises, comme on le remarque on a obtenu un graphe vert pour le premier trafic généré et rouge représentant le deuxième trafic généré.

Les deux graphes obtenus représentent la bande passante en fonction du temps (capturée à chaque minute), on a généré un trafic UDP allant du PC2 vers le PC1, en appliquant la politique du traffic shaping avec une bande passante minimisé à 50 mb/s pour le trafic allant au PC1.

**Constat :**

Les deux tests ont duré 10 minutes. Le graphe en vert et le graphe en rouge qui représentent le premier et deuxième trafic UDP généré asynchrones allant tout deux du PC2 vers le PC1, on a constaté rien d'anormale, les graphes sont stables avec une bande passante entre 28 mb/s et 35 mb/s, et un jitter entre 0.2ms et 0.8ms.

On a obtenu une moyenne de la bande passante de 31.20 mb/s pour le premier graphe vert et 30.50 mb/s pour le deuxième graphe en rouge.

On a aussi remarqué que les valeurs du jitter sont éloignés, en ce qui concerne le premier graphe en vert, on a obtenu une moyenne de jitter de 0.59ms, également pour le deuxième en rouge la moyenne est 0.41ms.

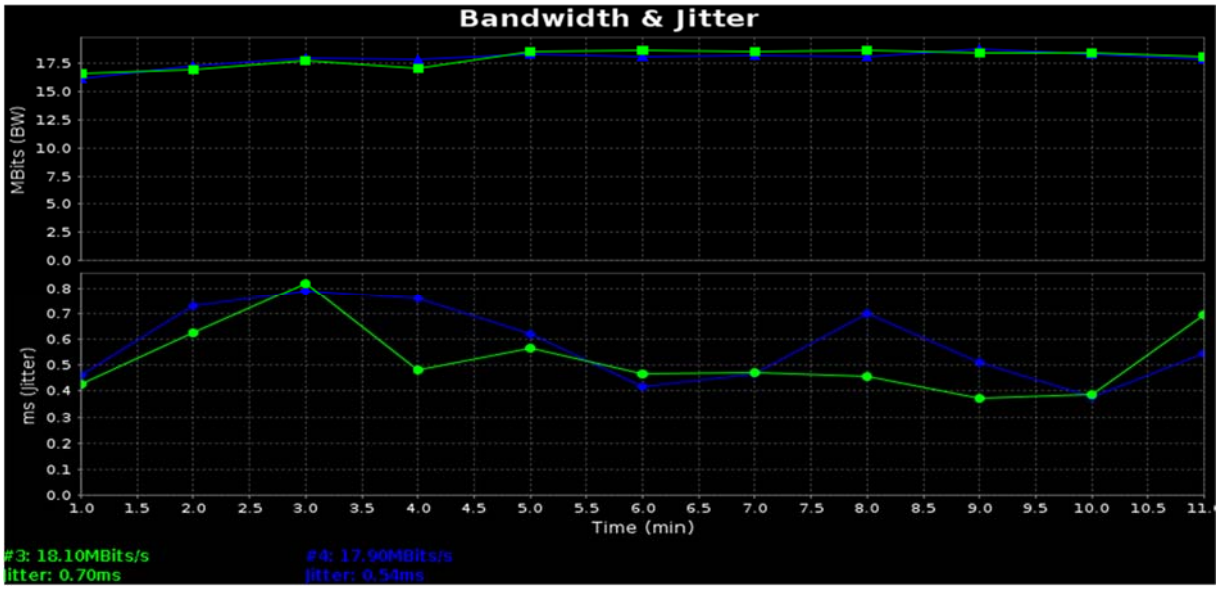



Figure 29 : Graphes de test UDP entre 2 PC représentants les résultats après l'implémentation du traffic shaping dans le port s1-eth2

Pour les graphes de la Figure 29 nous avons exécuté le test à deux reprises, comme on le remarque on a obtenu un graphe vert pour le premier trafic généré et bleu représentant le deuxième trafic généré.

Les deux graphes obtenus représentent la bande passante en fonction du temps (capturée à chaque minute), on a généré un trafic UDP allant du PC1 vers le PC2, en appliquant la politique du traffic shaping avec une bande passante minimisé à 20 mb/s pour le trafic allant au PC2.

**Constat :**

Les deux tests ont duré 10 minutes. Le graphe en vert et le graphe en bleu qui représentent le premier et deuxième trafic UDP généré asynchrones allant tout deux du PC1 vers le PC2, on a constaté rien d'anormale, les graphes sont stables avec une bande passante entre 16 mb/s et 18.50 mb/s, et un jitter entre 0.4ms et 0.8ms.

On a obtenu une moyenne de la bande passante de 18.10 mb/s pour le premier graphe vert et 17.90 mb/s pour le deuxième graphe en bleu.

On a aussi remarqué que les valeurs du jitter sont éloignés, en ce qui concerne le premier graphe en vert, on a obtenu une moyenne de jitter de 0.70ms, également pour le deuxième en bleu la moyenne est 0.54ms.

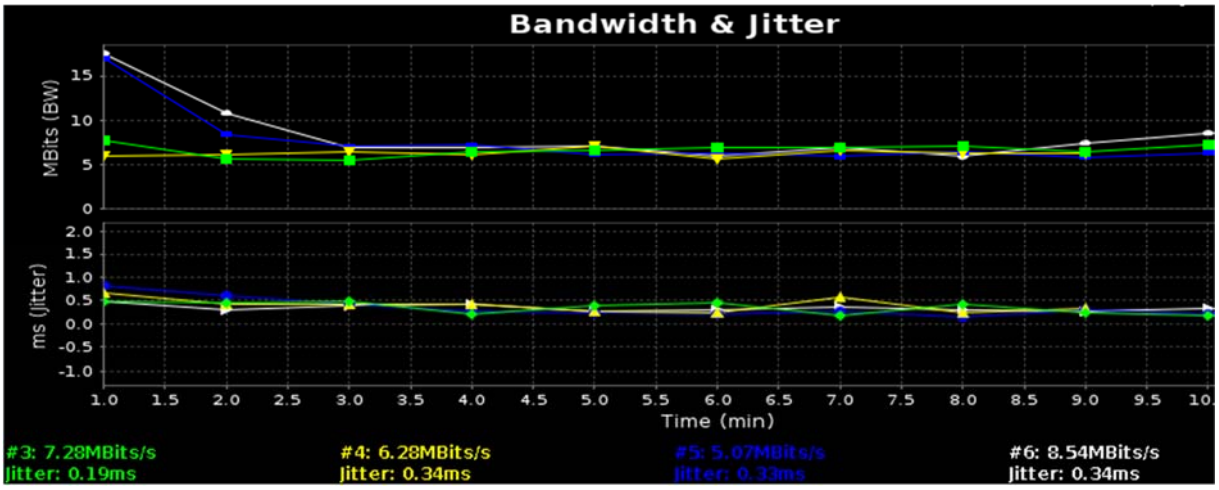



Figure 30 : Graphes de test UDP entre 6 PC représentants les résultats après l'implémentation du traffic shaping dans le port s1-eth2

Les graphes de la Figure 30 représentent la bande passante et jitter en fonction du temps (capturés à chaque minute), suite à la génération d'un trafic UDP allant du PC3 au PC2, PC4 au PC2, PC5 au PC2 et PC6 au PC2, en appliquant la politique du traffic shaping avec une bande passante minimisé à 20 mb/s pour le trafic allant au PC2.

En parallèle on a généré un trafic TCP entre le PC1 et PC2 pour augmenter plus la charge sur le réseau.

**Constat :**

Les tests ont duré 10 minutes. Le graphe vert représente la bande passante et le jitter entre PC3 et PC2, ou on a obtenu une moyenne de la bande passante de 7.28 mb/s et une moyenne de jitter de 0.19ms.

Le graphe jaune représente la bande passante et le jitter entre PC4 et PC2, ou on a obtenu une moyenne de bande passante de 6.28 mb/s, et une moyenne de jitter de 0.34ms

Le graphe bleu représente la bande passante et le jitter entre PC5 et PC2, ou on a obtenu une moyenne de bande passante de 5.07 mb/s, et une moyenne de jitter de 0.33ms.

Le graphe blanc représente la bande passante et le jitter entre PC6 et PC2, ou on a obtenu une moyenne de bande passante de 8.54 mb/s, et une moyenne de jitter de 0.34ms.

On a obtenu une moyenne de la bande passante et jitter assez proche pour chaque flux entre chaque PC.

## c. Remarque

La remarque la plus importante établie d'après tous les tests effectués est que la limite maximale et minimale de la bande passante a bien été respecter pour tous les cas, même quand la congestion était présente. Nous avons remarqué aussi que la moyenne des jitters sont assez proches (donc aucune priorité en ce qui concerne le jitter). Pour conclure le traffic shaping a été implémenter avec succès en conférant les résultats attendus.

## 5.4.2. Expérience 2 : Implémentation de DSCP

Dans cette deuxième expérience on va implémenter la méthode du DSCP en format pure SDN. Tout d'abord on doit s'assuré qu'OpenDaylight soit capable d'interagir avec les fonctionnalités de QoS. Après cela on va créer notre QoS dans le contrôleur OpenDaylight, il l'utilisera pour créer une entré dans la table de flux des OpenvSwitch, qui sera par la suite capable de reconnaitre un flux prioritaire en lui accordant sa priorité souhaitée.

Pour commencer on active les fonctionnalités nécessaires sur notre contrôleur pour qu'il accepte une commande complexe contenant le DSCP, il suffit d'entrer ces lignes sur la CLI OpenDaylight :

- feature:install odl-nic-core-mdsal
- feature :install odl-nic-console
- feature :install odl-nic-listeners

Maintenant qu'OpenDaylight est prêt à recevoir notre code de QoS, nous pouvons choisir l'une des deux approches suivantes pour l'injecter aux OpenvSwitch :

### a. 1ère Approche : Méthode d'injection de la QoS par le CLI de l'OpenDaylight

On commence par créer notre QoS sur la CLI du contrôleur avec la commande « intent » suivante :

- intent:qosConfig -p Highest_Quality -d 46
- intent:qosConfig -p Lowest_Quality -d 4

Comme on peut le remarquer nous utiliserons 2 QoS pour nos tests. La première qu'on a nommée « Highest_Quality » est plus prioritaire avec une valeur DSCP de 46 traduite en TOS par 184, qui signifie état critique. La deuxième nommée « Lowest_Quality » est de priorité inférieure par rapport à la première avec une valeur de DSCP de 4 traduite en TOS par 16, qui signifie état de routine.

Une fois notre QoS créer, on doit l'appliquer au flux dans les deux sens entre la(es) machines qu'en veut prioriser par rapport aux autres. Pour cela on utilise la commande « intent » suivante sur la CLI du contrôleur OpenDaylight :

- intent:add -a ALLOW -t 00:00:00:00:00:02 -f 00:00:00:00:00:01 -q QOS -p Highest_Quality
- intent:add -a ALLOW -t 00:00:00:00:00:01 -f 00:00:00:00:00:02 -q QOS -p Highest_Quality
- intent:add -a ALLOW -t 00:00:00:00:00:01 -f 00:00:00:00:00:03 -q QOS -p Lowest_Quality
- intent:add -a ALLOW -t 00:00:00:00:00:03 -f 00:00:00:00:00:01 -q QOS -p Lowest_Quality

Comme on peut le remarquer les deux machines choisies pour avoir un flux d'une priorité critique entre eux sont PC1 et PC2 avec leurs adresses MAC 00:00:00:00:00:01 et 00:00:00:00:00:02 respectivement. On remarque aussi un flux moins important que le premier entre les deux machines PC1 et PC3 avec leurs adresses MAC 00:00:00:00:00:01 et 00:00:00:00:00:03 respectivement, une priorité de routine lui a été affecter.

L'argument -a est pour « action », on a affecté la commande « ALLOW » dans les deux cas pour autoriser le trafic à circuler. L'argument -t est pour « adresse MAC de destination » en opposé

nous avons la commande -f qui est pour « adresse MAC source ». L'argument -q est pour « qualité de service », -p est pour le nom de la QoS qu'on veut attribuer.

