TABLE DES MATIERES

TABLE DES FIGURES

TABLE DES TABLEAUX

Chapitre 1 : Cadre théorique et méthodologique

1.1 Introduction générale

Avec les progrès continus des technologies de l'information, Internet est devenue une infrastructure complexe à grande échelle qui a changé les styles de travail, d'études et de vie des personnes. La complexité de l'Internet traditionnel rend sa configuration et sa gestion difficile pour les administrateurs de réseau lorsque l'on considère la dynamique du réseau et les divers besoins des applications. Il existe donc un besoin urgent de concevoir et de développer une nouvelle architecture de réseau capable de gérer le réseau de manière dynamique et flexible.

Le réseau programmable (Software Defined Networking) a été proposé pour surmonter les faiblesses du réseau traditionnel. En tant que nouveau paradigme de réseau, le SDN révolutionne la technologie de réseau en brisant l'idée fondamentale des réseaux traditionnels. Le SDN s'est développé comme un paradigme de réseau qui étend l'adoption des technologies de virtualisation des serveurs aux réseaux.

L'idée principale de SDN est de découpler le plan de contrôle et le plan de données. Les plans de contrôle sont fusionnés en une seule entité (ou plusieurs) nommée contrôleur. En tant que décideur, le contrôleur SDN est capable de fournir des interfaces de programme d'application (API) aux applications supérieures et de permettre aux opérateurs de déployer diverses stratégies réseau de manière flexible, en fonction des scénarios et des exigences des applications. Par conséquent, les contrôleurs jouent un rôle important dans les réseaux SDN.

Vue la grande importance de ce contrôleur et que les premiers architectures SDN appelée architecture initiale sont composées d'un seul contrôleur qui constitue le cerveau de l'ensemble du réseau et qui est aussi un point de défaillance unique, il est nécessaire de penser à d'autres architectures qui vont se constituer de plusieurs contrôleurs pour pouvoir résoudre ce point de défaillance unique noté sur l'architecture initiale des réseaux SDN.

L'utilisation de ces architectures multicontrôleur va permettre de gérer non seulement les réseaux à grande échelle mais aussi de traiter une très grande quantité de demande de flux.

Mais un problème tel que le problème de placement des contrôleurs découle de ces types d'architectures multicontrôleur. Ce problème de placement des contrôleurs vise à trouver :

- le nombre de contrôleur à déployer dans le réseau ;
- l'endroit où doit déployer chaque contrôleur ;
- L'allocation entre le commutateur et le contrôleur visant à optimiser les objectifs tel que la diminution de la latence, l'amélioration de la fiabilité et l'augmentation de l'efficacité énergétique.

Notre travail porte sur l'analyse de l'impact du placement des contrôleurs sur les performances des réseaux SDN. Après une présentation des concepts et fonctionnement des réseaux SDN, nous avons proposé une étude détaillée des solutions liées au problème de placement des contrôleurs. Ensuite, une solution choisie a été déployée et testée dans l'environnement de simulation Mininet.

Ce mémoire s'organise en quatre parties :

- La première partie porte sur la présentation générale du mémoire qui est composée d'une introduction générale, du contexte, de la problématique, et les objectifs visés pour l'élaboration du mémoire.

- La deuxième partie présente un état de l'art sur les réseaux SDN avec un accent particulier sur les concepts et le fonctionnement.
- La troisième partie présente une étude comparative de solutions existantes sur le problème de placement des contrôleurs dans les systèmes SDN.
- Dans la quatrième partie, nous proposons une implémentation d'une solution SDN multicontrôleur retenue.

1.2 Contexte

Les premières architecture SDN sont constituées d'un seul et unique contrôleur. Ce qui entraine non seulement un point de défaillance unique mais aussi l'incapacité du contrôleur à gérer une grande quantité de demande de flux et d'évoluer lorsque la taille du réseau augmente.

Ainsi, notre travail est orienté dans le contexte d'utiliser plusieurs contrôleurs pour partager les charges et synchroniser les données afin d'améliorer les performances et l'évolutivité des réseaux SDN.

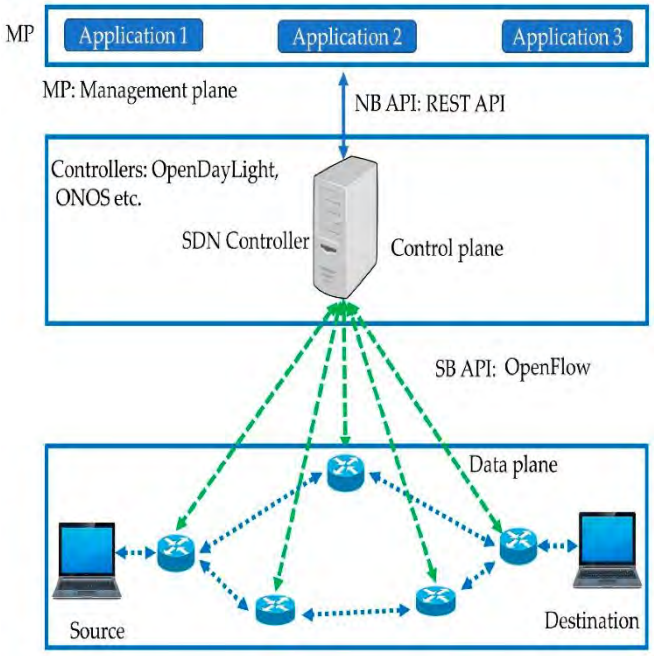


Figure 1.1 : Architecture initiale des réseaux SDN

1.3 Problématique

L'emplacement et le nombre de contrôleurs ont un impact énorme sur les performances du réseau dans SDN. Le problème de placement du contrôleur (CPP) est devenu un point chaud dans la recherche actuelle du SDN. D'où les questions suivantes qui déterminent la problématique de notre mémoire :

- Comment effectuer un placement optimal des contrôleurs pour prendre en charge l'évolutivité du réseau SDN ?
- Comment optimiser le placement des contrôleurs pour gérer de grande quantité de flux ?

1.4 Objectifs

1.4.1 Objectif général

L'objectif général de ce mémoire est de proposer un placement optimal des contrôleurs dans une architecture multicontrôleur afin d'augmenter les performances du réseau.
Ce placement optimal des contrôleurs va permettre aux différents contrôleurs de partager les charges, de diminuer la latence et d'augmenter la fiabilité du réseau.

1.4.2 Objectifs spécifiques

Nous ciblons quatre objectifs spécifiques dans ce mémoire :

- s'initier à la recherche scientifique ;
- faire une étude des réseaux SDN et ses différentes architectures : mono et multi-controleur ;
- éffectuer une étude comparative des contrôleurs (OpenDayLight, Floodlight, ONOS et POX) ;
- faire une étude comparative de solutions existantes concernant le problème de placement du contrôleur ;
- implémenter une solution de réseau SDN multicontrôleur retenue sous Mininet avec OpenDayLight ;

Chapitre 2: Software Defined Networking (SDN): Etat de L'art

2.1 Introduction

Toutefois, afin de continuer son succès, Internet doit relever le défi de traiter des volumes de plus en plus gigantesques de trafic dus à : une explosion du nombre de terminaux mobiles, une utilisation massive des réseaux sociaux et de nouvelles tendances des technologies de l'information comme le cloud computing, le big data, et l'internet des objets qui sont génératrices de gros volumes de données.
Le problème pour relever ce défi dans l'avenir, est qu'il est très difficile d'innover et d'apporter des changements au réseau Internet actuel tant sur le plan des protocoles que de l'infrastructure physique. D'ailleurs, son architecture fondamentale n'a connu aucun progrès significatif en presque trois décennies. La migration des équipements réseau d'IPv4 vers IPv6 a commencé depuis une dizaine d'années et est toujours en cours. Selon [1], le design, l'évaluation et le déploiement d'un nouveau protocole de routage peuvent prendre entre 5 à 10 années. Une des raisons de cette difficulté d'innover réside dans l'architecture des réseaux actuels où le plan de contrôle et le plan de données coexistent dans chaque équipement réseau. Ce problème du couplage du plan de contrôle et du plan de données dans les équipements réseaux, rend les opérations de configuration et de gestion complexes. En effet, les administrateurs réseaux doivent configurer et gérer séparément chaque équipement, ce qui augmente le risque d'erreurs. Et selon nos études, les facteurs humains sont signalés comme étant responsables de 50 à 80 % des interruptions de service et pannes de périphériques réseaux.
C'est dans ce contexte qu'a apparu le concept des réseaux définis par logiciels (Software Defined Networking ou SDN).
Ces changements dans l'infrastructure réseau ont pour conséquence de simplifier des équipements et de les rendre indépendants des fabricants. Il est en outre envisageable d'utiliser le SDN avec différentes approches, afin d'améliorer les performances des réseaux, par exemple l'utilisation de Machine Learning avec SDN permet de fournir plus d'intelligence aux réseaux, et cela grâce aux capacités du SDN. En outre, l'utilisation du concept SDN dans le réseau peut fournir des services innovants, notamment le routage multidiffusion, la sécurité, le contrôle d'accès, la gestion de la bande passante, l'ingénierie de trafic, la QoS, l'efficacité énergétique et diverses formes de gestion des stratégies.

2.2 Software-Defined Networking (SDN)

2.2.1 Définition

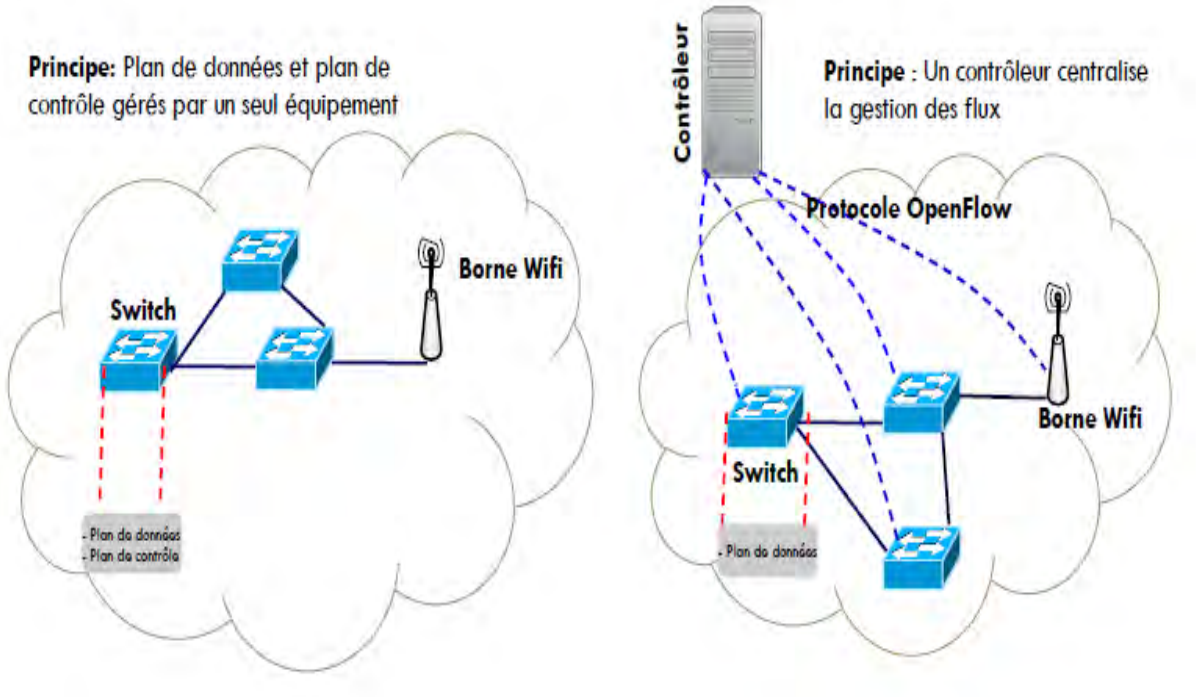
Selon l'ONF (Open Network Fundation) [12] (un consortium d'entreprises à but non lucratif fondé en 2011 pour promouvoir SDN et normaliser ses protocoles) SDN est une architecture qui sépare le plan de contrôle du plan de données, et unifie les plans de contrôle dans un software de contrôle externe appelé Contrôleur, pour gérer plusieurs éléments du plan de données via des APIs (Application Programming Interface).
En d'autres termes, on peut dire qu'une architecture réseau suit le paradigme SDN si elle vérifie les points suivants :

- Le plan de contrôle est complètement découplé du plan de données, cette séparation est matérialisée à travers la définition d'une interface de programmation (Southbound API)
- Toute l'intelligence du réseau est externalisée dans un point logiquement centralisé appelé contrôleur SDN, ce dernier offre une vue globale sur toute l'infrastructure physique.
- Le contrôleur SDN est un composant programmable qui expose une API (NorthboundAPI) pour spécifier des applications de contrôle.