Maintenant qu'on a appliqué nos valeurs DSCP aux flux souhaités, en utilisant la commande « sudo ovs-ofctl dump-flows s1 » sur n'importe quel OpenvSwitch de notre topologie (car tous les OpenvSwitch de notre topologie ont reçus notre flux d'une façon automatique par le contrôleur) voici le résultat afficher :

**Output de l'OVS :**

```
mininet> sh sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=48.666s, table=0, n_packets=0, n_bytes=0, idle_age=48, pri
ority=9000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02 actions=NORMAL,mod_
nw_tos:184
 cookie=0x0, duration=25.908s, table=0, n_packets=0, n_bytes=0, idle_age=25, pri
ority=9000,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01 actions=NORMAL,mod_
nw_tos:16
 cookie=0x0, duration=38.048s, table=0, n_packets=0, n_bytes=0, idle_age=38, pri
ority=9000,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01 actions=NORMAL,mod_
nw_tos:184
 cookie=0x0, duration=13.417s, table=0, n_packets=0, n_bytes=0, idle_age=13, pri
ority=9000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03 actions=NORMAL,mod_
nw_tos:16
```

Figure 31 : Résultat après la création des QoS avec la 1ère méthode DSCP

Nous remarquons dans la Figure 31 que nos flux ont bien été créé dans la table des flux des OpenvSwitch, la priorité du flux (9000) a été automatiquement fixé par notre contrôleur, la priorité du DSCP entre le PC1 et PC2 dans les deux sens est bien correcte nous remarquons leurs adresses MAC 00:00:00:00:00:01 et 00:00:00:00:00:02 respectivement avec une valeur de TOS 'new_tos :184' référent a notre QoS Highest_Quality, de même pour le flux entre PC1 et PC3 avec leurs adresses MAC 00:00:00:00:00:01 et 00:00:00:00:00:03 respectivement, nous remarquons pour les deux sens une valeur de TOS 'new_tos :16' référent a notre QoS Lowest_Quality.

## 1er Test : Avant l'implémentation du DSCP

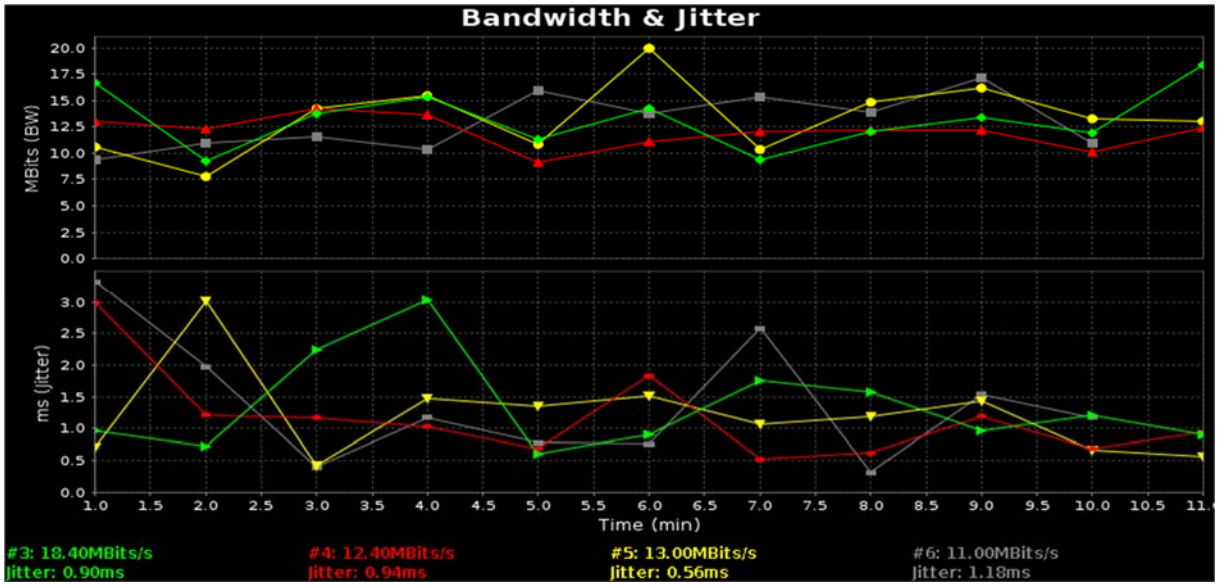



Figure 32 : Graphes de test UDP entre 5 PC représentants les résultats avant l'implémentation du DSCP

Les graphes de la Figure 32 représentent la bande passante et jitter en fonction du temps (capturés à chaque minute pendant 10 minutes), suite à la génération d'un trafic UDP allant du PC3 au PC2, PC4 au PC2, PC5 au PC2 et PC6 au PC2, sans appliqué de politique QoS.

**Constat :**

Le test a duré 10 minutes. Le graphe vert représente la bande passante et le jitter entre PC5 et PC2, ou on a obtenu une moyenne de la bande passante de 18.40 mb/s et une moyenne de jitter de 0.90ms.

Le graphe rouge représente la bande passante et le jitter entre PC3 et PC2, ou on a obtenu une moyenne de bande passante de 12.40 mb/s, et une moyenne de jitter de 0.94ms

Le graphe jaune représente la bande passante et le jitter entre PC6 et PC2, ou on a obtenu une moyenne de bande passante de 13.00 mb/s, et une moyenne de jitter de 0.56ms.

Le graphe gris représente la bande passante et le jitter entre PC4 et PC2, ou on a obtenu une moyenne de bande passante de 11.00 mb/s, et une moyenne de jitter de 1.18ms.

Avec tous ces trafics qui se dispute la bande passante, aucun d'entre eux n'as bénéficié d'une priorité quelconque en termes de bande passante et jitter.

## 2ème Tests : Après l'implémentation du DSCP avec la 1ère approche

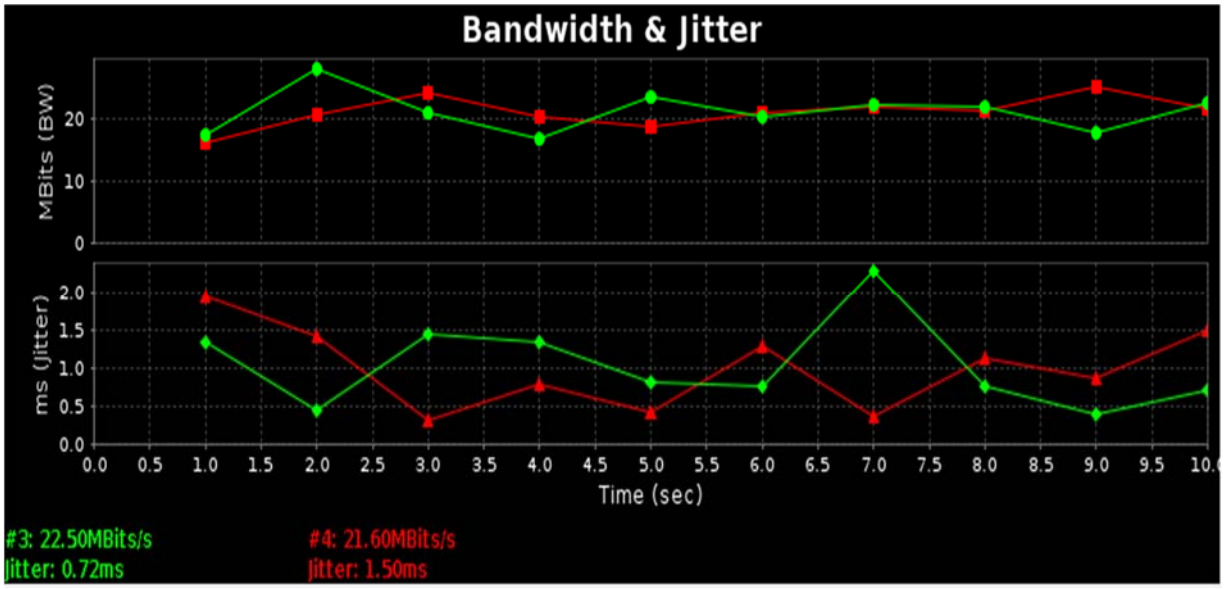



Figure 33 : Graphes de test UDP entre 3 PC représentants les résultats après l'implémentation du DSCP

Les graphes de la Figure 33 représentent la bande passante et jitter en fonction du temps (capturés à chaque secondes pendant 10 secondes), après avoir appliqué la méthode DSCP, en utilisant NBI (REST API) pour déterminer la valeur de DSCP. On a généré un flux UDP entre PC1 et PC2 avec une priorité DSCP qui égale à 46 (TOS=184). Au même temps on a aussi généré un flux UDP entre PC1 et PC3 avec une priorité DSCP de 4 (TOS=16).

**Constat :**

Le test a duré 10 secondes. Le graphe en vert représente la bande passante ainsi que le jitter entre PC1 et PC2 (le trafic le plus favorisé), on a obtenu une moyenne de bande passante de 22.50 mb/s et des valeurs de jitter rapprochées d'une moyenne de de 0.72ms.

Le graphe en rouge représente la bande passante ainsi que le jitter entre PC1 et PC2 (le trafic le moins favorisé), on a obtenu une moyenne de bande passante de 21.60 mb/s et des valeurs de jitter rapprochées d'une moyenne de de 1.50ms.

La QoS a bien été respecté pour notre trafic prioritaire qui a surpassé le trafic non prioritaire en termes de bande passante et jitter.

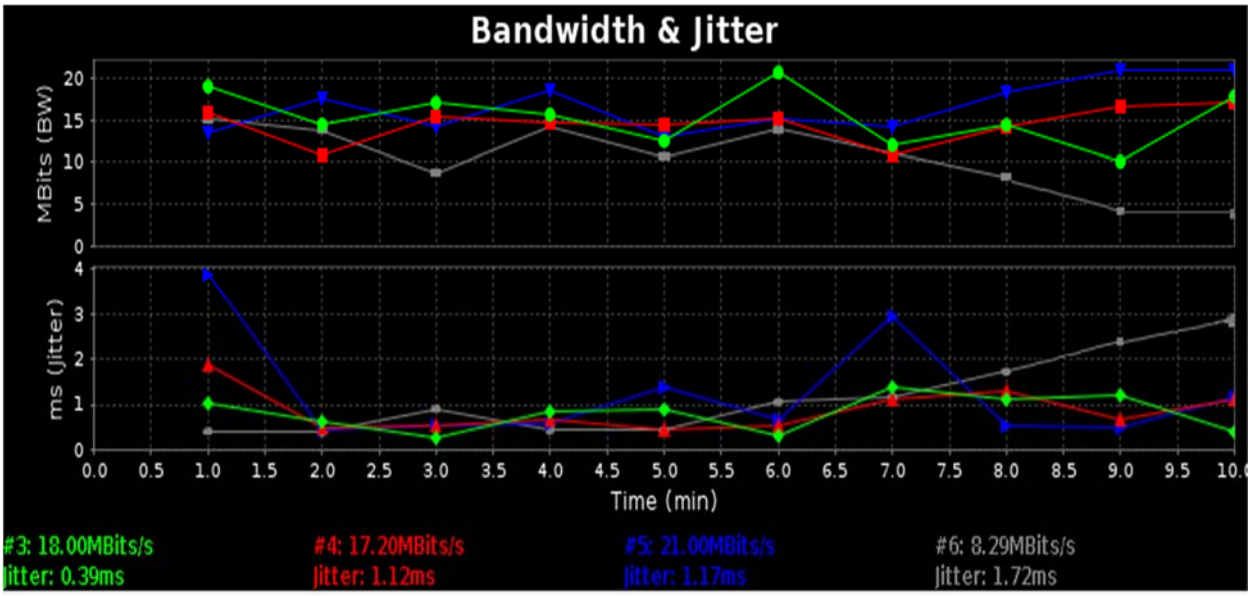



Figure 34 : Graphes de test UDP entre 5 PC représentants les résultats après l'implémentation du DSCP

Les graphes de la Figure 34 représentent la bande passante et jitter en fonction du temps (capturés à chaque minute pendant 10 minutes), après avoir appliqué la méthode DSCP, en utilisant NBI (REST API) pour déterminer la valeur de DSCP. On a généré un flux UDP entre PC1 et PC3 avec une priorité DSCP qui égale à 46 (TOS=184). On a aussi généré un flux UDP entre le PC1 avec PC4, PC5, PC6 avec une priorité DSCP de 4 (TOS=16).

**Constat :**

Le test a duré 10 minutes. Le graphe en vert représente la bande passante ainsi que le jitter entre PC1 et PC4, on a obtenu une moyenne de bande passante de 18 mb/s et des valeurs de jitter d'une moyenne de de 0.39ms.

Le graphe en rouge représente la bande passante ainsi que le jitter entre PC1 et PC5, on a obtenu une moyenne de bande passante de 17.20 mb/s et des valeurs de jitter d'une moyenne de de 1.12ms.

Le graphe en bleu représente la bande passante ainsi que le jitter entre PC1 et PC3 (le trafic le plus favorisé), on a obtenu une moyenne de bande passante de 21.00 mb/s et des valeurs de jitter d'une moyenne de de 1.17ms.

Le graphe en gris représente la bande passante ainsi que le jitter entre PC1 et PC6, on a obtenu une moyenne de bande passante de 8.29 mb/s et des valeurs de jitter d'une moyenne de de 1.72ms.

On remarque que le flux le plus prioritaire en bleu a une moyenne de bande passante plus que les autres flux mais la moyenne de son jitter est plus élevé (jitter trop instable) que certains flux moins prioritaires.

## b. 2$^{me}$ Approche : Méthode par injection de la QoS en utilisant l'API REST

On commence avec l'utilisation du logiciel Postman, on va envoyer notre code en langage XML avec l'API REST :

**Code source en Xml :**

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <flow-name>Trollo2</flow-name>
    <id>3</id>
    <cookie_mask>255</cookie_mask>
    <cookie>103</cookie>
    <table_id>0</table_id>
    <priority>1002</priority>
    <hard-timeout>1200</hard-timeout>
    <idle-timeout>3400</idle-timeout>
    <installHw>false</installHw>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <output-action>
                  <output-node-connector>NORMAL</output-node-connector>
                        <max-length>60</max-length>
                    </output-action>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
            <ethernet-destination>
                <address>00:00:00:00:00:01</address>
            </ethernet-destination>
            <ethernet-source>
                <address>00:00:00:00:00:03</address>
            </ethernet-source>
        </ethernet-match>
        <ip-match>
            <ip-protocol>4</ip-protocol>
            <ip-dscp>46</ip-dscp>
            <ip-ecn>0</ip-ecn>
        </ip-match>
    </match>
</flow>
```

A la fin Postman doit afficher un message comme ceci :


On doit appliquer notre code dans les deux sens, voici le deuxième code :

**Code Source en Xml :**

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <flow-name>Trollo2</flow-name>
    <id>4</id>
    <cookie>103</cookie>
    <table_id>0</table_id>
    <priority>1002</priority>
    <hard-timeout>0</hard-timeout>
    <idle-timeout>0</idle-timeout>
    <installHw>false</installHw>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <output-action>
                  <output-node-connector>NORMAL</output-node-connector>
                        <max-length>60</max-length>
                    </output-action>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
            <ethernet-destination>
                <address>00:00:00:00:00:03</address>
            </ethernet-destination>
            <ethernet-source>
                <address>00:00:00:00:00:01</address>
            </ethernet-source>
        </ethernet-match>
        <ip-match>
            <ip-protocol>4</ip-protocol>
            <ip-dscp>46</ip-dscp>
            <ip-ecn>0</ip-ecn>
```

```
        </ip-match>
    </match>
</flow>
```

Pour envoyer le code on doit utiliser l'adresse web qui suit dans Postman pour nous connecter au contrôleur OpenDaylight avec les paramètres adéquats :

- Méthode d'envoi de la requête vers le contrôleur = « PUT ».
- Type du contenu = « application/xml ».
- L'Autorisation : « Basic Auth » (nom d'utilisateur/mot de passe : admin/admin, par défauts).
- L'adresse web : http://192.168.209.159:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/3

Dans l'adresse web utiliser les deux valeurs essentielles dont il faut toujours s'assurer qu'elles sont à l'identique avec celles de notre code XML sont :

- Identifiant de la table « table/0 » dans l'adresse web et « <table_id>0</table_id> » dans les deux cas de notre code XML, nous remarquons qu'elles sont identiques. On a choisi d'utiliser la table de flux numéro 0, c'est la table principale par défauts créé par le contrôleur.
- Identifiant du flux « flow/3 » dans l'adresse web et « <id>3</id> » dans le premier code XML.
- Identifiant du flux « flow/4 » dans l'adresse web et « <id>4</id> » dans le deuxième code XML.

Comme on peut le remarquer on a appliqué notre QoS entre les deux hôtes PC1 et PC3 avec leurs adresses MAC 00:00:00:00:00:01 et 00:00:00:00:00:03 respectivement, on a appliqué une valeur critique de DSCP comme on peut le constater sur notre code '<ip-dscp>46</ip-dscp>', pour cette deuxième méthode on a appliqué aussi une autre valeur qui est l'ECN, la valeur de l'ECN est 0 comme on peut le constater sur notre code '<ip-ecn>0</ip-ecn>', ceci dit en cas de congestion le flux passant entre PC1 et PC3 ne sera pas rejeter.

Maintenant qu'on a appliqué nos valeurs DSCP aux flux souhaités, en utilisant la commande « sudo ovs-ofctl dump-flows s1 » sur n'importe quel OpenvSwitch de notre topologie (tous les OpenvSwitch de notre topologie ont reçus notre flux d'une façon automatique par le contrôleur) voici le résultat afficher :

**Output de l'OVS :**

```
mininet> sh sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x67, duration=172.668s, table=0, n_packets=0, n_bytes=0, idle_age=172, priorit
y=1002,tcp,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01,nw_tos=184 actions=NORMAL
 cookie=0x67, duration=103.778s, table=0, n_packets=0, n_bytes=0, idle_age=103, priorit
y=1002,tcp,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03,nw_tos=184 actions=NORMAL
```

Figure 35 : Résultat après la création des QoS avec la 2ème méthode DSCP

On remarque que nos flux ont bien été attribué à nos OpenvSwitch, la priorité a été fixé à « 1002 » par notre contrôleur, la priorité du DSCP est correcte entre le PC1 et PC3 avec leurs adresses MAC 00:00:00:00:00:01 et 00:00:00:00:00:03 respectivement, pour les deux sens des flux un attribut de TOS avec comme valeur 'new_tos :184' a été appliquer référent a notre QoS <ip-dscp>46</ip-dscp>.

## 1ère Tests : Après l'implémentation du DSCP avec la 2ème approche

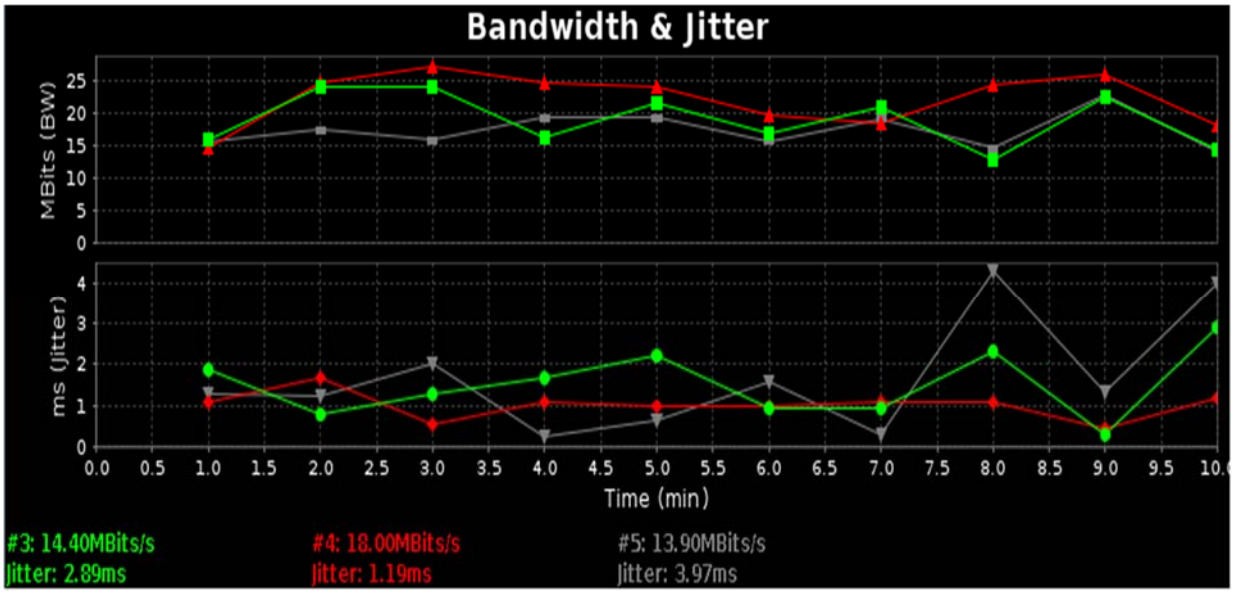



Figure 36 : Graphes de test UDP entre 4 PC représentants les résultats après l'implémentation du DSCP

Les graphes de la Figure 36 représentent la bande passante et jitter en fonction du temps (capturés à chaque minute), après avoir appliqué la méthode DSCP, en utilisant NBI (REST API) pour déterminer la valeur de DSCP et ECN. On a généré un flux UDP entre PC1 et PC3 avec une priorité DSCP qui égale à 46 (TOS=184) et ECN qui égale à zéro, pour éviter le rejet des flux en cas de congestion. On a aussi généré un flux UDP entre PC1 et PC2, PC4.

**Constat :**

Le test a duré 10 minutes. Le graphe en vert représente la bande passante ainsi que le jitter entre PC1 et PC4, on a obtenu une moyenne de bande passante de 14.40 mb/s et des valeurs de jitter d'une moyenne de de 2.89ms.

Le graphe en rouge représente la bande passante ainsi que le jitter entre PC1 et PC3 (le trafic le plus favorisé), on a obtenu une moyenne de bande passante de 18 mb/s et des valeurs de jitter rapprochées d'une moyenne de de 1.19ms.

Le graphe en gris représente la bande passante ainsi que le jitter entre PC1 et PC2, on a obtenu une moyenne de bande passante de 13.90 mb/s et des valeurs de jitter d'une moyenne de de 3.97ms.

On remarque que le flux le plus prioritaire en rouge a une moyenne de bande passante plus que les autres flux, aussi la moyenne de son jitter est plus basse (jitter plus stable) que les autres flux moins prioritaires.

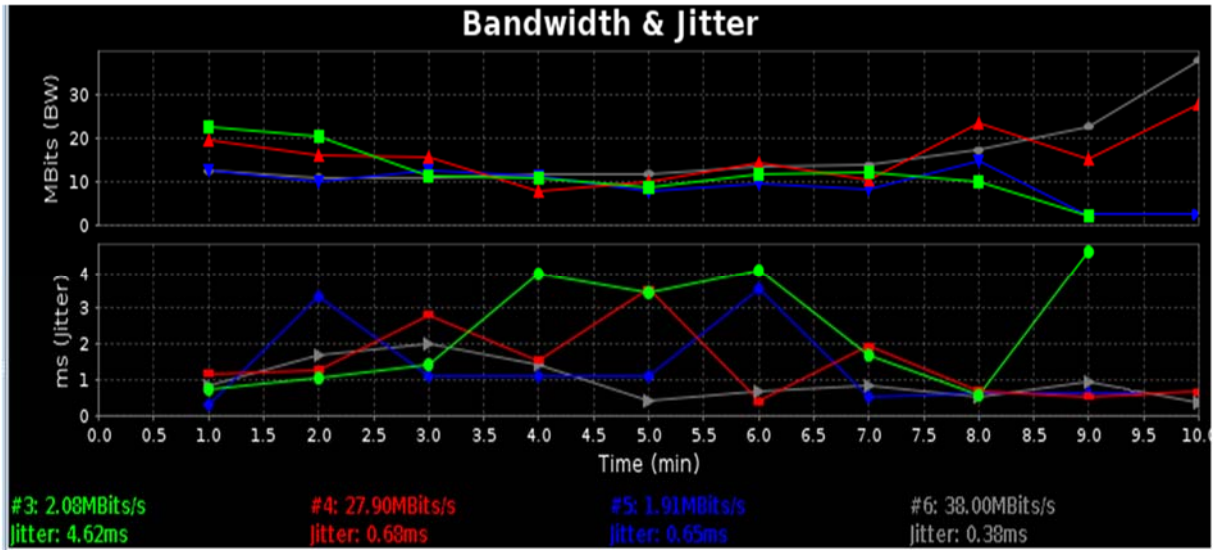



Figure 37 : Graphes de test UDP entre 5 PC représentants les résultats après l'implémentation du DSCP

Les graphes de la Figure 37 représentent la bande passante et jitter en fonction du temps (capturés à chaque minute), après avoir appliqué la méthode DSCP, en utilisant NBI (REST API) pour déterminer la valeur de DSCP et ECN. On a généré un flux UDP entre PC1 et PC3 avec une priorité DSCP qui égale à 46 (TOS=184) et ECN qui égale à zéro, pour éviter le rejet des flux en cas de congestion. On a aussi généré un flux UDP entre PC1 et PC2, PC4, PC5.

**Constat :**

Le test a duré 10 minutes. Le graphe en vert représente la bande passante ainsi que le jitter entre PC1 et PC4, on a obtenu une moyenne de bande passante de 2.08 mb/s et des valeurs de jitter d'une moyenne de de 4.62ms.

Le graphe en rouge représente la bande passante ainsi que le jitter entre PC1 et PC2, on a obtenu une moyenne de bande passante de 27.90 mb/s et des valeurs de jitter d'une moyenne de de 0.68ms.