2.3 Différence entre un réseau SDN et un réseau traditionnel

Traditionnellement, un réseau informatique est composé d'équipements réseau interconnectés tels que des switchs et des routeurs. Dans chaque équipement, nous trouvons un mécanisme de transmission dans la couche de transmission et un plan de contrôle qui incorpore le système d'exploitation et les applications. Dans ce modèle, les équipements réseau sont fermés et nous n'avons aucune possibilité d'ajouter une nouvelle application. De plus, avec cette approche traditionnelle de gestion des réseaux, chaque équipement doit être configuré indépendamment. Ce qui est non seulement lourd et fastidieux, mais induit aussi le risque de ne pas avoir l'uniformité dans les configurations. D'où le SDN pour permettre la centralisation des configurations.

Le SDN a pour idée de simplifier la gestion des réseaux et de quitter l'architecture distribuée, pour revenir à un système de gestion centraliser du réseau afin de gérer plus facilement les réseaux. La figure 2.1 donne un aperçu sur la différence entre l'architecture traditionnelle et le SDN.



Réseau traditionnel (Contrôle distribue) Réseau SDN (Contrôle centralisé)

Figure 2.1 : Comparaison entre un réseau traditionnel et un réseau SDN [11]

2.4 Architecture SDN

L'architecture SDN est constituée de trois couches à savoir la couche infrastructure, la couche de contrôle et la couche application. Ces différentes couches se communiquent via les interfaces Southbound, Northbound et East-Westbound APIs comme l'indique la figure 3 Nous décrivons dans la suite ces couches ainsi que les interfaces de communication entre les couches.

La couche la plus basse est la couche de transmission, aussi appelée plan de données. Elle contient les équipements de transmission (FEs : Forwarding Element) tels que les switchs physiques et virtuels. Son rôle principal est de transmettre les données, surveiller les informations locales et collecter les statistiques. La couche de contrôle, également appelée plan de contrôle, est constituée d'un ou de plusieurs logiciels de contrôle (Contrôleurs), ces contrôleurs utilisent des interfaces Sud ouvertes pour contrôler le comportement des FEs et communiquent via des APIs Nord avec la couche supérieure pour superviser et gérer le réseau. La couche d'application (la plus haute dans la figure3) héberge les applications qui peuvent introduire de nouvelles fonctionnalités réseau, comme la sécurité, la configuration dynamique et la gestion. La couche d'application exploite la vue globale et distante du réseau offerte par le plan de contrôle pour fournir des directives appropriées à la couche de contrôle.

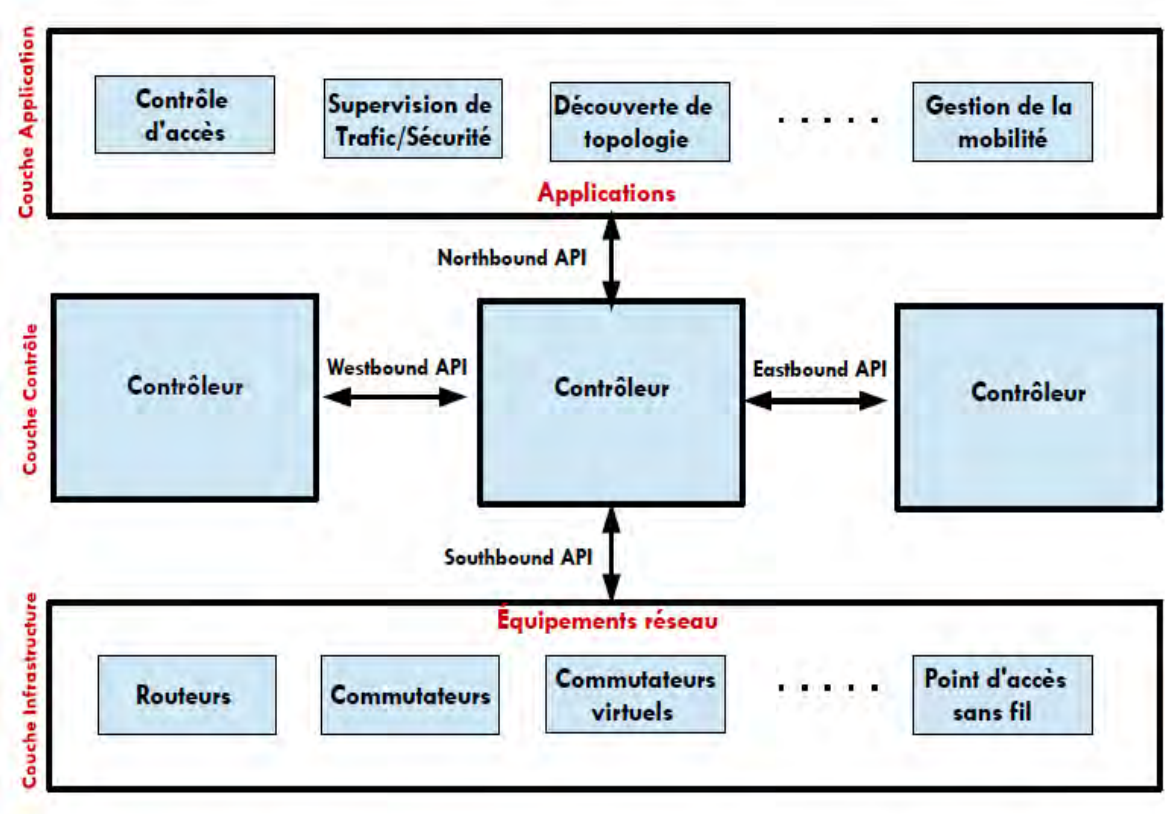


Figure 3 : Architecture détaillée d'un réseau SDN [15]

2.5 Les interfaces de communications

Le contrôleur interagit avec les autres couches à travers les interfaces Sud et Nord et avec les autres contrôleurs à travers une interface Est-Ouest.

2.5.1 Le Southbound API

L'interface entre la couche infrastructure et la couche de contrôle est appelée Southbound API ou interfaces Sud. Les Southbound APIs représentent les interfaces de communication, qui permettent au contrôleur SDN d'interagir avec les équipements de la couche d'infrastructure, tel que les switches, et les routeurs. OpenFlow est le Southbound API le plus souvent utilisé dans l'architecture SDN permettant la programmation du plan de données.
Bien qu'OpenFlow est le Southbound API le plus utilisé et même le standard de facto, il y a aussi plusieurs autres Southbound API pour gérer les communications entre ces deux niveaux de l'architecture SDN, tels que Forwarding and Contrôle Element Separation (ForCES) [13] et OpFlex [14].

2.5.2 Le Northbound API

L'interface entre la couche de contrôle et la couche application est le Northbound API ou l'interface nord qui est une des abstractions clés de l'environnement SDN. Il fournit généralement une liste des fonctions réseaux de base associées au fournisseur, qui est ensuite utilisées pour configurer un équipement spécifique à l'utilisateur, et le contrôleur l'interprète dans un langage que chaque équipement réseau peut comprendre.
Au moment de la rédaction de ce mémoire, il n'existe aucun standard intervenant entre la couche de contrôle et celle d'application du côté d'ONF ou d'autres organisations. C'est pourquoi ONF a créé un groupe de travail pour élaborer un protocole standard sur cette interface. Il existe actuellement plusieurs types de contrôleurs SDN qui offrent une grande variété de Northbound API. Les Northbound API peuvent être classées principalement en trois catégories, à savoir les API REST (REpresentational State Transfer) [16], les API ad hoc spécialisées et les langages de programmation tels que Procera [17], Frenetic [18].
L'API REST est le plus utilisé car il fournit une intégration simple et permet une interaction minimale entre un client et un serveur. L'API REST est l'œuvre de Roy Thomas Fielding qui a aussi participé à la définition du protocole HTTP.

2.5.3 Le East Westbound API

L'interface qui permet aux contrôleurs de partager une vue commune du réseau et de coordonner l'exécution des règles et des protocoles est l'API East Westbound ou interfaces Est/Ouest. C'est à travers cette interface qu'est gérée la manière dont les contrôleurs du SDN interagissent les uns avec les autres pour partager des informations. Pour obtenir une vue globale de l'ensemble du réseau, les opérateurs de réseau utilise les API East-Westbound pour faire communiquer les contrôleurs les uns avec les autres et partager les vues du réseau. Une interface East-Westbound est également importante pour automatiser les décisions de réseau afin de limiter les interventions des administrateurs réseaux sur des réseaux opérateurs de grande envergure.

Un avantage considérable dans le développement du SDN est que la plupart de ces interfaces sont du domaine ouvert (disponible sous licence open source).
ALTO et HyperFlow sont des exemples de protocoles (Eastbound et Westboud) utilisés entre les contrôleurs en cas d'utilisation de plusieurs contrôleurs.

2.6 La couche de contrôle

Le rôle de la couche de contrôle est de contrôler et de gérer les équipements de l'infrastructure, et de les relier avec les applications. Il est composé d'un ou de plusieurs contrôleurs et il est considéré comme système d'exploitation du réseau. Les premières versions du SDN présentent un plan de contrôle composé d'un seul contrôleur centralisé. Par la suite des architectures distribuées utilisant plusieurs contrôleurs ont été proposées pour améliorer les performances et la scalabilité du réseau.

2.7 Le protocole OpenFlow

Le protocole OpenFlow est actuellement considéré comme un protocole utilisé comme une interface Sud dans les architectures basées sur SDN, cependant, dans ses débuts il a été la première implémentation réelle du concept SDN avec toute une architecture composée par les switchs, une chaine de communication sécurisée et un contrôleur de référence. Le protocole OpenFlow a été initialement proposé et implémenté par l'université de Stanford, et il a été standardisé par la suite par l'ONF.

2.8 L'architecture du protocole OpenFlow

L'architecture d'OpenFlow est une implémentation réelle des réseaux SDN, nous la détaillons ici pour bien comprendre les concepts SDN. Cette architecture est basée sur trois composantes : (1) l'infrastructure appelée aussi le plan de données, composée des commutateurs OpenFlow ; (2) le plan de contrôle, constitué par un ou plusieurs contrôleurs ; (3) la chaîne sécurisée qui connecte les commutateurs avec le plan de contrôle. Un commutateur OpenFlow est un équipement de transmission qui contient des tables de flux ou flow table comme l'indique la figure 4. Ces tables de flux contiennent un ensemble d'entrées aussi appelées règles, où chacune est constituée par les champs d'en-tête, des compteurs et des actions.
Les champs d'en-tête d'une entrée contiennent les informations nécessaires pour déterminer le paquet auquel cette entrée sera appliquée. Pour permettre une transmission rapide des paquets avec OpenFlow, le commutateur utilise une mémoire de type TCAM (Ternary Content Addressable Memory) qui permet une recherche rapide des masques. Le champ d'en-tête peut aussi identifier différents protocoles selon les spécifications d'OpenFlow, comme Ethernet, IPv4, IPv6 ou MPLS. Les Compteurs sont réservés à la collecte des statistiques de flux. Ils enregistrent le nombre de paquets et d'octets reçus et la durée des flux dans la mémoire temporaire. Les Actions spécifient comment les paquets d'un flux seront traités. Les Actions communes sont : transmettre, supprimer, modifier le champ, etc. Le contrôleur est responsable de l'installation et de la mise à jour des tables de flux des commutateurs. En

insérant, modifiant et supprimant les entrées de flux, le contrôleur peut modifier le comportement des commutateurs en ce qui concerne la transmission.