Le graphe en bleu représente la bande passante ainsi que le jitter entre PC1 et PC5, on a obtenu une moyenne de bande passante de 1.91 mb/s et des valeurs de jitter d'une moyenne de de 0.65ms.

Le graphe en gris représente la bande passante ainsi que le jitter entre PC1 et PC3 (le trafic le plus favorisé), on a obtenu une moyenne de bande passante de 38 mb/s et des valeurs de jitter rapprochées d'une moyenne de de 0.38ms.

On remarque que le flux le plus prioritaire en gris a une moyenne de bande passante plus que les autres flux, aussi la moyenne de son jitter est plus basse (jitter plus stable) que les autres flux moins prioritaires.

### c. Remarque

La remarque la plus importante établie d'après tous les tests précédents est que le trafic priorisé avec une valeur TOS a bénéficié d'une bande passante plus large que les autres, mais pour que son jitter sois plus stable que les autres on a dû lui rajouter une autre valeur qui est l'ECN. Pour conclure le marquage de trafic en utilisant la valeur TOS seulement a donné des résultats moins pertinents en termes de jitter par rapport au TOS avec la valeur ECN ajouté ou le jitter a été plus stable.

## 5.4.3. Comparaison entre les deux solutions

En comparant les résultats des tests obtenus, on déduit que la 2ème approche qui consiste à implémenter le DSCP + l'ECN est meilleur que la 1ère approche qui consiste en l'implémentation du traffic shaping. Le traffic shaping a fourni les résultats attendus en termes de bande passante mais il n'a donné aucun résultat bénéfique pour le jitter par rapport au DSCP + ECN qui ont priorisé la bande passante et jitter du trafic prioritaire.

Enfin malgré que la 2ème approche (DSCP) ne soit plus vulnérable aux attaques réseaux due au lack de sécurité présente dans les NBI du SDN, elle est nettement plus dynamique par rapport à la 1ère approche (traffic shaping), car elle exploite tout le potentiel du SDN avec l'automatisation grâce à notre solution qui consiste à communiquer avec le contrôleur en full réseau SDN (envoie du code source avec seulement le NBI).

## 5.5. Conclusion

Ce chapitre a été consacré aux détails de l'implémentation et l'étude de performances de la QoS dans le SDN, il donne d'abord une présentation globale de l'architecture LAN traditionnel. Après cela, la conception et implémentation de la topologie SD-LAN sous l'émulateur Mininet. On discute ensuite les résultats obtenus après les tests sur les techniques QoS dans le SDN. Enfin on a donné nos points de vue sur la meilleur technique QoS à implémenter pour la priorisation d'un flux bien défini.

# Conclusion générale

La réalisation de ce projet de fin d'étude a été motivée par l'essor rapide du SDN, qui prétend remplacer l'architecture actuelle des réseaux, ainsi que la participation accrue des industries et des recherches dans le domaine de la QoS en SDN. Comme nous l'avons vu tout au long du projet, pour atteindre l'objectif mentionné de priorisation des flux, le SDN doit fournir une meilleure qualité de service. La QoS est le besoin de l'heure, et les chercheurs ont tenté de mettre au point un modèle complet de la QoS.

On a d'abord introduit le réseau SDN après une comparaison avec le réseau traditionnel en donnant ces avantages dans le milieu d'entreprise. Nous sommes passé après cela a la définition de la QoS et comment l'implémenté dans le SDN avec une présentation des solutions proposé dans les travaux connexes.

Nous avons découvert dans ce projet que chaque implémentation de composants SDN fournit un certain niveau de support QoS. La spécification OpenFlow Switch est le principal moteur des fonctionnalités disponibles dans les projets SDN. Bien que la plupart des contrôleurs comme OpenDaylight et Ryu prennent en charge le dernier protocole OF1.5, c'est l'existence de fonctionnalités optionnelles dans la spécification OpenFlow qui rend le niveau de support asynchrone entre les différents composants du Framework SDN. L'absence d'un protocole normalisé de gestion et de configuration des dispositifs Southbound contribue également au problème d'interopérabilité entre les composantes d'un réseau. La mise en œuvre de la QoS dans le SDN est plus facile et moins coûteuse que sur les réseaux traditionnels. Il nous a été possible d'implémenter des techniques complexes de QoS comme DiffServ dans un réseau, ce qui est jugé trop complexe à implémenter dans les réseaux actuels. De plus, la programmabilité et la flexibilité du SDN nous donnent l'opportunité de regarder au-delà des techniques de QoS disponibles comme DiffServ et le traffic shaping et de proposer une approche radicale et complète de la QoS qui ne serait jamais possible dans les réseaux actuels.

Nous avons testé les capacités QoS dans le SDN, en utilisant le contrôleur OpenDaylight. On a constaté que l'implémentation de chaque composant du SDN nécessite un certain niveau de support QoS.

Enfin les résultats obtenus ont été très satisfaisants, les objectifs recherchés ont été atteints :

- Donner des garanties de bande passante par flux.
- Prioriser correctement le trafic en fonction de la garantie, respectant ainsi les limites fixées.
- Autoriser le trafic excessif lorsque les ressources sont disponibles.

## Perspectives

Nous envisageons d'enrichir notre travail par :

- L’implémentation des solutions QoS citées dans le SD-WAN.

- L’implémentation de la solution QoS du traffic shaping en mode full SDN.

- Le SDN a une lacune dans le domaine de sécurité, il serait intéressant d’implémenter une technique de sécurité pour combler le problème du BYOD par exemple pour les NBI.

- Une implémentation potentielle de l'intelligence artificielle s'exécutant sur un contrôleur SDN qui peut garder la trace des ressources disponibles dans le réseau, peut prendre les requêtes des applications et configurer les paramètres QoS correspondants sur les périphériques réseau pour garantir les ressources demandées serait une approche intéressante. Bien qu'ils doivent d'abord s'attaquer à toutes les lacunes identifiées, ce n'est pas une chose irréalisable si des ressources adéquates sont affectées à cette tâche. De plus, de nombreuses recherches examinées dans le projet ont fourni la solution à une catégorie de la mise en œuvre de la QoS. Un travail qui peut exploiter toutes ces techniques et les combiner dans un système de QoS qui peut répondre à toutes les catégories de techniques de QoS serait une solution intéressante pour SDN. La standardisation du protocole de configuration des dispositifs (OF-CONFIG par ONF est un candidat possible) est quelque chose qui faciliterait le processus d'implémentation de tout système de QoS dans le SDN et qui devrait être implémenté aussi rapidement que possible. Cela pourrait accroître encore davantage le taux de recherche sur la qualité de service dans le SDN. De plus, à l'avenir, l'interopérabilité entre les différentes implémentations de composants SDN devrait être d'une importance primordiale pour l'ensemble de la communauté SDN, car un réseau réel comporterait un ensemble hétérogène de dispositifs et l'implémentation de différents paramètres QoS sur différents dispositifs qui ne coopèrent pas entre eux serait un obstacle majeur à la croissance du SDN.

# Annexes 1 : Installation de l'OpenDaylight

## Procédure d'installation d'Opendaylight

**Etape 1 :** Installer n'importe quelle machine virtuelle (VMware ou virtual box).

**Etape 2 :** Créer une machine virtuelle avec Ubuntu comme système d'exploitation sous le nom d'OpenDaylight et télécharger la dernière version d'OpenDaylight qui s'y trouve. (Dans notre cas, nous avons téléchargé Ubuntu 14.04 et la version béryllium d'OpenDaylight). Télécharger-le sur www.opendaylight.org/downloads

**Etape 3 :** Construire deux machines séparées d'Ubuntu et de mininet sur VMware (vous pouver télécharger mininet sur www.mininet.org) comme indiqué dans la Figure 38.

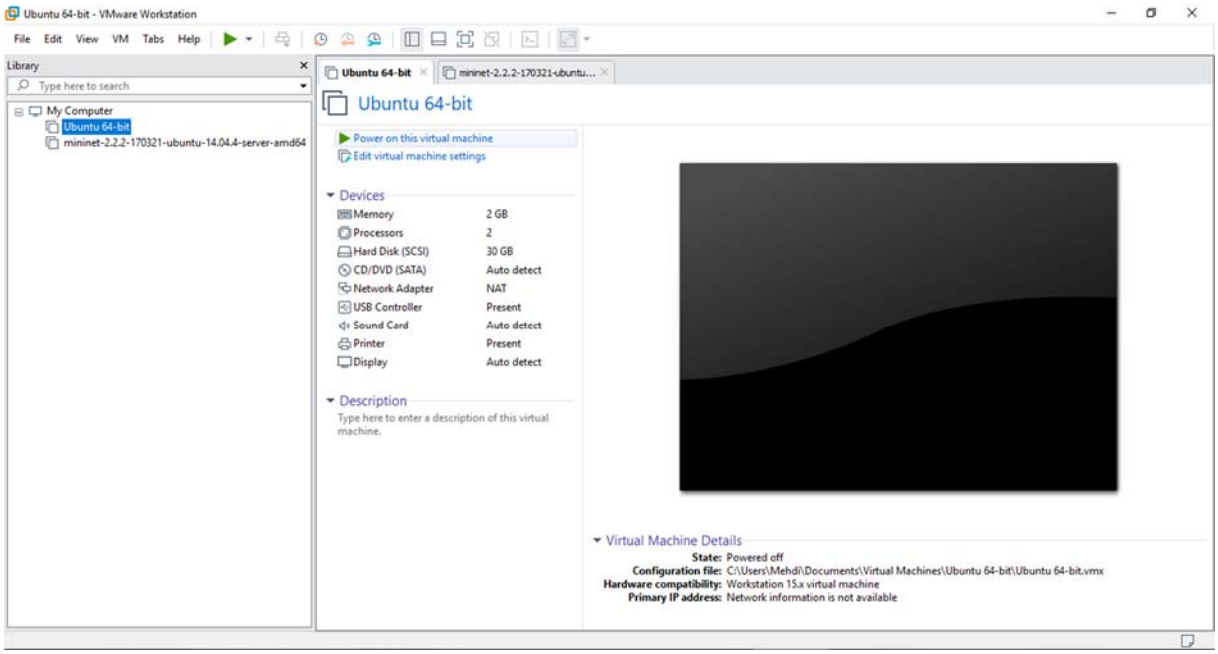



Figure 38 : Création de deux machines sur VMware Workstation

**Etape 4 :** Le contrôleur OpenDaylight SDN est un programme écrit sous le langage Java donc installer l'environnement d'exécution Java avec la commande suivante :

- apt-get install openjdk-8-jdk.
- apt-get install openjdk-8-jre.

**Etape 5 :** Extraire le fichier zip de la version publiée d'OpenDaylight que vous avez télécharger à l'étape 2. (Nous avons donné les noms odl.zip et odl aux fichiers extraits respectivement) comme indiqué dans la Figure 39.

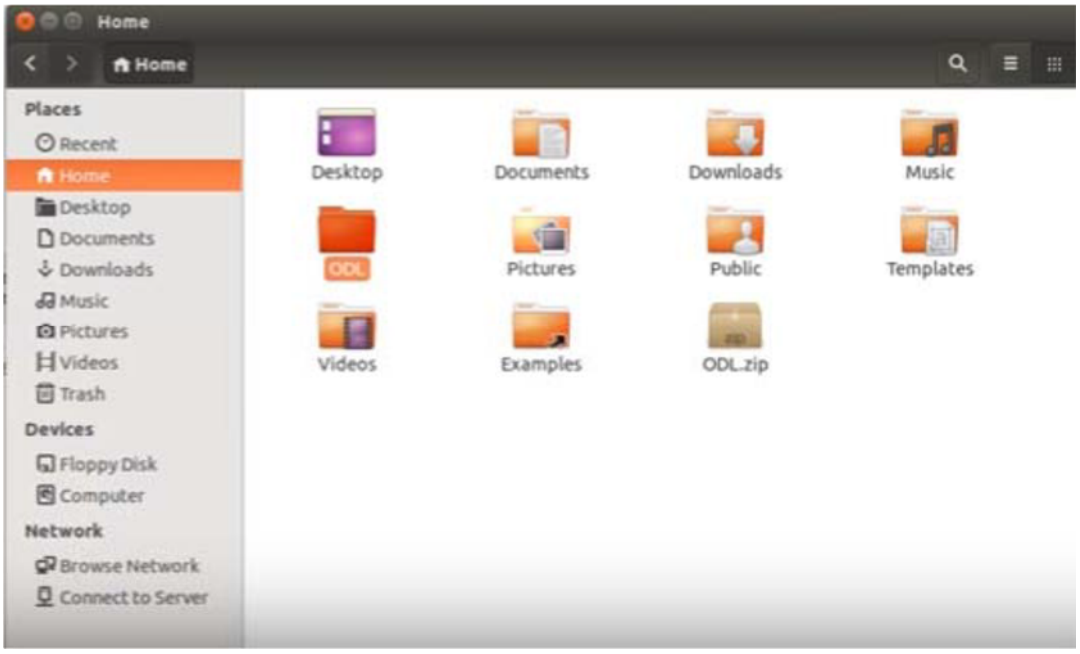



Figure 39 : Création du dossier après extraction

Lancer maintenant la commande suivante sur le terminal Ubuntu

- cd ODL/bin
- ./karaf -of13

Après avoir exécuté les commandes mentionnées ci-dessus, l'écran suivant apparaîtra comme indiqué dans la Figure 40.

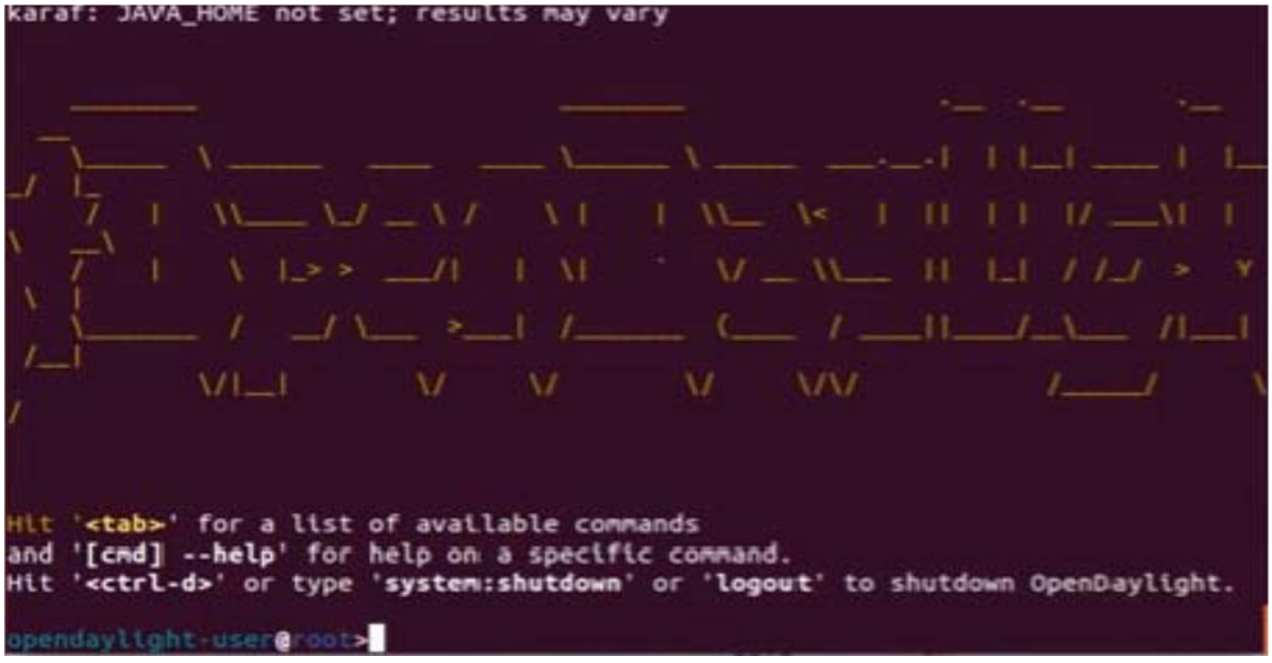



Figure 40 : Contrôleur Opendaylight lancé

**Etape 6 :** Ouvrer le terminal n°2 dans Ubuntu et exécuter la commande suivante

- Sudo apt-get-install nmap
- Hôte local nmap

**Nmap** (Network Mapper) écrit à l'origine par Gordon Lyon est un scanner de sécurité utiliser pour découvrir les hôtes et les services sur un réseau informatique et enfin construire une "carte" du réseau, d'où son nom nmap. Pour atteindre son but, l'envoi de flux spécialement conçus par nmap au nœud(s) visé(s) [94].

**Etape 7 :** Installer les fonctionnalités à l'Opendaylight dans le terminal no.1 où les fonctionnalités karaf ont été lancer en utilisant les commandes suivantes :

- feature : install odl-l2switch-switch-ui
- feature : install odl-dlux-all (OpenDaylight n'a aucune fonctionnalité installée par défaut.

Vous devrez utiliser la console Karaf pour installer les fonctionnalités requises par DLUX) [95].

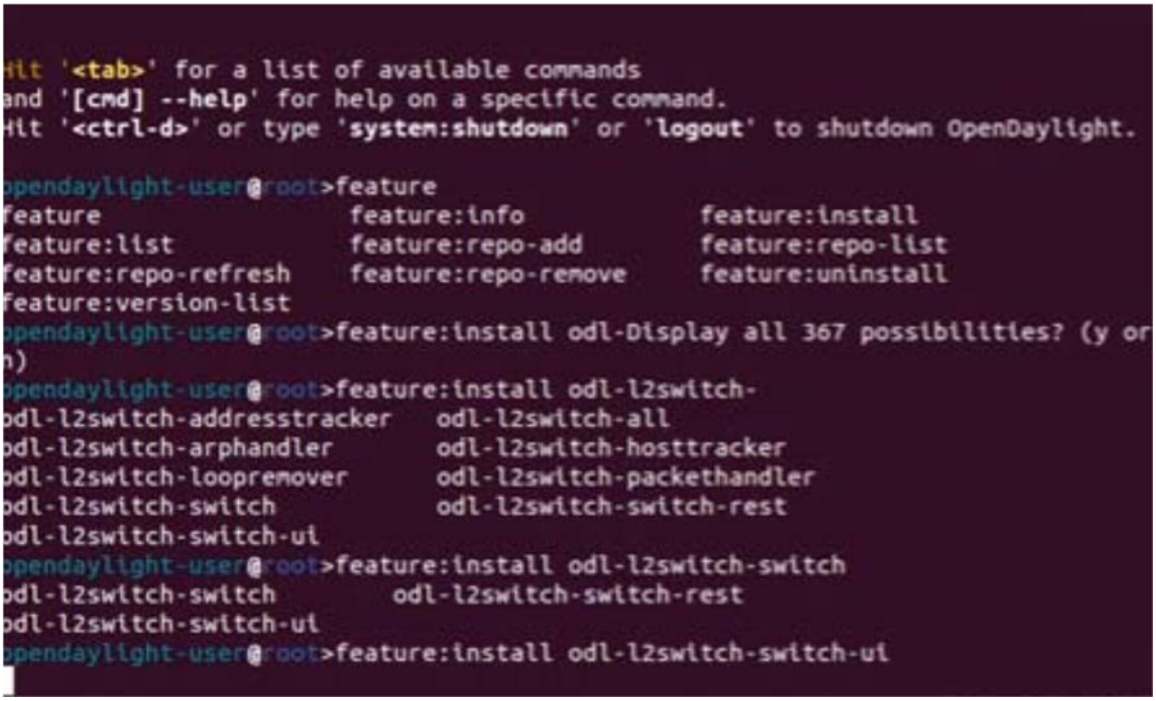



Figure 41 : ajouter des fonctionnalités au contrôleur

Dans le terminale n°2, exécuter la commande « ifconfig » pour avoir l'adresse ip de la machine mininet.

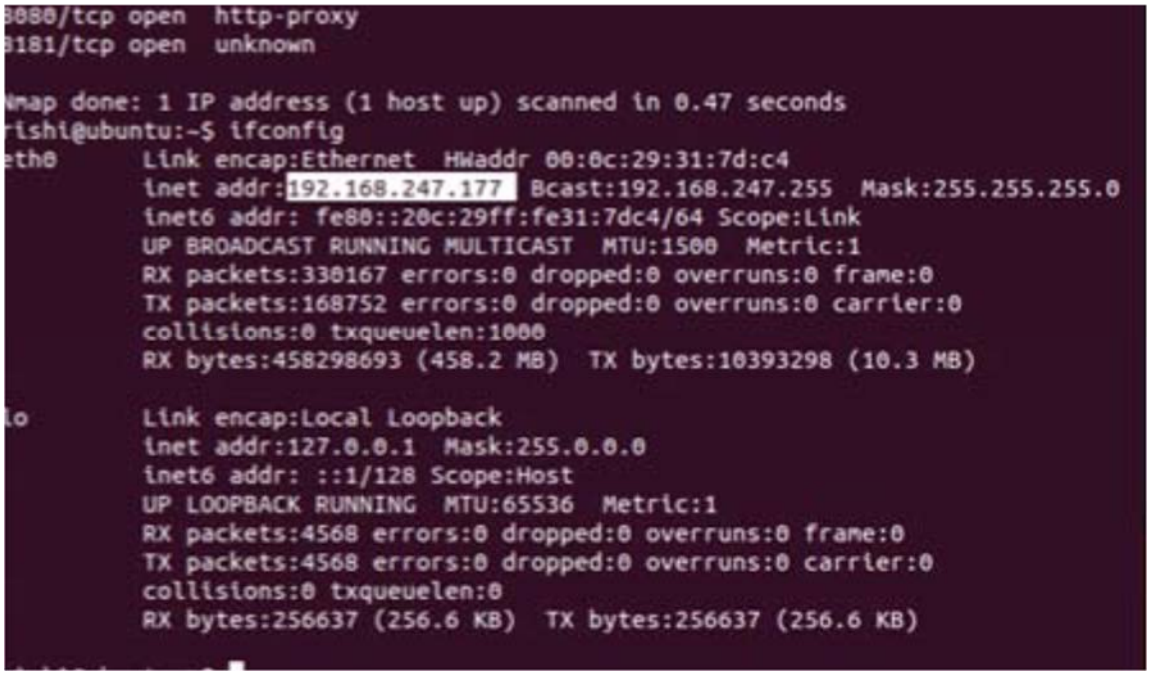



Figure 42 : Adresse IP de la machine mininet

Maintenant utiliser l'URL 192.168.247.177: 8181/index.html sur n'importe quel web browser comme indiqué sur la Figure 43.

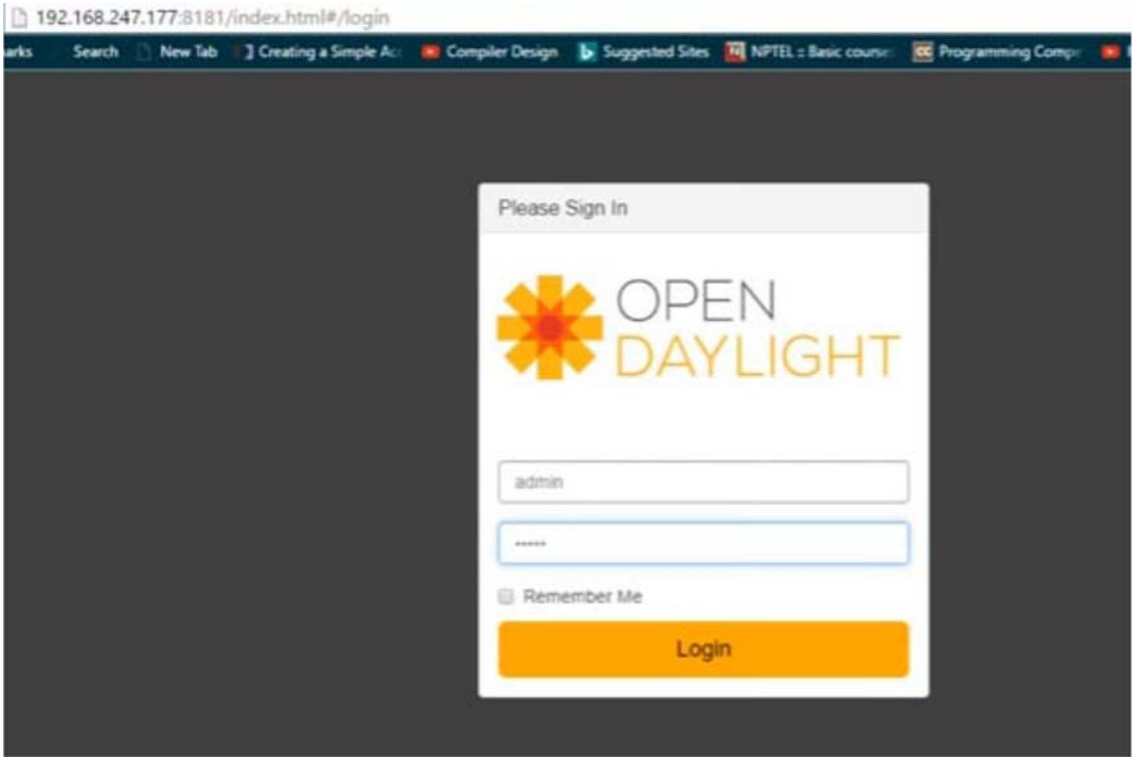



Figure 43 : Interface web représentant l'accueil du contrôleur OpenDaylight

Maintenant il faut utiliser le nom d'utilisateur et mot de passe du contrôleur Opendaylight par défaut sont :