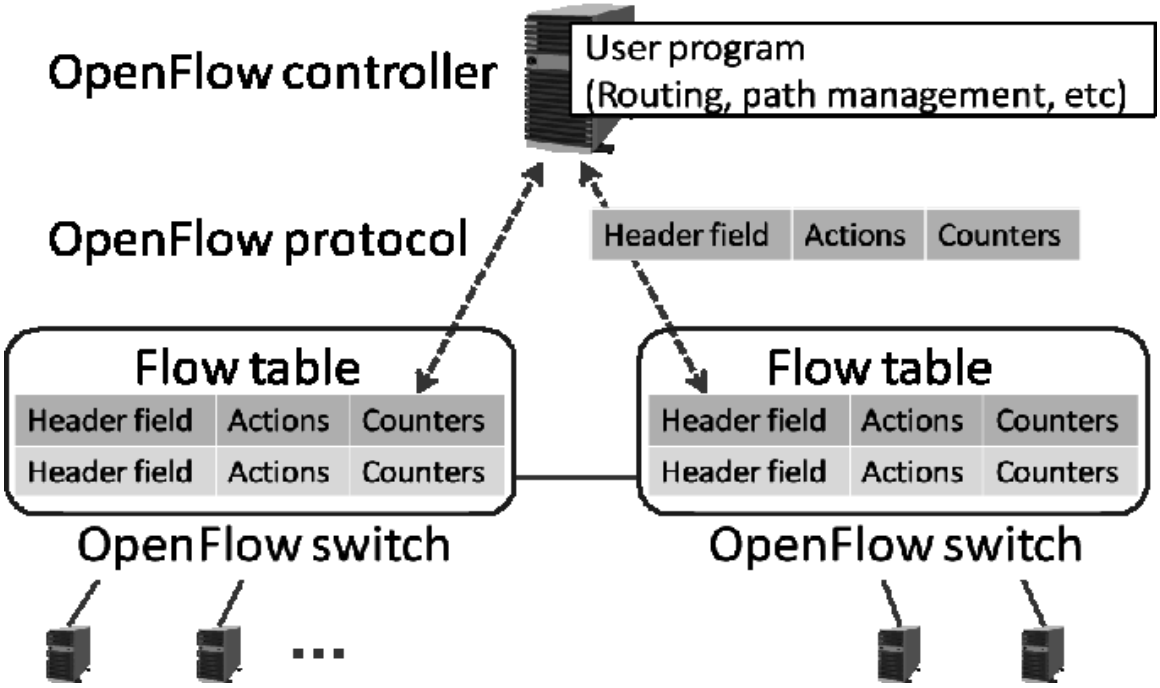


Figure 4: Architecture détaillée d'un réseau SDN [19]

2.9 Fonctionnement du protocole OpenFlow

Lorsqu'un paquet arrive à un commutateur, le commutateur vérifie s'il y a une entrée dans la table de flux qui correspond à l'en-tête de paquet. Si c'est le cas, le commutateur exécute l'action correspondante dans la table de flux. Dans le cas contraire, c'est-à-dire s'il n'y a pas une entrée correspondante (1) , le commutateur génère un message asynchrone vers le contrôleur (2) sous la forme d'un 'Packet_in', puis le contrôleur décide selon sa configuration une action pour ce paquet, et envoie une nouvelle règle de transmission sous la forme d'un 'Packet_out' et 'Flow-mod' au commutateur (3), et enfin, la table de flux du commutateur est actualisée, pour prendre en compte la nouvelle règle installé par le contrôleur (4). La Figure 5 [2] décrit le processus de transmission d'un paquet avec openflow

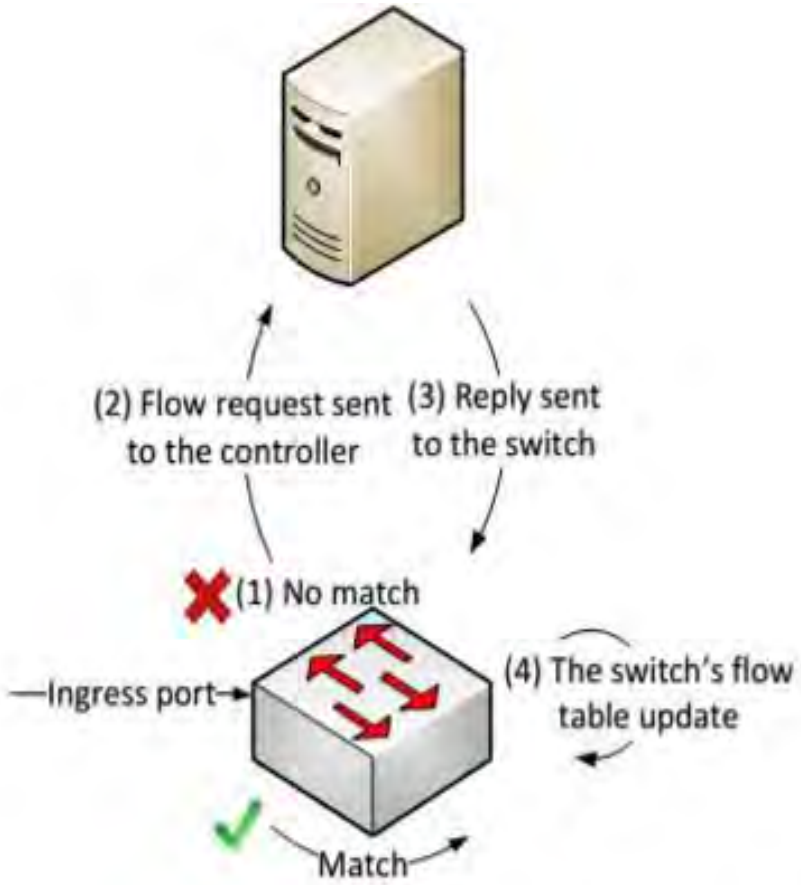


Figure 5: processus de transmission d'un paquet avec openflow [2]

2.10 Quelques contrôleurs OpenFlow

Etant la composante la plus importante du concept SDN, le contrôleur SDN est un programme logiciel comparable à un système d'exploitation réseau. Grace au Southbound API, le contrôleur est responsable de la manipulation de la table de flux d'un commutateur. Autrement dit, le contrôleur SDN a la charge de la gestion et de la configuration du réseau en installant des règles d'acheminement des paquets, appropriées sur les équipements réseau (commutateurs, routeurs) de la couche infrastructure à travers le Southbound API. Cette interface de programmation est très importante car elle offre la possibilité d'innovation pour les programmeurs et permet d'accélérer le développement et l'implémentation des services réseaux. OpenFlow est le Southbound le plus utilisé. Il permet au contrôleur SDN d'ajouter, de modifier et de supprimer des entrées dans la table de transfert d'un commutateur OpenFlow. Un canal sécurisé relie le contrôleur à un commutateur OpenFlow. Grâce à ce canal qui sert d'interface de connexion, le contrôleur gère les commutateurs OpenFlow, reçoit des paquets des commutateurs OpenFlow et envoie des paquets aux commutateurs OpenFlow. Il existe plusieurs types de contrôleur logiciel dont les plus connus sont ONOS, OpenDaylight, POX, RYU, Floodlight et Beacon. Ils se distinguent par les langages de programmation utilisés et le protocole supporté. A titre d'exemple et comme l'indique le tableau 1, le contrôleur OpenDaylight exploité dans le cadre de nos travaux est conçu en Java et en Python. Le contrôleur tient une vue globale du réseau à travers son service de découverte de topologie, très important pour une exploitation correcte des autres services réseaux, comme la configuration réseau par exemple. Certains contrôleurs sont centralisés (Beacon,

Floodlight) et d'autres distribués (ONOS, OpenDaylight) et intègrent des API supplémentaires comme Eastbound et Westbound. Les contrôleurs distribués permettent de relever les limites sur l'usage d'un contrôleur unique, qui peut non seulement devenir un point critique du point de vue de la sécurité, mais ne permet pas de gérer des larges réseaux avec un nombre élevé des équipements réseaux.

2.10.1 NOX

NOX est le premier contrôleur disponible au public écrit en langage C, l'utilisation d'un seul thread dans son cœur limite son déploiement. Plusieurs dérivations du NOX ont été proposées plus tard, notamment le Nox dans sa version multithread appelée NOX-MT, le QNOX, NOX supportant la qualité de service (QoS), FortNOX, une extension du NOX implémentant un analyseur de détection des conflits dans les règles de flux causées par les applications, et enfin, le contrôleur POX un contrôleur basé sur NOX en langage python, dont le but est d'améliorer les performances du contrôleur original NOX.

2.10.2 OpenDayLight

OpenDayLight est un contrôleur SDN Open source modulaire qui permet de concrétiser la virtualisation des fonctions réseaux à travers une définition cohérente de ses interfaces de programmation. Conçu en Java et en Python, le contrôleur OpenDayLight est multiplateforme et fait partir des contrôleurs distribués. Considéré comme une référence dans la programmabilité des réseaux, OpenDayLight permet à ses utilisateurs et équipementiers de personnaliser leurs solutions de gestion des réseaux selon leurs besoins.

Depuis son introduction, OpenDaylight a connu plusieurs versions dont chaque version apporte soit des correctifs, des nouveaux services ou le support des nouveaux protocoles. OpenDayLight est très largement adopté par les intégrateurs et par l'industrie des équipements. L'architecture de la plateforme OpenDaylight, représentée sur la figure 6, est basée sur le Model-Driven Service Abstraction Layer (MD-SAL) où les équipements réseaux et les applications sont représentés sous forme d'objets ou de modèles dont les interactions sont traités dans la couche d'abstraction appelé Service Abstraction Layer (SAL). Le SAL est un élément très important du contrôleur OpenDaylight, permettant d'assurer le mécanisme d'échange et d'adaptation de données entres les modèles YANG 24, représentant les équipements réseaux et les applications. Il permet aussi de faciliter l'intégration d'une nouvelle Southbound API pour le développement des services sans lien avec les API utilisées pour les implémenter.

OpenDaylight est un framework modulaire et multi-protocoles. Cet aspect permet aux utilisateurs d'OpenDaylight de choisir et d'installer uniquement les services et protocoles adaptés à leurs contextes. Cela permet aussi aux utilisateurs de résoudre des problèmes complexes en choisissant une combinaison des outils et protocoles qui les intéressent.