- Nom d'utilisateur : admin
- Mot de passe : admin

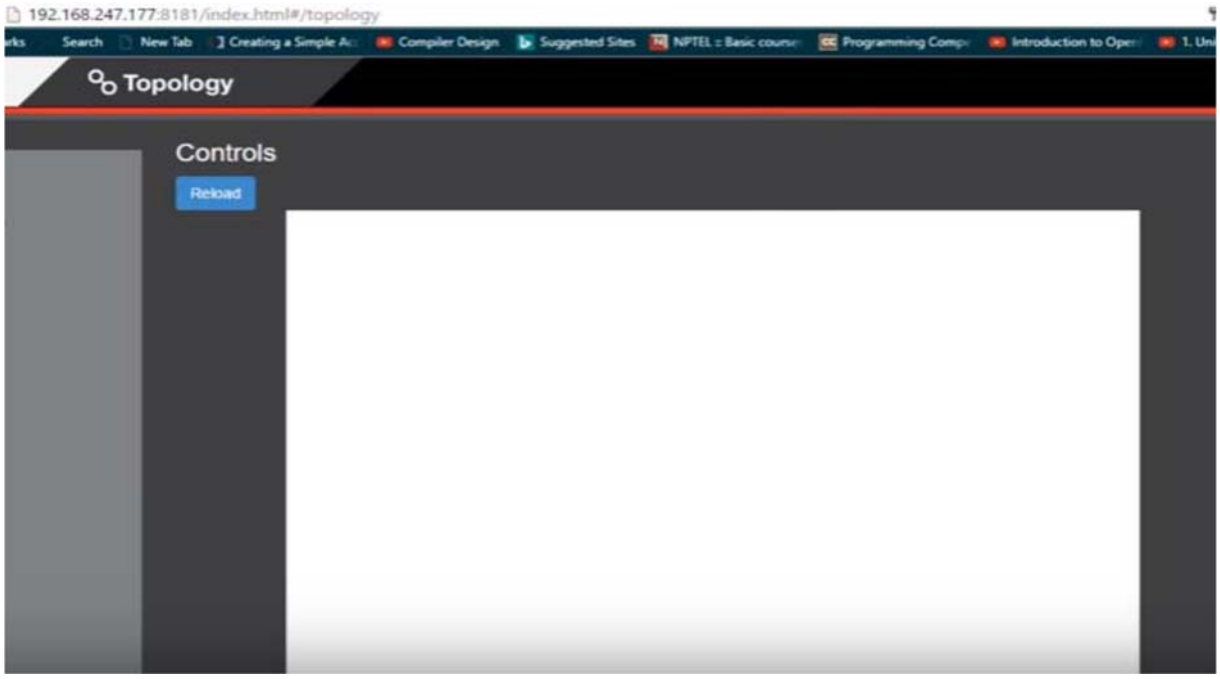



Figure 44 : Connexion à l'interface web du contrôleur Opendaylight

**Etape 8 :** Ouvrer la deuxième machine (Mininet) sur VMware et entrer le mot nom d'utilisateur et mot de passe par défaut :

- Nom d'utilisateur : mininet
- Mot de passe : mininet

Maintenant sur le terminale n°2 de la machine Ubuntu lancer la commande suivante :

ssh-X mininet@192.168.247.158 (c'est notre @ip de la machine Mininet que nous avons obtenu en utilisant la commande ifconfig).

# Annexes 2 : Présentation de l'organisme d'accueil

## 1. Organisation de Sonatrach

Le nouveau schéma d'organisation de la macrostructure de SONATRACH s'inscrit dans le cadre de l'évolution de l'environnement aussi bien interne qu'externe qui exige de l'entreprise d'adapter son schéma d'organisation et son mode de gestion pour faire face aux défis, notamment ceux inscrits dans son plan à moyen terme, à savoir l'augmentation du niveau de la production et des réserves dans l'amont et la réalisation des projets de raffinage et de pétrochimie dans l'aval.

L'adaptation de l'organisation de la modernisation du mode de gestion de l'entreprise figurent au rang des priorités de la Direction Générale et ce, pour répondre aux besoins du marché national en constante croissance et soutenir la position de SONATRACH sur les marchés pétroliers et gaziers.

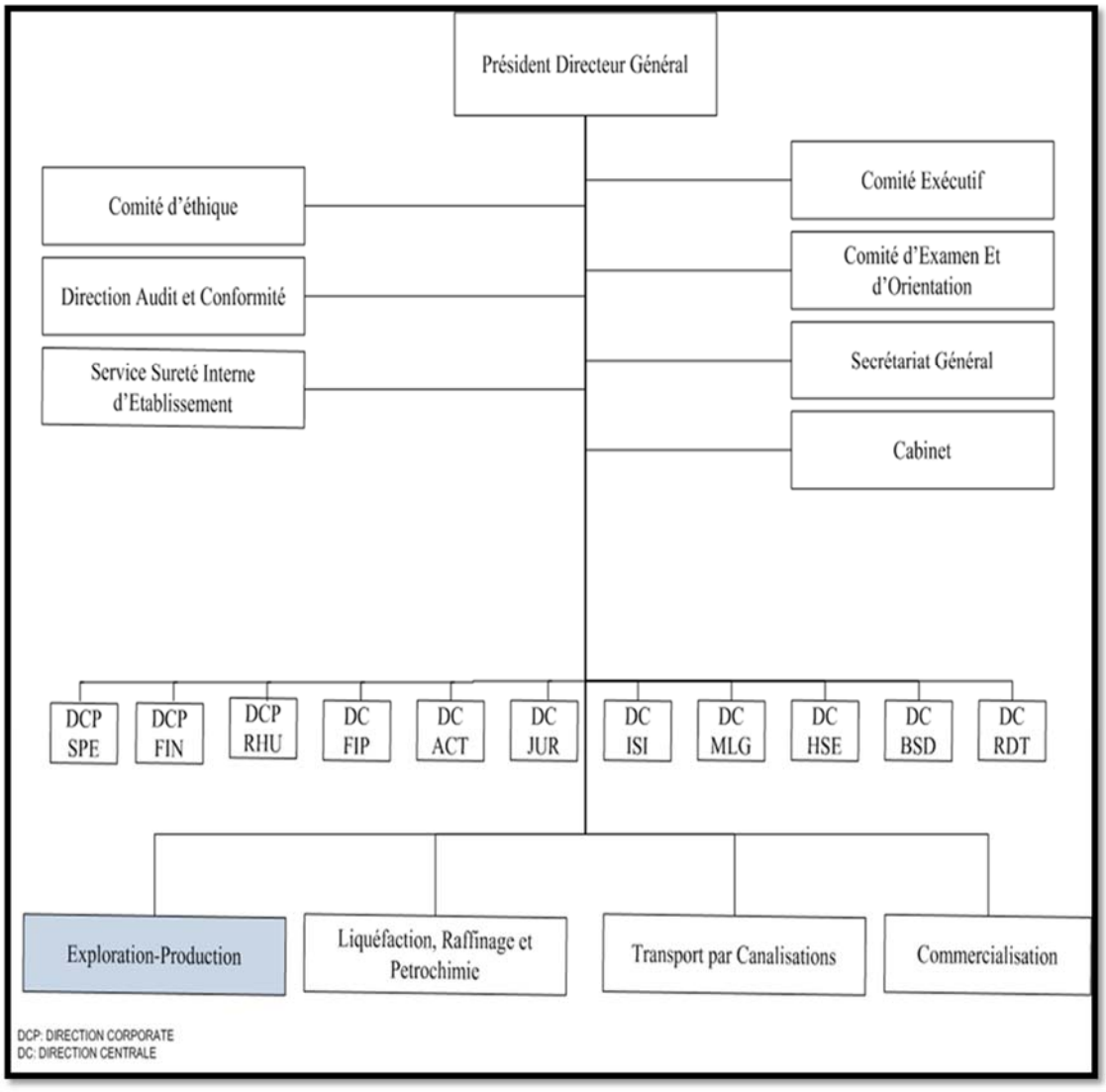



Figure 45 : Organigramme de la Sonatrach

# 2. Présentation de la division production

## 2.1. Organisation de la division production

La division production (SH/DP) est composée d'un siège regroupant des directions à Alger et de dix (10) régions réparties à travers le sud du pays.

Elle a pour mission de développement de l'exploitation des gisements d'hydrocarbures ainsi que l'optimisation de la production, par l'utilisation des méthodes efficaces de récupération d'entretien et de conservation des réserves en place.

La division production à sa charge l'exploitation de 1925 puits producteurs d'huile, 400 puits producteurs de gaz naturel, 173 puits injecteurs d'eau et 200 puits injecteurs de gaz.

Au titre de l'année 1991, la production brute de la « Sonatrach », tous produits confondus, a atteint 160.4 millions de TEP (tonnes équivalent pétrole) dont 54.6 TEP (de Gaz) ont été réinjectés dans les gisements. La part du gaz naturel liquéfié (GNL), représente 27.71% des exportations des hydrocarbures, La production du (GNL) a atteint 31.9 millions de mètre cubes de gaz liquides.

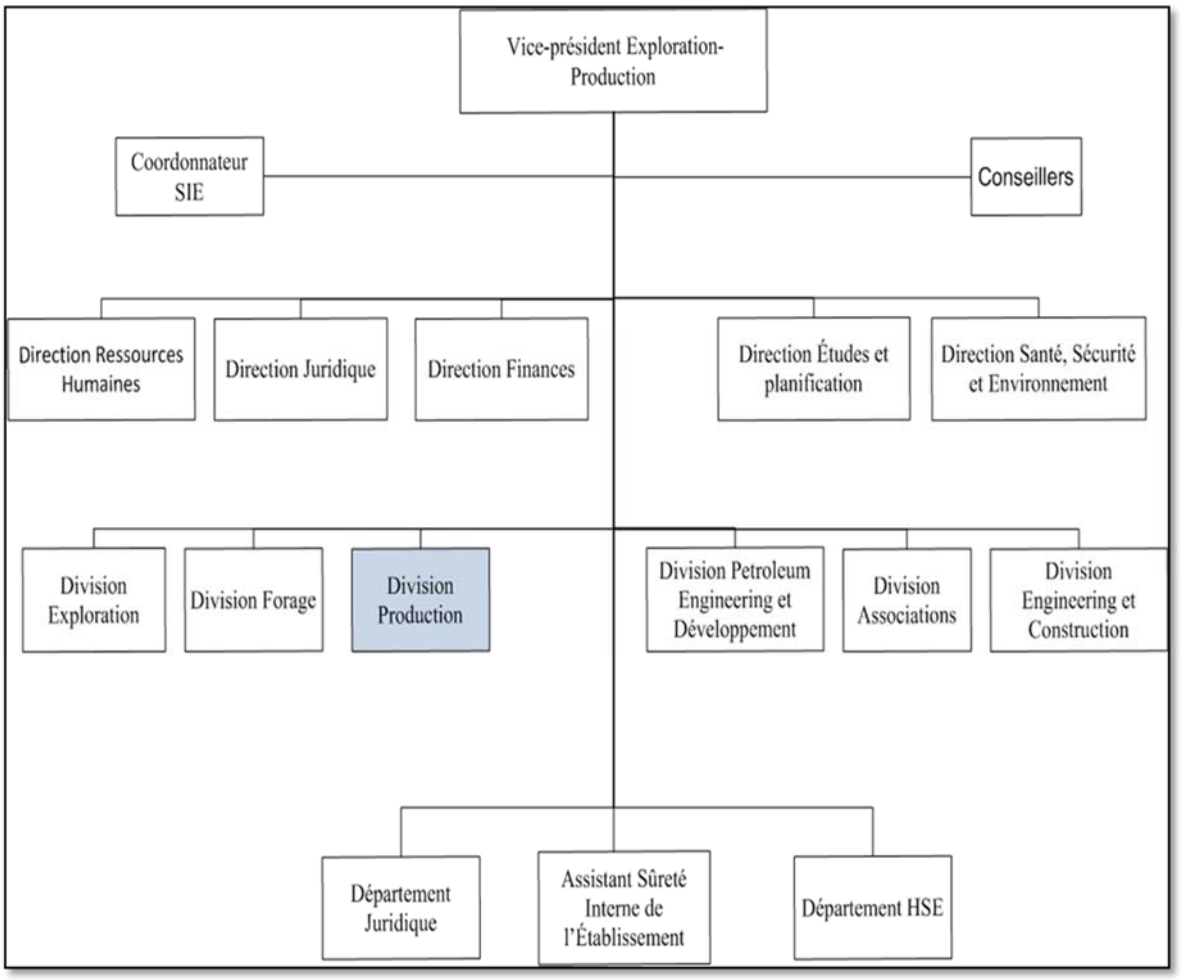



Figure 46 : Organigramme de l'activité Amont

La division production emploi actuellement plus de 16.000 agent répartis entre le siège et les régions et structures de : Sept (07) directions :

- Direction Opérations.
- Direction Réalisation.
- Direction Finance et Comptabilité.
- Direction Gestion du Personnel.
- Direction Approvisionnement et Transport.
- Direction Moyens Généraux.
- Direction Informatique.