Du point de vue sécurité, OpenDaylight est conçu pour fournir l'authentification, l'autorisation, la traçabilité ainsi que la détection et la sécurisation automatique des équipements

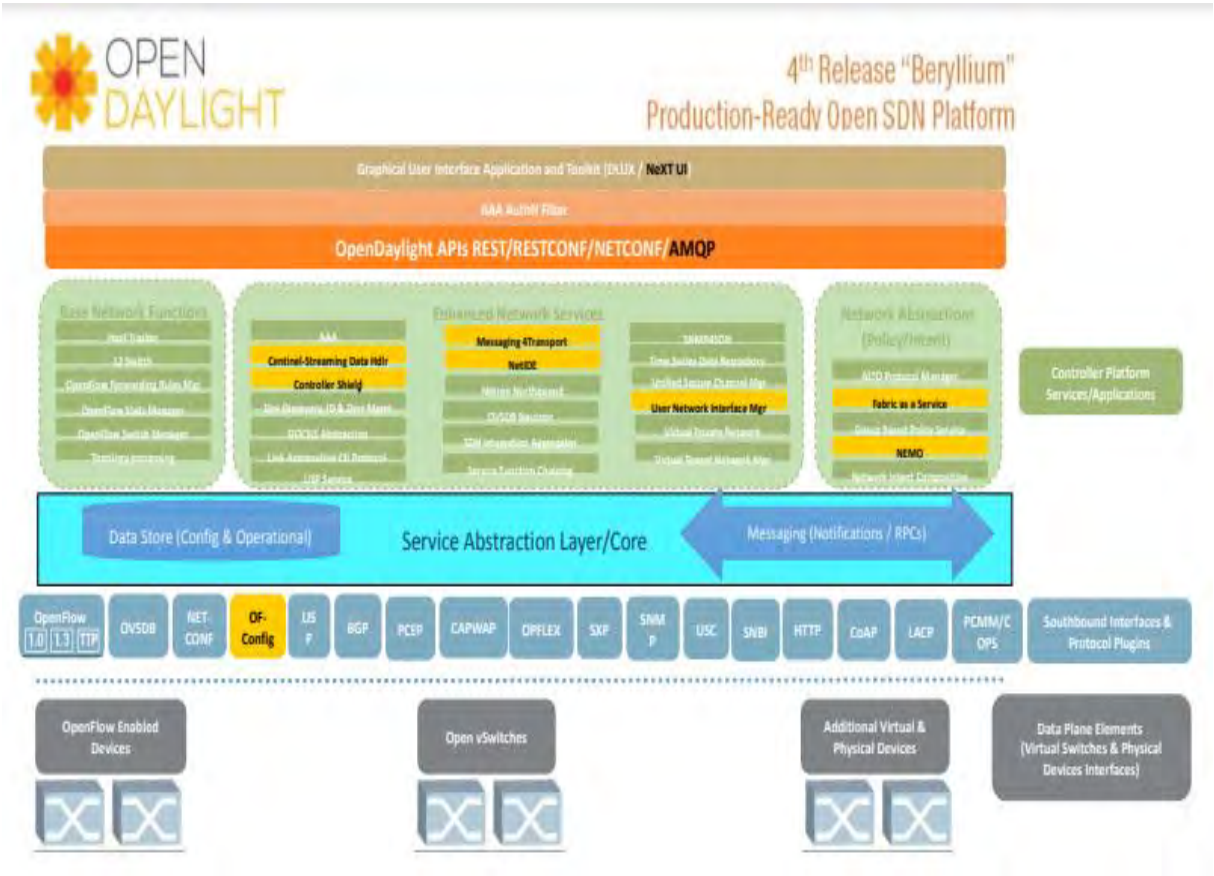


Figure 6: Architecture du contrôleur OpenDaylight (version Beryllium) [16]

2.10.3 Floodlight

Floodlight est un contrôleur open-source OpenFlow basé sur Java, pris en charge par Big Switch Networks. Il est sous licence Apache. Il est facile à configurer et a montré aussi de grandes performances. Avec toutes ses fonctionnalités, Floodlight est plus une solution complète. C'est une variante de Beacon caractérisée par sa simplicité et sa performance. Il est aujourd'hui supporté et amélioré par une large communauté de développeurs, comprenant des industriels comme Intel, Cisco, HP, Big switch et IBM.

2.10.4 POX

POX est une plate-forme logicielle de mise en réseau écrite en Python. POX a commencé sa vie en tant que contrôleur OpenFlow, mais peut maintenant également fonctionner comme un commutateur OpenFlow, et peut être utile pour écrire des logiciels de mise en réseau en général. Il prend actuellement en charge OpenFlow 1.0 et inclut un support spécial pour les extensions Open vSwitch / Nicira.

2.10.5 ONOS

Le contrôleur ONOS est principalement conçu pour les réseaux de transporteurs. Cela leur donne la possibilité de fournir de nouveaux Services SDN avec leurs services propriétaires initiaux.

L'architecture ONOS est conçue pour maintenir une vitesse élevée des réseaux à grande échelle. Sa principale caractéristique est son support pour les réseaux hybride

Contrôleur	Langage	Topologie du Contrôleur	Open Source	Distribué	Difficulté de développement
NOX	C++	Plat	Oui	Non	Moyen
POX	Python	Plat	Oui	Non	Facile
ONOS	Java	Plat	Oui	Oui	Moyen
Floodlight	Java	Plat	Oui	Oui	Difficile
Opendaylight	Java	Plat	Oui	Oui	Difficile

Tableau 1: Tableau comparatif de quelques contrôleurs SDN

2.11 Architecture initiale (mono contrôleur)

Les premières propositions et déploiements de réseaux SDN utilisaient un seul contrôleur centralisé (ce qu'on appelle un plan de contrôle physiquement centralisé). Cela pose le problème d'avoir un point unique de défaillance (SPOF, Single Point Of Failure), et l'incapacité du contrôleur à gérer une grande quantité de demandes de flux et, par conséquent, l'incapacité à évoluer lorsque la taille du réseau augmente.

Avec un seul contrôleur physique, les commutateurs sollicitent le contrôleur à chaque fois qu'il n'y a pas de règle qui correspond au paquet reçu dans leur table de flux. Dans ce cas, le contrôleur traite le paquet selon l'application désirée et instaure les actions appropriées dans les tables de flux des commutateurs. Ceci pose le problème de performance du contrôleur et sa capacité à traiter un grand nombre de demandes (débit), surtout quand la taille du réseau augmente. Aussi, le temps d'installer une nouvelle règle (latence) est très important car il influence le délai de traitement des paquets et par conséquent il peut améliorer le fonctionnement du réseau.

Par exemple, le premier contrôleur SDN, appelé NOX, a un temps d'installation de flux de moins de 10 ms et peut gérer jusqu'à 30000 demandes par seconde. Par la suite, d'autres contrôleurs ont été proposés comme Floodlight et OpenDayLight pour améliorer les performances d'un seul contrôleur.

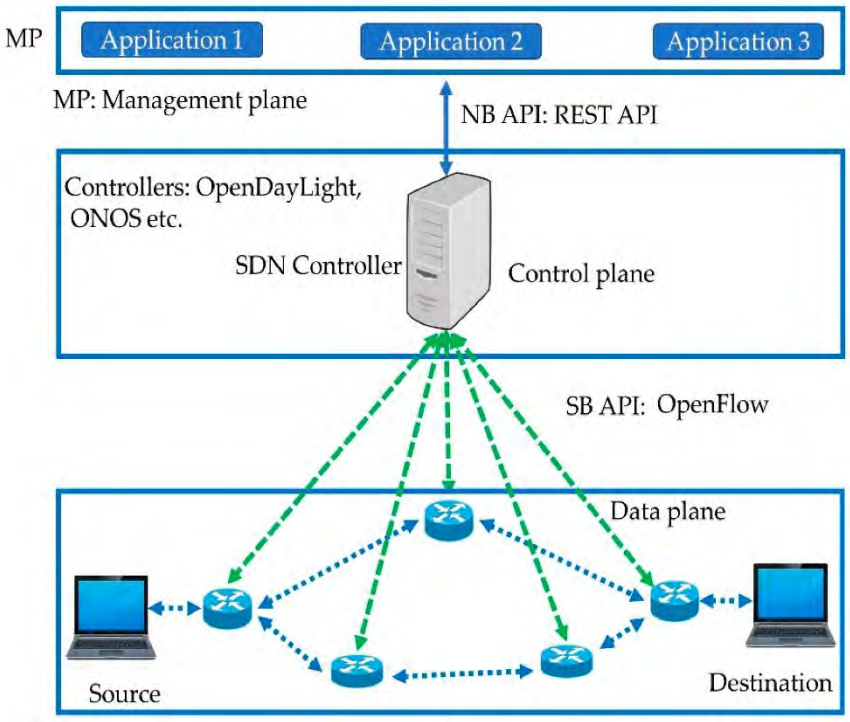


Figure 7: Architecture initiale de SDN *[20]*

2.12 Architecture multi contrôleur

Pour surmonter le problème du SPOF et améliorer les performances, nous allons utiliser les architectures avec plusieurs contrôleurs SDN. Une architecture multicontrôleur appelé aussi architecture physiquement distribuée est un ensemble de contrôleurs fonctionnant ensemble pour atteindre un certain niveau de performance et d'évolutivité.
Il existe deux modèles d'architectures pour ce type d'architecture physiquement distribuée qui sont : l'architecture logiquement centralisée et l'architecture logiquement distribuée.

2.12.1 Architecture logiquement centralisée :

L'architecture logiquement centralisée utilise plusieurs contrôleurs pour partager les charges et synchroniser les données entre les contrôleurs afin d'améliorer les performances et l'évolutivité des réseaux SDN. C'est comme s'il n'y avait qu'un seul contrôleur qui commande tout le réseau. L'intérêt de cette solution apparaît dans les réseaux de centres de données (data center), où les contrôleurs partagent une grande quantité d'informations pour assurer la cohérence d'un réseau large avec précision. Parmi les solutions qui utilisent une architecture logiquement centralisée nous trouvons :

- HyperFlow : utilise deux composantes majeures, un contrôleur implémenté comme application sur le NOX et un système de distribution des évènements utilisant WheelFS [3] pour construire un modèle de messagerie de type publié/souscrire (Publish/Subscribe). En conséquence, HyperFlow peut supporter plus d'évènements de flux en gardant un délai d'installation de flux minimal.

- Opendaylight (ODL) : fonctionne en mode simple où un seul contrôleur est utilisé, et en mode cluster pour construire un réseau distribué et fournir une haute disponibilité, fiabilité, et évolutivité. Il utilise le Framework Akka [21] pour synchroniser les données entre les contrôleurs multiples. En effet, ODL fournit la persistance en distribuant les bases de données, l'accès à distance en constituant des connexions entre les membres du cluster et la découverte des membres en utilisant le protocole Gossip.

- Onos : acronyme de Système d'Exploitation pour les Réseaux ouverts (Open Network Operating System), où chaque contrôleur individuel partage une base de données commune en utilisant le modèle de cluster. Onos fournit deux prototypes, le premier servant à extraire une vue globale du réseau à partir de l'architecture distribuée, destiné aux applications qui nécessitent une évolutivité du réseau et une tolérance de panne. Le deuxième prototype est conçu pour l'amélioration des performances notamment la latence des événements.

Néanmoins, ils ne sont pas adaptés aux réseaux étendus composés de plusieurs domaines ou systèmes autonomes (AS), et nécessitent un trafic extensif entre les contrôleurs pour garder une vue globale sur le réseau.

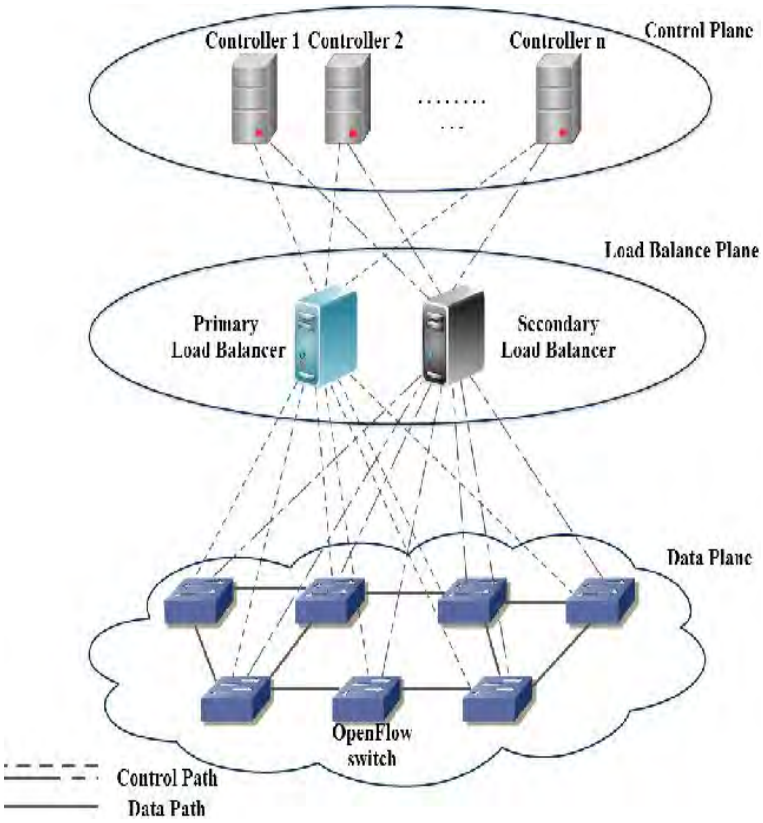


Figure 8: Architecture logiquement centralisée *[20]*

2.12.2 Architecture logiquement distribuée

Les contrôleurs sont physiquement et logiquement distribués. De plus, chaque contrôleur a juste une vue du domaine dont il est responsable, et il ne peut prendre des décisions qu'à sa place, contrairement à une architecture logiquement centralisée, où chaque contrôleur prend une décision basée sur une vue globale du réseau. Parmi les solutions qui utilisent une architecture logiquement distribuée nous trouvons :

- SDNi : IETF développe un protocole Est/Ouest appelé SDNi pour réaliser une interconnexion entre les contrôleurs SDN. Dans ce projet, les chercheurs décrivent une interface pour échanger les informations entre plusieurs domaines supportant le SDN, afin de synchroniser la base de données réseau et coordonner les décisions entre les contrôleurs (Figure 9). Une proposition d'implémentation du SDNi est fournie par la société TATA (TCS) [22], en proposant le protocole BGP (Border Gateway Protocol) comme candidat pour implémenter SDNi

- DISCO : Disco propose des contrôleurs SDN logiquement distribués, où chaque contrôleur DISCO commande son domaine réseau et communique avec les autres extensions en utilisant un canal de contrôle appelé 'Agent'. Alors, il fournit un plan de contrôle distribué et ouvert pour les réseaux multi-domaines en utilisant une communication orientée vers les messages. Le succès derrière DISCO, c'est qu'il sépare le plan de contrôle en deux plans. La partie intra-domaine qui rassemble les principales fonctionnalités du contrôleur, et la partie inter-domaine qui gère la communication entre les autres contrôleurs, en envoyant un agent de type réservation, état du réseau ou opérations d'interruption. Cette solution a été testée pour l'interruption des topologies inter-domaine, les demandes de priorité de service bout-en-bout (end-to-end) et les cas de migration des machines virtuelles (VM).

- WEBridge : La proposition 'WEBridge' aborde le même problème, mais dans le cas de plusieurs systèmes d'exploitation. Elle utilise trois nouveaux modules pour créer une vue globale sur le réseau et garantir l'intégrité des données entre les contrôleurs, ces modules sont la virtualisation réseau, le pont Est-Ouest et l'extension LLDP. Le module de virtualisation réseau permet une abstraction du réseau physique au réseau virtuel pour chaque domaine. Par la suite, le module 'WEBridge' utilise un modèle de publier/souscrire pour synchroniser les données entre les contrôleurs. Enfin, le rôle de l'extension LLDP est de permettre à 'WEBridge' de supporter plusieurs contrôleurs issus de différents contrôleurs.

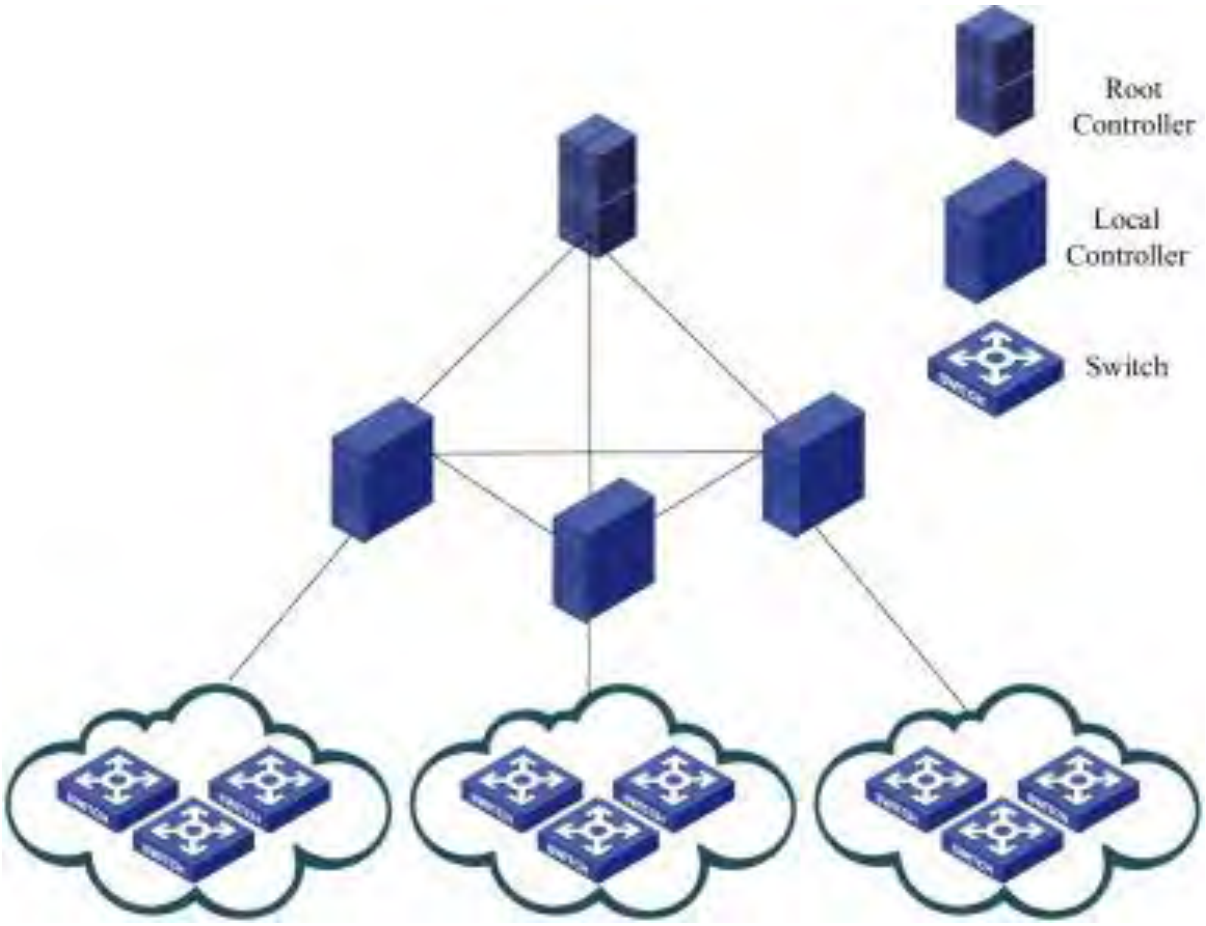


Figure 9: Architecture logiquement distribuée [20]

2.13 Avantages et inconvénients de SDN

2.13.1 Avantages

- SDN permet de modifier les stratégies réseau d'une manière simple, car il suffit de changer une politique de haut niveau et non de multiples règles dans divers équipements de réseau.
- L'intelligence du réseau est centralisée dans le contrôleur SDN qui maintient une vision globale du réseau.
- Avec son contrôleur, SDN garantit également une politique unifiée et à jour sur l'ensemble du réseau, puisque le contrôleur est responsable de l'ajout de règles calculées dans les commutateurs, il n'y a aucun risque qu'un administrateur de réseau ait oublié un commutateur ou installé des règles incohérentes entre les appareils. En effet, l'administrateur se contentera de spécifier une nouvelle règle et le contrôleur adaptera la configuration pour envoyer des règles cohérentes dans chaque périphérique pertinent.
- SDN apporte une grande flexibilité dans la gestion du réseau. Il devient facile de réacheminer le trafic, d'inspecter des flux particuliers, de tester des nouvelles politiques ou de découvrir des flux inattendus.
- SDN peut également être utilisée pour gérer les informations de routage de manière centralisée en déléguant le routage et en utilisant une interface pour le

contrôleur. Ainsi il peut être utilisé pour gérer l'utilisation des liens et améliorer la performance en augmentant l'utilisation moyenne de chaque lien sans saturer totalement le lien car il serait contre-productif.

2.13.2 Inconvénient

- Par rapport aux réseaux traditionnels, les réseaux SDN montrent de grands avantages dans de nombreux aspects, mais il existe également le problème du déséquilibre de charge. Si la répartition de la charge est inégale dans les réseaux SDN, cela affectera grandement les performances du réseau.
- Il faut modifier toute l'infrastructure réseau pour implémenter le protocole SDN et le contrôleur SDN. Il nécessite donc une reconfiguration complète du réseau. Cela augmente les coûts en raison de la reconfiguration.
- Le réseau programmable (SDN) découple le plan de contrôle et le plan de données. Cette séparation pose un problème connu sous le nom de placement du contrôleur, c'est-à-dire le nombre et l'endroit où les contrôleurs devraient être installés. On va présenter et discuter ce problème dans le chapitre suivant.

Chapitre 3 : Etudes des différentes solutions existantes sur le problème de placement du contrôleur (CPP)

3.1 Introduction

La mise en réseau définie par logiciel (SDN) fournit des réseaux de communication avec un fonctionnement et une programmabilité flexibles dans lesquels la gestion ainsi que le fonctionnement d'un réseau sont effectués via un contrôleur ayant une vue globale du réseau. Néanmoins, le plan de contrôle ne représente pas nécessairement un appareil unique et centralisé ; mais il peut comprendre plusieurs contrôleurs chargés de contrôler divers domaines administratifs du réseau ou différentes parties de l'espace de flux.

En plus avec l'expansion du réseau SDN, il est difficile pour un contrôleur unique de répondre aux besoins de gestion étendus. Afin d'améliorer l'évolutivité et la fiabilité du réseau et éviter un point de défaillance unique, une architecture réseau multi-contrôle centralisée logiquement et physiquement distribuée est nécessaire. Dans ces architectures de contrôleur SDN distribuée, les performances du réseau dépendent considérablement du placement des contrôleurs, ce que l'on appelle le problème de placement du contrôleur (CPP).

Pour un réseau donné, le problème de placement du contrôleur considère principalement trois questions :

- Le nombre de contrôleurs
- La position des contrôleurs
- L'allocation entre le commutateur et le contrôleur

Dans ce chapitre, nous présentons des travaux liés au problème de placement des contrôleurs (CPP) dans les réseaux programmables et leurs impacts sur les performances,

3.2 Pourquoi le problème de placement des contrôleurs

Le nombre de contrôleurs et leur placement dépendent du ou des objectifs à optimiser. Nous classons les métriques de performance en deux sous classes, l'une indépendante, où les objectifs peuvent être considérés isolément et l'autre dépendante, où les métriques doivent être considérées conjointement avec les métriques indépendantes. Les mesures de performance qui relèvent d'objectifs indépendants sont la latence, le temps de configuration du flux, la disponibilité et la capacité. La métrique de latence peut être davantage sous-classée en fonction du type de nœuds entre lesquels la latence est mesurée. La métrique de disponibilité peut également être sous-classée en fonction de la disponibilité du réseau en cas de panne ou de la manière dont le réseau se remet des pannes. L'équilibrage de charge est la mesure la plus courante prenant en compte la capacité. Les objectifs dépendants sont conscients de l'énergie et des coûts. Les deux objectifs principaux qui relèvent de l'énergie consciente sont le contrôleur élastique et le lien élastique. Les métriques sensibles aux coûts qui sont principalement utilisées sont la métrique de coût de déploiement et la métrique de coût de migration de commutateur.