Et dix (10) directions régionales qui sont :

- Hassi Messaoud (HMD).
- Hassiremel (HRM).
- HaoudBerkaoui (HBK).
- GassiTouil (GTL).
- Ohanet (OHT).
- Stah (STH).
- InAmenas (INA).
- Rhoudes El Nouss (RNS).
- Tin FouyéTabankort (TFT).
- Rourd El Baguel (REB).
- Hors régions (OeudGuetteni, Djebel Onk,BBK,TGT ).

## a. Les directions régionales ont pour objet

- La réalisation et le servi des programmes détaillés de production et d'expédition.
- La conduite des opérations de production en conformité avec les règles et consigne de sécurité.
- La valorisation et l'optimisation du gisement et des installations surface.
- La réalisation des travaux de développement et la prestation de toute assistance nécessaire, afin d'obtenir la mise de production rapide des puits des installations surface.

## b. Activité de la division production « DP »

L'activité de la division production consiste :

- Au développement et à l'exploitation des gisements d'hydrocarbures, situés dans leur quasi-totalité dans le sud-est Algérien et qui sont actuellement au nombre de 66 gisements.

- A la production d'hydrocarbures liquides et gazeux (pétrole brut, condensat GPL et GAZ).
- L'exploitation et maintenance des installations et équipements de production d'hydrocarbures de pression des gisements.
- A la gestion et l'exploitation des raffineries de Hassi-messaoud et d'Ain amines.

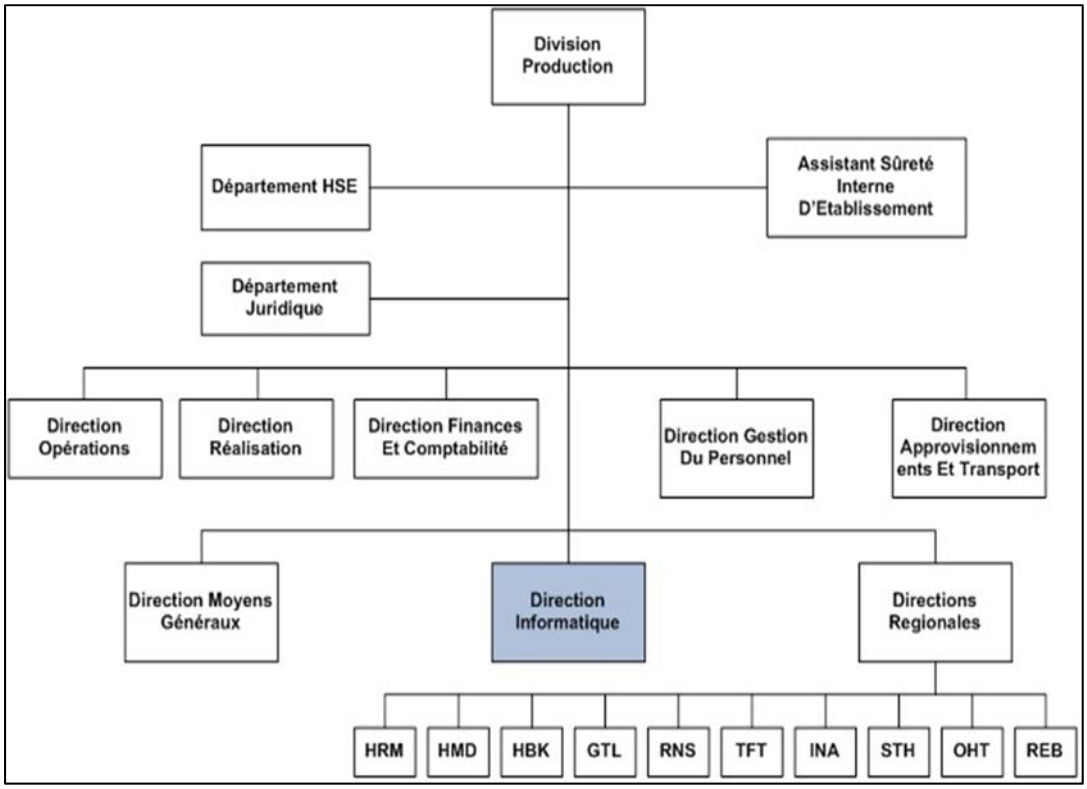



Figure 47 : Organigramme de la Division Production

## c. Direction Informatique

La direction informatique est dotée d'un potentiel scientifique de haut niveau, un nombre appréciable de post-gradués, d'ingénieurs, de licenciés et de techniciens, elle dispose aussi d'un matériel très considérable, qui peut répondre d'une façon satisfaisante aux exigences de la SONATRACH La direction informatique a pour missions :

- La définition de la politique de la division production en matière de développement informatique conformément à la stratégie et la politique de la Société.
- L'étude et l'évaluation des besoins en matière de système de traitement de l'information.

- L'élaboration et la mise en œuvre des plans de développement des TICC de la Division production.
- L'administration du réseau informatique.
- La standardisation des normes en matière d'acquisition des équipements de renouvellement du parc informatique.
- L'élaboration et la mise en œuvre du plan informatique de la division production.
- La prestation de service, à toutes les structures de la division production, en matière de traitement de l'information pour les besoins scientifiques et techniques de la division production.
- La construction et la tenue à jour d'une banque de données techniques, économiques et financières de la division production.

La gestion et l'exploitation des moyens informatiques du siège de la division production Elle se compose de quatre départements qui sont :

- Département maintenance.
- Département développement.
- Département Système et exploitation.
- Département réseau et télécommunication.

# Bibliographie


[1] N. McKeown *et al.*, « OpenFlow: Enabling innovation in campus networks », *Comput. Commun. Rev.*, vol. 38, p. 69□74, avr. 2008.

[2] F. A. Kuipers, « Quality of Service Routing in the Internet. Theory, Complexity and Algorithms », 2004.

[3] R. Braden, D. Clark, et S. Shenker, « Integrated Services in the Internet Architecture: an Overview », janv. 1994.

[4] A. P. Bianzino, C. Chaudet, D. Rossi, et J. Rougier, « A Survey of Green Networking Research », *IEEE Commun. Surv. Tutor.*, vol. 14, nº 1, p. 3□20, First 2012.

[5] S. Ben Chahed, « Mise en oeuvre des aspects de gestion des réseaux définis par logiciels (réseaux SDN) », masters, École Polytechnique de Montréal, 2015.

[6] K. Greene, « TR10: Software-Defined Networking », *MIT Technology Review*. [En ligne]. Disponible sur: http://www2.technologyreview.com/news/412194/tr10-software-defined-networking/. [Consulté le: 27-août-2019].

[7] N. Gude *et al.*, « NOX: Towards an Operating System for Networks », *SIGCOMM Comput Commun Rev*, vol. 38, nº 3, p. 105–110, juill. 2008.

[8] H. Jamjoom, D. Williams, et U. Sharma, « Don't call them middleboxes, call them middlepipes », *HotSDN 2014 - Proc. ACM SIGCOMM 2014 Workshop Hot Top. Softw. Defin. Netw.*, août 2014.

[9] R. Masoudi et A. Ghaffari, « Software defined networks: A survey », *J. Netw. Comput. Appl.*, vol. 67, p. 1□25, mai 2016.

[10] « The Road to SDN - ACM Queue ». [En ligne]. Disponible sur: https://queue.acm.org/detail.cfm?id=2560327. [Consulté le: 27-août-2019].

[11] D. B. Rawat et S. R. Reddy, « Software Defined Networking Architecture, Security and Energy Efficiency: A Survey », *IEEE Commun. Surv. Tutor.*, vol. 19, nº 1, p. 325□346, Firstquarter 2017.

[12] M. Yu, J. Rexford, M. J. Freedman, et J. Wang, « Scalable Flow-based Networking with DIFANE », in *Proceedings of the ACM SIGCOMM 2010 Conference*, New York, NY, USA, 2010, p. 351–362.

[13] Y. Ganjali et A. Tootoonchian, « HyperFlow: A Distributed Control Plane for OpenFlow », in *INM/WREN*, 2010.

[14] T. Eugene et Z. Cai, « Maestro: achieving scalability and coordination in centralizaed network control plane », 2012.

[15] B. P. Rimal, A. Jukan, D. Katsaros, et Y. Goeleven, « Architectural Requirements for Cloud Computing Systems: An Enterprise Cloud Approach », *J. Grid Comput.*, vol. 9, nº 1, p. 3□26, mars 2011.

[16] M. Armbrust *et al.*, « A View of Cloud Computing », *Commun ACM*, vol. 53, p. 50□58, avr. 2010.

[17] S. Scott-Hayward, G. O'Callaghan, et S. Sezer, « Sdn Security: A Survey », in *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*, 2013, p. 1□7.

[18] R. Klöti, V. Kotronis, et P. Smith, « OpenFlow: A security analysis », in *2013 21st IEEE International Conference on Network Protocols (ICNP)*, 2013, p. 1□6.

[19] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, et R. Smeliansky, « Advanced study of SDN/OpenFlow controllers », présenté à ACM Proc. CEE-SECR, 2013.

[20] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, et P. Tran-Gia, « Modeling and performance evaluation of an OpenFlow architecture », in *2011 23rd International Teletraffic Congress (ITC)*, 2011, p. 1□7.

[21] « Cbench Controller Benchmarker | BibSonomy ». [En ligne]. Disponible sur: https://www.bibsonomy.org/bibtex/111bc70e4d8012f1196123dbc3e310b7. [Consulté le: 27-août-2019].
[22] M. Jarschel, F. Lehrieder, Z. Magyari, et R. Pries, « A Flexible OpenFlow-Controller Benchmark », in *2012 European Workshop on Software Defined Networking*, 2012, p. 48□53.
[23] D. Tipper, « Resilient Network Design: Challenges and Future Directions », *Telecommun. Syst.*, vol. 56, p. 5□16, mai 2013.
[24] D. B. Hoang, « Software Defined Networking ? Shaping up for the next disruptive step? », *Aust. J. Telecommun. Digit. Econ.*, vol. 3, nº 4, nov. 2015.
[25] « OpenFlow - mininet ». [En ligne]. Disponible sur: https://sites.google.com/site/mininetbasic/open-flow/openflow. [Consulté le: 01-oct-2019].
[26] B. Pfaff et B. Davie, « The Open vSwitch Database Management Protocol ». [En ligne]. Disponible sur: https://tools.ietf.org/html/rfc7047. [Consulté le: 27-août-2019].
[27] A. Shaghaghi, M. A. Kaafar, R. Buyya, et S. Jha, « Software-Defined Network (SDN) Data Plane Security: Issues, Solutions and Future Directions », *ArXiv180400262 Cs*, avr. 2018.
[28] « OpenFlow Switch Specification - PDF ». [En ligne]. Disponible sur: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.
[29] « WHITEPAPER. PicOS Overview - PDF ». [En ligne]. Disponible sur: https://docplayer.net/75660786-Whitepaper-picos-overview.html. [Consulté le: 27-août-2019].
[30] « Open_vSwitch database schema - PDF ». [En ligne]. Disponible sur: http://www.openvswitch.org/support/dist-docs/ovs-vswitchd.conf.db.5.html. [Consulté le: 28-août-2019].
[31] J. Dix, « Clarifying the role of software-defined networking northbound APIs », *Network World*, 02-mai-2013. [En ligne]. Disponible sur: https://www.networkworld.com/article/2165901/clarifying-the-role-of-software-defined-networking-northbound-apis.html. [Consulté le: 28-août-2019].
[32] « The Northbound SDN API - A Big Little Problem », *NetworkStatic | Brent Salisbury's Blog*, 10-juin-2012. .
[33] N. Foster *et al.*, « Frenetic: A Network Programming Language », présenté à Sigplan Notices - SIGPLAN, 2011, vol. 46, p. 279□291.
[34] C. Monsanto, N. Foster, R. Harrison, et D. Walker, « A Compiler and Run-time System for Network Programming Languages », in *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, NY, USA, 2012, p. 217–230.
[35] A. Voellmy et P. Hudak, « Nettle: Taking the Sting Out of Programming Network Routers », 2011, p. 235□249.
[36] M. Monaco, O. Michel, et E. Keller, « Applying Operating System Principles to SDN Controller Design », *Proc. Twelfth ACM Workshop Hot Top. Netw. - HotNets-XII*, p. 1□7, 2013.
[37] « SDN architecture - PDF ». [En ligne]. Disponible sur: https://www.opennetworking.org/wp-content/uploads/2013/02/TR_SDN_ARCH_1.0_06062014.pdf.
[38] V. K. Gurbani, M. Scharf, T. V. Lakshman, V. Hilt, et E. Marocco, « Abstracting network state in Software Defined Networks (SDN) for rendezvous services », in *2012 IEEE International Conference on Communications (ICC)*, 2012, p. 6627□6632.
[39] « The "Platform as a Service"Model for Networking ». [En ligne]. Disponible sur: https://pdfs.semanticscholar.org/9a55/023fce48938fbdb1f37f652f9fc4cd5dd1b3.pdf.
[40] « ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN) - ACM SIGCOMM 2014 ». [En ligne]. Disponible sur: https://conferences.sigcomm.org/sigcomm/2014/hotsdn.php. [Consulté le: 28-août-2019].
[41] K. Phemius, M. Bouet, et J. Leguay, « DISCO: Distributed multi-domain SDN controllers », in *2014 IEEE Network Operations and Management Symposium (NOMS)*, 2014, p. 1□4.

[42] B. Lee, S. H. Park, J. Shin, et S. Yang, « IRIS: The Openflow-based Recursive SDN controller », in *16th International Conference on Advanced Communication Technology*, 2014, p. 1227□1231.
[43] Z. Cai, A. L. Cox, et T. S. E. Ng, « Maestro: A System for Scalable OpenFlow Control », déc. 2010.
[44] J. Medved, R. Varga, A. Tkacik, et K. Gray, « OpenDaylight: Towards a Model-Driven SDN Controller architecture », in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, 2014, p. 1□6.
[45] S. Kaur, J. Singh, et N. Singh Ghumman, « Network Programmability Using POX Controller », 2014.
[46] H. Akçay et D. Yiltas, « Web-Based User Interface for the Floodlight SDN Controller », *Int. J. Adv. Netw. Appl.*, vol. 8, p. 3175□3180, avr. 2017.
[47] C. El Khalfi, A. El Qadi, et H. Bennis, « A Comparative Study of Software Defined Networks Controllers », 2017, p. 1□5.
[48] M. Brandt, R. Khondoker, R. Marx, et K. Bayarou, « Security analysis of software defined networking protocols - OpenFlow, OF-Config and OVSDB », 2014.
[49] O. Dugeon, « Architectures des réseaux pour le contrôle de la QoS », 18-déc-2008. [En ligne]. Disponible sur: http://oatao.univ-toulouse.fr/2264/. [Consulté le: 28-août-2019].
[50] S. P. Mirashe et N. V. Kalyankar, « Quality of Service with Bandwidth », *ArXiv10034073 Cs*, mars 2010.
[51] B. Zhang, T. S. E. Ng, A. Nandi, R. H. Riedi, P. Druschel, et G. Wang, « Measurement-Based Analysis, Modeling, and Synthesis of the Internet Delay Space », *IEEEACM Trans. Netw.*, vol. 18, nº 1, p. 229□242, févr. 2010.
[52] W. Sugeng, J. E. Istiyanto, K. Mustofa, et A. Ashari, « THE IMPACT OF QoS CHANGES TOWARDS NETWORK PERFORMANCE », *Int. J. Comput. Netw. Commun. Secur.*, vol. 3, p. 48□53, févr. 2015.
[53] « What is QoS (quality of service) ? - Definition from WhatIs.com », *SearchUnifiedCommunications*. [En ligne]. Disponible sur: https://searchunifiedcommunications.techtarget.com/definition/QoS-Quality-of-Service. [Consulté le: 28-août-2019].
[54] L. Zhang, S. Berson, S. Herzog, et S. Jamin, « Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification ». [En ligne]. Disponible sur: https://tools.ietf.org/html/rfc2205. [Consulté le: 28-août-2019].
[55] « Resource ReSerVation Protocol (RSVP) ». [En ligne]. Disponible sur: https://www.cl.cam.ac.uk/~jac22/books/mm/book/node52.html. [Consulté le: 28-sept-2019].
[56] M. Villapol et J. Billington, « Internet Service Quality: A Survey and Comparison of the IETF Approaches », vol. 50, déc. 2000.
[57] E. Davies, M. A. Carlson, W. Weiss, D. Black, S. Blake, et Z. Wang, « An Architecture for Differentiated Services ». [En ligne]. Disponible sur: https://tools.ietf.org/html/rfc2475. [Consulté le: 28-août-2019].
[58] « DiffServ -- The Scalable End-to-End QoS Model [QoS Signaling] - Cisco Systems ». [En ligne]. Disponible sur: https://www.cisco.com/en/US/technologies/tk543/tk766/technologies_white_paper09186a00800a3e2f.html. [Consulté le: 28-sept-2019].
[59] V. Firoiu, W. Courtney, B. Davie, et A. Charny, « An Expedited Forwarding PHB (Per-Hop Behavior) ». [En ligne]. Disponible sur: https://tools.ietf.org/html/rfc3246. [Consulté le: 28-août-2019].
[60] J. Wroclawski, W. Weiss, J. Heinanen, et F. Baker, « Assured Forwarding PHB Group ». [En ligne]. Disponible sur: https://tools.ietf.org/html/rfc2597. [Consulté le: 28-août-2019].

[61] Y. Liu, G. Lu, W. Zhang, F. Cai, et Q. Kong, « A DSCP-Based Method of QoS Class Mapping between WLAN and EPS Network », in *Algorithms and Architectures for Parallel Processing*, 2014, p. 204□213.
[62] « Comparing Traffic Policing and Traffic Shaping for Bandwidth Limiting - Cisco ». [En ligne]. Disponible sur: https://www.cisco.com/c/en/us/support/docs/quality-of-service-qos/qos-policing/19645-policevsshape.html. [Consulté le: 28-sept-2019].
[63] « NE20E-S V800R010C10SPC500 Feature Description - QoS 01 - Huawei ». [En ligne]. Disponible sur: https://support.huawei.com/enterprise/en/doc/EDOC1100055126/b48def52/traffic-shaping. [Consulté le: 10-sept-2019].
[64] « IP Precedence and DSCP Values », *NetworkLessons.com*, 23-sept-2014. .
[65] « Understanding Differentiated Services (TOS) field in Internet Protocol Header », *slashroot.in*. [En ligne]. Disponible sur: https://www.slashroot.in/understanding-differentiated-services-tos-field-internet-protocol-header. [Consulté le: 28-août-2019].
[66] K. Nichols, D. L. Black, S. Blake, et F. Baker, « Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers ». [En ligne]. Disponible sur: https://tools.ietf.org/html/rfc2474. [Consulté le: 28-août-2019].
[67] C. Gbaguidi, H. J. Einsiedler, P. Hurley, W. Almesberger, et J.-P. Hubaux, « A Survey of Differentiated Services Proposals for the Internet », août 1998.
[68] « IP Datagram, ip options, tos field ». [En ligne]. Disponible sur: http://www.rhyshaden.com/ipdgram.htm. [Consulté le: 28-août-2019].
[69] S. Tomovic, N. Prasad, et I. Radusinovic, « SDN control framework for QoS provisioning », in *2014 22nd Telecommunications Forum Telfor (TELFOR)*, 2014, p. 111□114.
[70] « tc(8): show/change traffic control settings - Linux man page ». [En ligne]. Disponible sur: https://linux.die.net/man/8/tc. [Consulté le: 28-août-2019].
[71] C. Lee et Y. Kim, « QoS-aware hierarchical token bucket (QHTB) queuing disciplines for QoS-guaranteed Diffserv provisioning with optimized bandwidth utilization and priority-based preemption », in *The International Conference on Information Networking 2013 (ICOIN)*, 2013, p. 351□358.
[72] « tc-htb - Hierarchy Token Bucket - Linux Man Pages (8) ». [En ligne]. Disponible sur: //www.systutorials.com/docs/linux/man/docs/linux/man/8-tc-htb/. [Consulté le: 28-août-2019].
[73] « HTB manual - user guide ». [En ligne]. Disponible sur: http://luxik.cdi.cz/~devik/qos/htb/manual/userg.htm. [Consulté le: 28-août-2019].
[74] « Metering in OVS datapath - PDF ». [En ligne]. Disponible sur: http://workshop.netfilter.org/2017/wiki/images/d/db/Nfws-ovs-metering.pdf.
[75] J. Yan, H. Zhang, Q. Shuai, B. Liu, et X. Guo, « HiQoS: An SDN-based multipath QoS solution », *China Commun.*, vol. 12, nº 5, p. 123□133, mai 2015.
[76] S. Sharma *et al.*, « Implementing Quality of Service for the Software Defined Networking Enabled Future Internet », in *2014 Third European Workshop on Software Defined Networks*, 2014, p. 49□54.
[77] M. F. Bari, S. R. Chowdhury, R. Ahmed, et R. Boutaba, « PolicyCop: An Autonomic QoS Policy Enforcement Framework for Software Defined Networks », in *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*, 2013, p. 1□7.
[78] « Traffic shaping with OVS and SDN - PDF ». [En ligne]. Disponible sur: https://docplayer.net/52092075-Traffic-shaping-with-ovs-and-sdn.html. [Consulté le: 28-août-2019].
[79] M. Seddiki *et al.*, « FlowQoS: QoS for the rest of us », *HotSDN 2014 - Proc. ACM SIGCOMM 2014 Workshop Hot Top. Softw. Defin. Netw.*, août 2014.
[80] R. Khondoker, A. Zaalouk, R. Marx, et K. Bayarou, « Feature-based comparison and selection of Software Defined Networking (SDN) controllers », in *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*, 2014, p. 1□7.

[81] M. T.BAH, A. Azzouni, M. T. Nguyen, et G. Pujolle, « Topology Discovery Performance Evaluation of OpenDaylight and ONOS Controllers », in *2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, 2019, p. 285□291.
[82] L. Zhu, M. M. Karim, K. Sharif, F. Li, X. Du, et M. Guizani, « SDN Controllers: Benchmarking & Performance Evaluation », *ArXiv190204491 Cs*, févr. 2019.
[83] G. Tank, A. Dixit, A. Vellanki, et D. Annapurna, « Software-Defined Networking-The New Norm for Networks », 2012.
[84] « What is the Brocade SDN Controller? - Defined », *SDxCentral*. [En ligne]. Disponible sur: https://www.sdxcentral.com/networking/sdn/definitions/brocade-vyatta-controller/. [Consulté le: 28-août-2019].
[85] S. Badotra et J. Singh, « Open Daylight as a Controller for Software Defined Networking », *Int. J. Adv. Comput. Res.*, vol. 8, mai 2017.
[86] impro, « impro's Software Defined World ». [En ligne]. Disponible sur: http://improf.egloos.com. [Consulté le: 28-août-2019].
[87] « Roadmap », *OpenDaylight*. .
[88] « Beryllium », *OpenDaylight*. .
[89] « Ryu Certification ». [En ligne]. Disponible sur: http://osrg.github.io/ryu/certification.html. [Consulté le: 28-août-2019].
[90] *Emulator for rapid prototyping of Software Defined Networks: mininet/mininet*. Mininet, 2019.
[91] F. Benamrane, M. Ben Mamoun, et B. Redouane, « Etude des Performances des Architectures du Plan de Contrôle des Réseaux 'Software-Defined Networks' », 2017.
[92] « Mininet: An Instant Virtual Network on your Laptop (or other PC) - Mininet ». [En ligne]. Disponible sur: http://mininet.org/. [Consulté le: 28-août-2019].
[93] « iPerf - The TCP, UDP and SCTP network bandwidth measurement tool ». [En ligne]. Disponible sur: https://iperf.fr/. [Consulté le: 28-août-2019].
[94] « Chapter 15. Nmap Reference Guide | Nmap Network Scanning ». [En ligne]. Disponible sur: https://nmap.org/book/man.html. [Consulté le: 28-août-2019].
[95] « Apache Karaf - The enterprise class platform ». [En ligne]. Disponible sur: https://karaf.apache.org/documentation.html. [Consulté le: 28-août-2019].