3.3 Formulation du problème

La topologie du réseau du plan de données est modélisée sous la forme d'un graphe non orienté G (S, L) où S représente l'ensemble des commutateurs et L désigne l'ensemble des liens entre les commutateurs. Un lien lij entre le commutateur si et sj est défini comme suit :

$$l_{ij} = \begin{cases} 1, & \text{s'il y a un lien entre si et sj;} \\ 0 & \text{sinon.} \end{cases} \qquad (1)$$

3.4 Les impacts du CPP sur les performances

Heller a d'abord analysé l'impact du déploiement du contrôleur sur la latence moyenne du réseau et la latence maximale. Par la suite, d'autres objectifs ont été proposés, notamment l'équilibrage de charge, la fiabilité, l'économie d'énergie. En raison du conflit entre différents objectifs, l'optimisation multi-objectifs a également été une solution réalisable. Le CPP est d'abord classé en 4 sections : latence, fiabilité, coût et multi-objectif. Ensuite, la latence est classée en 4 sous-sections : latence moyenne entre le contrôleur et le commutateur (latence moyenne SC), latence la plus faible entre le contrôleur et le commutateur (latence la plus faible SC), latence moyenne entre les contrôleurs (latence moyenne CC), et latence de traitement. La fiabilité est divisée en 3 sous sections : chemin de contrôle multiple, contrôleur multiple et chemin de contrôle le plus court. Et le coût est divisé en 2 sous sections : coût de déploiement et consommation d'énergie.

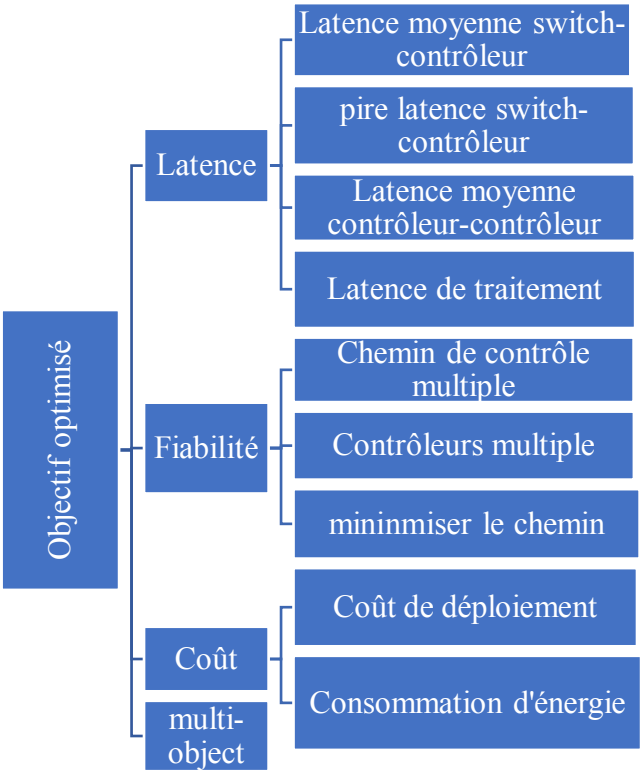


Figure 10 : Classification de l'objectif optimisé

3.4.1 La Latence

L'une des mesures de performances les plus fréquemment utilisées est la latence ou le délai. La latence globale comprend la transmission, la propagation, la mise en file d'attente et le délai de traitement. Lorsque le réseau n'est pas obstrué, la latence de la file d'attente est négligeable. La latence de transmission est liée au paquet de données et au débit du port, et généralement à une valeur fixe dans le cas du même périphérique réseau. Par conséquent, le problème CPP ne prend généralement en compte que la latence de propagation et la latence de traitement où la latence de propagation est principalement déterminée par la distance entre deux nœuds et La latence de traitement est principalement affectée par la capacité de traitement et la charge du contrôleur.

La latence peut également être analysée à partir de différents niveaux de l'architecture SDN. En intra-domaine, il existe deux types de latence, la pire latence entre le contrôleur et le commutateur et la latence moyenne entre le contrôleur et le commutateur en reflétant les performances globales doit être aussi faible que possible; dans l'inter domaine, la latence moyenne entre les contrôleurs doit être prise en compte en raison de la communication entre les contrôleurs pour la cohérence de la vue, et la latence de traitement, qui augmenterait considérablement lorsque la charge dépasse la puissance de traitement du contrôleur, doit être réduite par l'équilibrage de charge entre les contrôleurs. Après une analyse minutieuse, nous classons la latence en 4 aspects : (1) la latence moyenne entre le commutateur et le contrôleur (SC-avg latency), (2) la pire latence entre le commutateur et le contrôleur (SC-pire latence), (3) la latence moyenne entre les contrôleurs (latence CC-moyenne) et (4) latence de traitement.

3.4.1.1 La latence moyenne entre le commutateur et le contrôleur (SC-avg latency)

SC-avg latency représente la valeur moyenne de la latence de transmission de paquets entre le commutateur et le contrôleur, reflétant les performances de base de la latence de propagation dans SDN.

3.4.1.2 la pire latence entre le commutateur et le contrôleur (SC-pire latence)

SC-pire latence désigne la valeur maximale de la latence de transmission de paquets entre le commutateur et le contrôleur, qui est généralement un objectif optimal dans un environnement à hautes performances ou à une contrainte stricte.

3.4.1.3 La latence moyenne entre les contrôleurs (CC-AVG LATENCE)

La communication entre les contrôleurs est cruciale pour atteindre une vue cohérente de l'état du réseau, nécessaire pour bon fonctionnement de l'application réseau. L'observation et l'enquête confirment que les frais généraux de communication causé par le maintien de l'état partagé entre les contrôleurs est très important.

3.4.1.4 Latence de traitement

La latence de traitement est principalement affectée par la puissance de traitement et la charge du contrôleur. Lorsque la charge du contrôleur approche ou dépasse la puissance de traitement du contrôleur, le délai de traitement augmente considérablement. Par conséquent, la latence de traitement est généralement optimisée en équilibrant la charge des contrôleurs. L'expression

3.4.2 Fiabilité

Étant donné que les pannes de réseau peuvent provoquer des interruptions de communication entre les composants du réseau (tels que les contrôleurs ou les commutateurs) dans le SDN, ce qui entraîne une perte de paquets et une dégradation des performances sévères, il est d'une grande importance de tenir compte de la fiabilité du réseau SDN lors du déploiement des contrôleurs. Les messages SDN de gestion et de contrôle sont transmis via le chemin de contrôle, de sorte que la fiabilité du chemin de contrôle affecte directement la fiabilité du SDN. Un chemin de contrôle complet se compose de nœuds, de liaisons et de contrôleurs. En cas de défaillance du contrôleur, les commutateurs doivent être connectés à plusieurs contrôleurs différents pour éviter un point de défaillance unique. Et pour les pannes de nœuds et de liaison, la première approche considère que les commutateurs sont connectés à un contrôleur sur deux chemins de contrôle disjoints. La méthode tente de réduire la longueur du chemin de contrôle qui se compose de moins d'éléments de réseau.

Après analyse, nous classons la méthode d'amélioration de la fiabilité en 3 aspects : chemin de contrôle multiple, contrôleur multiple et minimisant le chemin de contrôle.

3.4.3 Le Coût

Compte tenu des deux aspects de l'économie et de la protection de l'environnement, le coût de consommation du réseau ne doit plus être ignoré avec le développement rapide d'internet. Le coût du SDN comprend principalement le cout de la construction du réseau et les coûts

d'exploitation et d'entretien ultérieurs. Ainsi, nous classons la solution de coût existante en deux aspects : (a) le coût de déploiement et (b) la consommation d'énergie.

- Le cout de déploiement : Le coût de déploiement fait référence au coût des périphériques réseau (contrôleurs et commutateurs) et à leurs dépenses opérationnelles, y compris l'installation du contrôleur dans le réseau, la liaison du contrôleur au commutateur et la liaison de ces contrôleurs.
- Consommation d'énergie : La consommation d'énergie des périphériques réseau sous faible charge réseau représente toujours plus de 90% de la charge pendant les heures de pointe. Et une solution de déploiement de contrôleur raisonnable pourrait arrêter les liaisons et les contrôleurs autant que possible lorsque le réseau est inactif.

3.4.4 Multi-Objects

Dans le scénario d'application réel du SDN, plusieurs métriques de performance sont généralement sélectionnées pour résoudre le CPP en même temps, de sorte que le problème du CPP peut également être utilisé comme un problème d'optimisation multi-objectif. Étant donné que plusieurs mesures ne peuvent pas obtenir le meilleur de manière synchrone, comme les conflits entre les économies d'énergie et les performances du réseau, la difficulté de l'optimisation multi-objectifs est de savoir comment faire un compromis raisonnable entre plusieurs mesures de performance.

3.5 Etude des différentes solutions existantes

IL existe différentes solutions qui ont été développé par les chercheurs scientifiques et parmi les auteurs de ces différentes solutions nous avons :

3.5.1 Solution 1 : Sallahi et M.St-Hilaire [4]

Sallahi et M. St-Hilaire [4] Proposent un modèle mathématique pour le problème de placement du contrôleur dans SDN, ce modèle détermine simultanément le nombre optimal, l'emplacement et le type de contrôleur (s) ainsi que les interconnexions entre tous les éléments du réseau.

1. Objectif de la solution :

Le modèle proposé a pour objectif de minimiser les coûts de planification de réseau tel que le coût d'installation des contrôleurs Cc (x), le coût de liaison des contrôleurs aux commutateurs Cl (v) et le coût de liaison des contrôleurs Ct (z).

2. Architecture utilisée :

Pour atteindre ces objectifs l'auteur de cette solution propose de respecter ces conditions suivantes :

Assurez-vous que chaque commutateur de chaque domaine a un seul lien relié au contrôleur du domaine. Et que chaque domaine est connecté à un contrôleur.

Assurez-vous que le nombre de commutateurs et de contrôleurs connectés à un contrôleur spécifique est inférieur au nombre de ports sur le contrôleur.

Assurez-vous que le nombre de paquets envoyés par chaque commutateur peut être traité par le contrôleur.

Assurez-vous que l'inventaire de chaque contrôleur n'est pas dépassé.

Assurez-vous que le lien choisi entre le contrôleur et le commutateur peut gérer la bande passante nécessaire au commutateur.

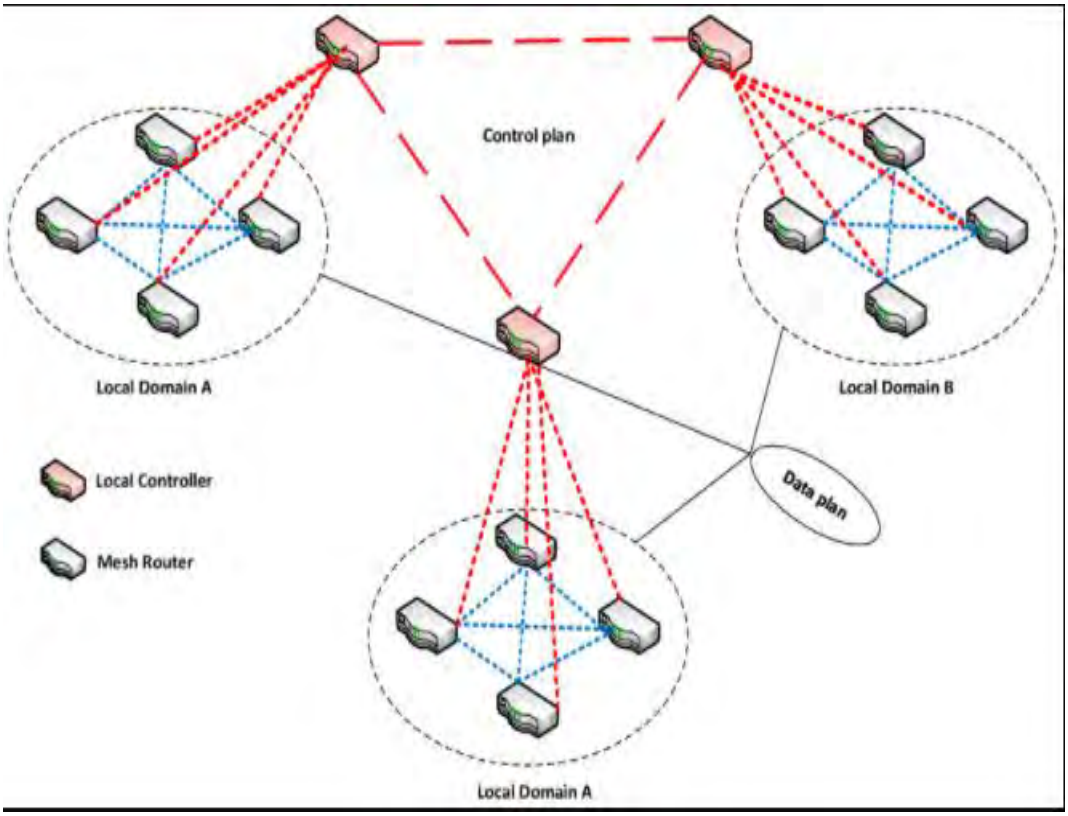


Figure 11: La topologie utilisée pour cette solution *[4]*

3. Approche utilisée :

La démarche utilisée pour résoudre ce problème est la suivante :

$$Minimize\ (Cc(x)+Cl(v)+Ct(z))$$

4. Analyse des résultats obtenus :

Les résultats de ce modèle sont interprétés en utilisant les expressions suivantes :
|S| : représente le nombre de commutateurs déployé dans le réseau
|P| : représente le nombre d'emplacement possible pour installer les contrôleurs
|P'| : représente le nombre moyen de contrôleurs installé par le solveur
|L'| : représente le nombre moyen de liens installés pour l'ensemble du réseau
Packets : représente le nombre total de paquets pour le réseau.
Cost ($) : représente le coût de la solution ($)

CPU (s) : est le temps mis par le solveur pour trouver la solution optimale.

Pour évaluer le modèle, le nombre de commutateurs dans l'ensemble S (c'est-à-dire |S|) varie de 10, 20, 30, 40, 50, 75, 100, 150 et 200. Pour chaque valeur de |S|, le nombre d'emplacements possibles pour installer les contrôleurs (c'est-à-dire |P|) varie de 10, 15 et 20. Ces 27 combinaisons sont générées dans une zone de 1 km x 1 km et nous permet de voir comment le temps et le coût du modèle d'optimisation sont affectés.

Les simulations sont exécutées sur un PC équipé de 2 processeurs Intel Xeon X5675 fonctionnant à 3,07 GHz avec une mémoire totale de 96 Go. Tous les résultats sont obtenus à partir de l'optimiseur CPLEX 12.5 [9]. Une limite de temps de 30 heures a été fixée et l'optimiseur n'a été autorisé à utiliser qu'un seul thread. Tous les autres paramètres ont été définis sur leurs valeurs par défaut.

Si plus d'un contrôleur existe (c'est-à-dire si |P'|>1) alors il y a un total de

$$|L'| = \frac{|P'|(|P'|-1)}{2} + |S|$$

liens pour chaque scénario.

Si nous commençons par analyser le temps CPU mis pour trouver les solutions, nous pouvons voir que pour une valeur fixe de |S|, le temps augmente généralement à mesure que le nombre d'emplacements potentiels pour les contrôleurs augmente. Par contre, ce n'est pas toujours le cas. Une raison pour laquelle l'optimiseur peut prendre moins de temps pour une plus grande taille d'entrée est que l'optimiseur utilise l'algorithme de branche. Différentes combinaisons et des heuristiques sont utilisées pour raccourcir l'espace combinatoire en trouvant la solution à la relaxation du nœud actuel. De même, on peut également voir que pour une valeur fixe de |P|, le temps augmente généralement à mesure que le nombre de commutateurs est augmenté. Sur les 108 instances différentes analysées, 11 cas (10%) n'étaient toujours pas résolus avant que le délai ne soit atteint. Cela indique que seuls les problèmes de petite taille peuvent être calculés dans un délai raisonnable. Nous avons également calculé l'intervalle de confiance à 95% et le plus grand intervalle se produit au problème 18 avec un intervalle de ±27,019 secondes. Cet intervalle large se produit parce que certaines des instances ont atteint la limite de temps et d'autres ont été résolues « rapidement ».

En termes de coût de la solution, nous pouvons faire deux observations. Tout d'abord, nous pouvons voir que pour une valeur fixe de |P|, le coût augmente linéairement avec une augmentation du nombre de commutateurs. Cela était en quelque sorte attendu car le réseau doit accueillir plus de commutateurs, nécessitant donc plus de contrôleurs et plus de liaisons.

3.5.1 Solution 2 : Y. Hu et al. [5]

Les auteurs de l'article [5] étudient le problème de positionnement du contrôleur du point de vue de la consommation d'énergie. Ce problème est modélisé comme un programme entier binaire (BIP).

1. Objectif de la solution

L'objectif de ce modèle est de minimiser la consommation d'énergie du réseau sous les contraintes du délai du chemin de contrôle et de charge du contrôleur.

2. Approche utilisée :

Le modèle utilisé pour résoudre ce problème est la suivante :

$$minimize \sum_{e=1}^{|E|} \rho e \; xe$$

Où *pe* est la consommation d'énergie de la liaison ***e***, et ***xe*** une variable aléatoire binaire indiquant si le lien ***e*** appartient au réseau de contrôle.
Cependant, le modèle BIP ne peut être utilisé que pour les réseaux à petite et moyenne échelle en raison de sa grande complexité. Pour les réseaux à grande échelle, un algorithme génétique est amélioré, appelé Improved Genetic Controller Placement Algorithm (IGCPA), et utilisée pour trouver une solution sous-optimale efficace.
Les auteurs prouvent que la consommation d'énergie augmente considérablement lors que l'utilisation d'un contrôleur augmente. Ils ont adopté les deux modèles BIP et IGCPA de consommation d'énergie afin de parvenir un équitable entre l'utilisation, la charge et la consommation d'énergie. Ils ont désactivé autant de liens que possibles en s'assurant que chaque commutateur a un chemin vers un contrôleur et que les charges de contrôleurs sont également équilibrées.

3. Analyse des résultats obtenus :

Les simulations sont effectuées à l'aide de Matlab. L'algorithme de Yen est utilisé pour calculer les K premiers chemins les plus courts, et l'optimisateur IBM ILOG CPLEX est adopté pour résoudre le modèle BIP. Dans IGCPA, choisir à chaque fois 3 paires de solutions parentales. Les simulations ont été exécutées sur un ordinateur équipé de 2 processeurs Intel Xeon E5-2430 à 8 cœurs, équipés de 16 Go de RAM.
Les résultats de la simulation montrent que :

- ✓ L'IGCPA peut trouver une solution proche de la solution optimale.
- ✓ L'algorithme heuristique n'utilise pas plus de 4 % de liens supplémentaires par rapport à la solution BIP lorsque tous les liens consomment la même énergie.
- ✓ Économies d'énergie allant jusqu'à 55 % pendant les heures creuses.

3.5.2 Solution 3 : F.J. Ros et al [6]

F.J. Ros et al [6] définissent un problème de placement de contrôleur tolérant aux pannes (FTCP).

1. Objectif de la solution :

L'objectif de cette solution est de déterminer le nombre de contrôleurs qui doivent être instanciés, de l'endroit où ils doivent être déployés et les noeuds du réseau qui sont contrôlés par chacun d'eux, afin d'obtenir une grande fiabilité dans l'interface sud entre les contrôleurs et les noeuds.

2. Approche utilisée :

Bien que la minimisation des latences entre chaque nœud et son contrôleur assigné constitue un aspect crucial du problème de placement du contrôleur, il existe de nombreux autres objectifs, éventuellement concurrents, qui doivent être pris en compte. Dans ce qui suit, les objectifs qui sont couverts dans ce travail sont présentés avec des exemples qui motivent leur nécessité dans différents cas d'utilisation. Le réseau Internet2 OS3E est utilisé comme exemple de topologie et le meilleur placement par rapport à la latence maximale entre les nœuds et les contrôleurs pour k = 5 contrôleurs, comme le montrent les travaux de Heller et al. [10] fait office de référence. La figure 12 affiche les performances de ce placement lorsqu'il est évalué par rapport à différentes mesures objectives et différentes conditions, par exemple, les latences et l'équilibre de la charge en présence de pannes de nœuds ou de contrôleurs.
La topologie (a) de la figure 12 illustre la latence entre chaque nœud et son contrôleur attribué lorsque plusieurs contrôleurs cessent de fonctionner. La couleur de chaque nœud indique sa latence par rapport au contrôleur de fonctionnement le plus proche. La couleur est normalisée avec le diamètre du graphique avec le vert représentant une latence de zéro, le jaune représentant une latence qui correspond à 50% du diamètre du graphique et le rouge indiquant une latence de 100%. Alors que la structure du contrôleur distribué affirme de faibles latences dans le cas sans panne, le scénario de panne illustré met en évidence le fait qu'en présence de pannes, la position de chaque contrôleur compte. Afin de mieux gérer les scénarios de ce type, le mécanisme d'optimisation des placements de contrôleurs résilients doit prendre en compte des scénarios de défaillance. Selon le cas d'utilisation spécifique, les latences moyennes et maximales peuvent être des mesures utiles.
La topologie (b) de la figure 12 montre le cas de deux nœuds de réseau défaillants en même temps, ce qui entraîne les deux phénomènes susmentionnés. Tout d'abord, le graphique est décomposé en deux composantes disjointes. Deuxième, la partie droite du graphique ne contient pas de contrôleur. Ainsi, les nœuds de cette partie du graphe perdent l'accès à toute fonctionnalité réalisée par le contrôleur. Ces nœuds sont représentés par des icônes de point d'interrogation⍰.

En supposant que les nœuds se connectent à leur contrôleur le plus proche, certains placements ont tendance à entraîner des affectations déséquilibrées, c'est-à-dire que certains contrôleurs fournissent des instructions pour beaucoup plus de nœuds que d'autres. La topologie (c) illustre l'aspect de déséquilibre du placement optimal de latence. Chaque nœud est coloré et formé en fonction du contrôleur auquel il est affecté. Alors que le contrôleur bleu est responsable de dix éléments de réseau, les contrôleurs vert et rouge doivent gérer seulement quatre nœuds, c'est-à-dire moins de la moitié.

De plus, nos enquêtes précédentes montrent que pour certaines implémentations de contrôleurs, le nombre et l'ordre des commutateurs connectés peuvent causer des injustices en ce qui concerne des aspects tels que les temps de configuration du flux des commutateurs. Ainsi, l'équilibrage de charge doit faire partie des critères de décision lors du choix d'un emplacement de contrôleur pour divers types de configurations SDN et NFV.

Cependant, une telle architecture nécessite également diverses formes de synchronisation d'état entre les contrôleurs individuels. Cela garantit une fonctionnalité adéquate en cas de panne et permet de prendre des décisions qui ne se limitent pas à une vue locale sur une partie du réseau. Par conséquent, un autre objectif de la tâche de placement de contrôleur est de maintenir une faible latence entre les contrôleurs afin de minimiser les temps de synchronisation. Une visualisation de cette mesure est fournie sur la figure 1d, où chaque contrôleur est coloré en fonction de la distance par rapport au contrôleur qui est le plus éloigné de lui. Pour la plupart des contrôleurs, le placement indiqué entraîne des latences maximales élevées vers d'autres contrôleurs, ce qui peut ne pas être acceptable pour certains cas d'utilisation. Par conséquent, la latence entre les contrôleurs fait partie de l'ensemble des objectifs analysés de POCO. De plus, la paire de latence inter-contrôleur et de latence nœud à contrôleur constitue un ensemble d'objectifs concurrents. Alors qu'un cluster serré de contrôleurs entraîne de faibles latences entre contrôleurs et des latences élevées entre les nœuds et les contrôleurs, une distribution spatialement étendue des contrôleurs conduit au contraire. Ces relations entre les objectifs sont la motivation pour l'analyse des placements optimaux de Pareto dans POCO, qui permet aux décideurs d'exprimer leurs préférences après avoir inspecté les placements possibles.

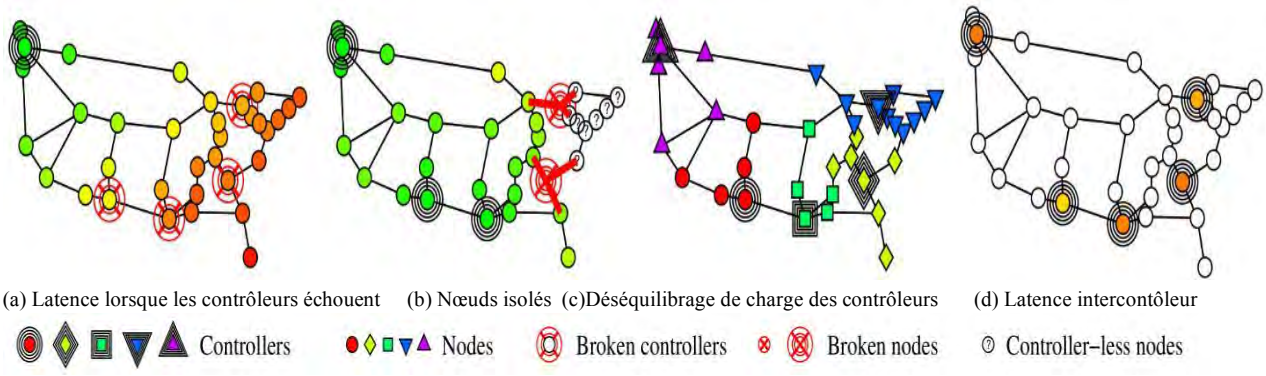


Figure 12 : Les Topologies utilisées pour cette solution [9]

3. Modèle Mathématique utilisé :

Pour réaliser ce travail les auteures développent un heuristique algorithme qui permet de calculer les emplacements pour répondre à cette fiabilité et de donner une limite supérieure sur le nombre de contrôleurs. Le modèle mathématique utilisé pour calculer la fiabilité est la suivante :

$$R(G^{'}, j_0, t) = 1 - \prod_{\forall \pi \epsilon \prod j_0 t} (1 - \prod_{\forall (u,v) \in \pi} P_{u,v})$$

Où R représente la fiabilité, Π $j0$, t l'ensemble des chemins disjoints entre $j0$ et t et Pu, est la probabilité opérationnelle.

L'algorithme a été appliqué sur un ensemble de 124 topologies de réseau accessibles au public. Il s'est exécuté dans un ordinateur Linux pour tous les scénarios simulés. Les contrôleurs sont lancés sur des machines virtuelles utilisant VirtualBox comme hyperviseur. Ces machines virtuelles contrôlent les topologies de réseau créées avec mininet.

4. Analyse des résultats obtenus après simulation :

Comme résultats, le nombre total de contrôleurs varie considérablement et est plus lié à la topologie du réseau qu'à la taille du réseau. Néanmoins, 8 contrôleurs ou moins sont suffisants pour 75% des cas les plus intéressants. En plus, chaque noeud est nécessaire pour se connecter à seulement 2 contrôleurs, qui fournissent généralement plus que le seuil de fiabilité requis.

3.5.3 Solution 4 : K. S. Sahoo et al [7]

K. S. Sahoo et al [7] proposent une méta-heuristique approche pour résoudre le problème de CPP dans SDN-WAN, cette approche fournit un mécanisme de sauvegarde transparent contre une défaillance de liaison unique avec un délai de communication minimal basé sur le modèle de capacité de survie. Les auteurs utilisent deux techniques méta-heuristiques basées sur la population telles que : PSO (Particle Swarm Optimization) et l'algorithme FireFly (FFA). Pour le CPP, trois métriques ont été prises en compte :

- La latence du contrôleur pour basculer,
- La latence inter-contrôleur
- La connectivité multi-chemins entre le commutateur et le contrôleur.

1. Objectif utilisé :

L'objectif est de trouver un placement de contrôleur tel qu'il minimise la latence moyenne entre le commutateur et le contrôleur π^{avglat} et entre contrôleurs $\pi^{avgiclat}$ et en même temps maximise le nombre moyen de chemins disjoints $\pi^{avgdispat^h}$ entre le contrôleur et les commutateurs.

2. Architecture et Approche utilisées :

Supposons qu'il y a trois contrôleurs déployés dans la topologie donnée et que la topologie est constituée de plusieurs 3 domaines.
Le Domaine (CD-1) est conçu comme une structure carrée. Pour le CD-1, les emplacements possibles du contrôleur peuvent être n'importe quel emplacement A, B, C ou D.
Tous les emplacements de ces positions (A vers C et D vers B) prennent au maximum une latence élevée entre le contrôleur et le commutateur. Cependant, si le contrôleur est situé en Z, alors le maximum la latence est réduite à moitié.
Ainsi, l'emplacement du contrôleur réduit de moitié la latence entre le commutateur et le contrôleur.
De même, si l'on considère le domaine 2 (CD-2), le contrôleur est placé à l'emplacement E. Pour le commutateur G, il existe trois chemins disjoints qui existent entre le commutateur et le contrôleur, à savoir G vers F vers E, G vers H vers O vers E et G vers E.
Et donc il existe deux chemins alternatifs pour le commutateur G qui sont disponibles en cas de défaillance de la liaison primaire (supposons G vers E est le lien principal), la communication du commutateur avec le contrôleur reste inchangée.
D'un autre côté, choisir un autre chemin joue un rôle important car le délai supplémentaire affecte la qualité de service de l'utilisateur final du réseau. Par conséquent, nous concluons qu'une bonne décision pour placer les contrôleurs peut avoir un impact positif majeur sur le réseau.

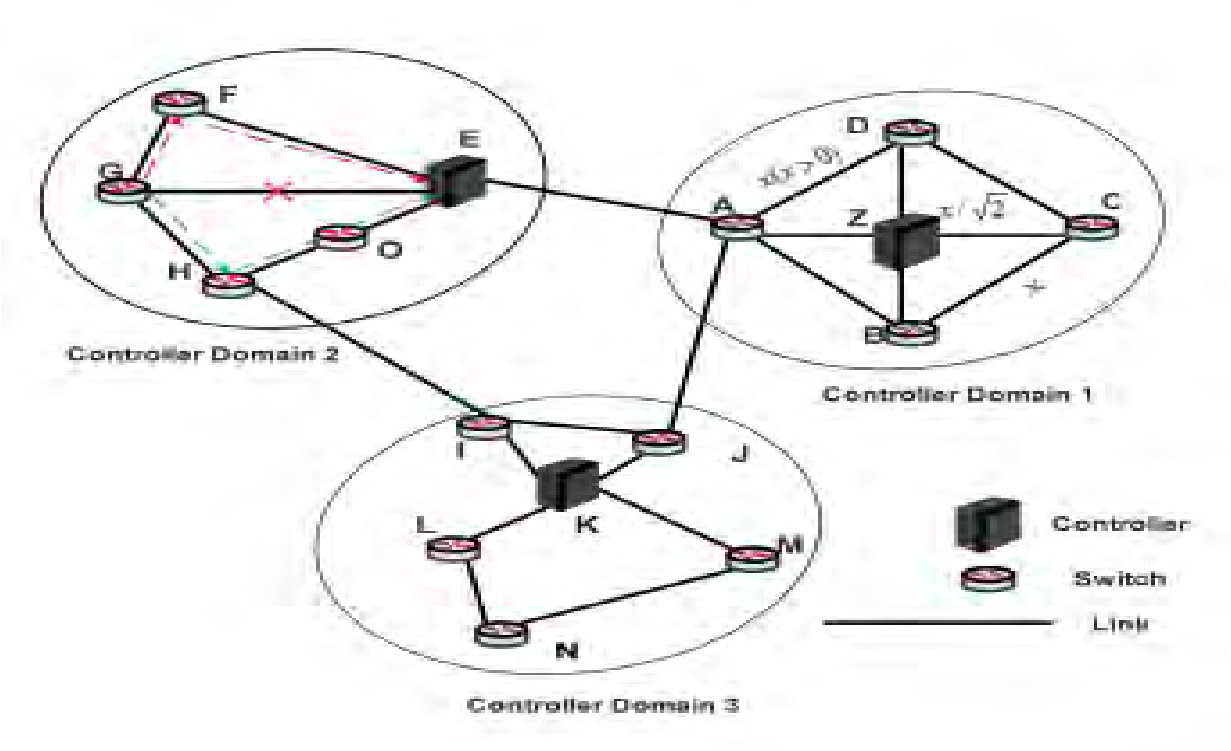


Figure 13 : Topologie utilisée pour cette solution [7]

3. Approche utilisée :

La démarche utilisée pour résoudre ce problème est la suivante :

$$f\left(C_i\right) = min\left\{\frac{\pi^{avglat} + \pi^{avgiclat}}{\pi^{avgdispath}}\right\}$$

Où f(C_i) représente l'emplacement des contrôleurs. Les performances des algorithmes (PSO et FFA) sont évaluées sur un ensemble de topologies de réseau accessibles au public (TopologyZoo) afin d'obtenir le nombre optimal de contrôleurs et de positions de contrôleurs. Ces algorithmes sont écrits dans la version MatLab R2014a et fonctionnent sur un système doté d'Intel Core i5 avec des processeurs 4 coeurs et de 8 Go de RAM avec Ubuntu 13.10 du système d'exploitation 64 bits. Les résultats sont obtenus en exécutant les algorithmes 100 fois.

4. Résultat de la simulation :

Le résultat de la simulation montre que :

- ✓ L'approche proposée minimis e le délai moyen lors d'une panne de liaison unique par rapport aux travaux existants.
- ✓ Bien que les deux algorithmes aient des résultats acceptables, l'algorithme FFA (CPP_FFA) a produit le résultat optimal avec un temps de calcul moindre

3.6 ABCP : Approche basée sur le cluster de la densité

Liao et al [8] propose une méthode appelée placement de contrôleur basé sur la densité (DBCP), qui utilise un algorithme de regroupement de commutateurs basé sur la densité pour diviser le réseau en plusieurs sous-réseaux. Étant donné que les commutateurs sont étroitement connectés dans le même sous-réseau et ont moins de connexions aux commutateurs d'autres sous-réseaux, il suffit de déployer un contrôleur par sous-réseau. Dans DBCP, la taille de chaque sous-réseau peut être déterminée par la capacité du contrôleur déployé. De plus, le nombre optimal de contrôleurs est obtenu sur la base du clustering basé sur la densité.

La division du réseau permet de :

- Déterminer le nombre optimal de contrôleurs qui doivent être placé dans un réseau arbitraire.
- Obtenir des sous-réseaux relativement stables. Cela signifie qu'ils peuvent réduire la probabilité de déployer un contrôleur dans des endroits à haut risque.
- Le problème de placement de plusieurs contrôleurs peut être simplifié comme le problème de placement d'un contrôleur, qui est facile à résoudre.
- Minimisation de la latence maximale

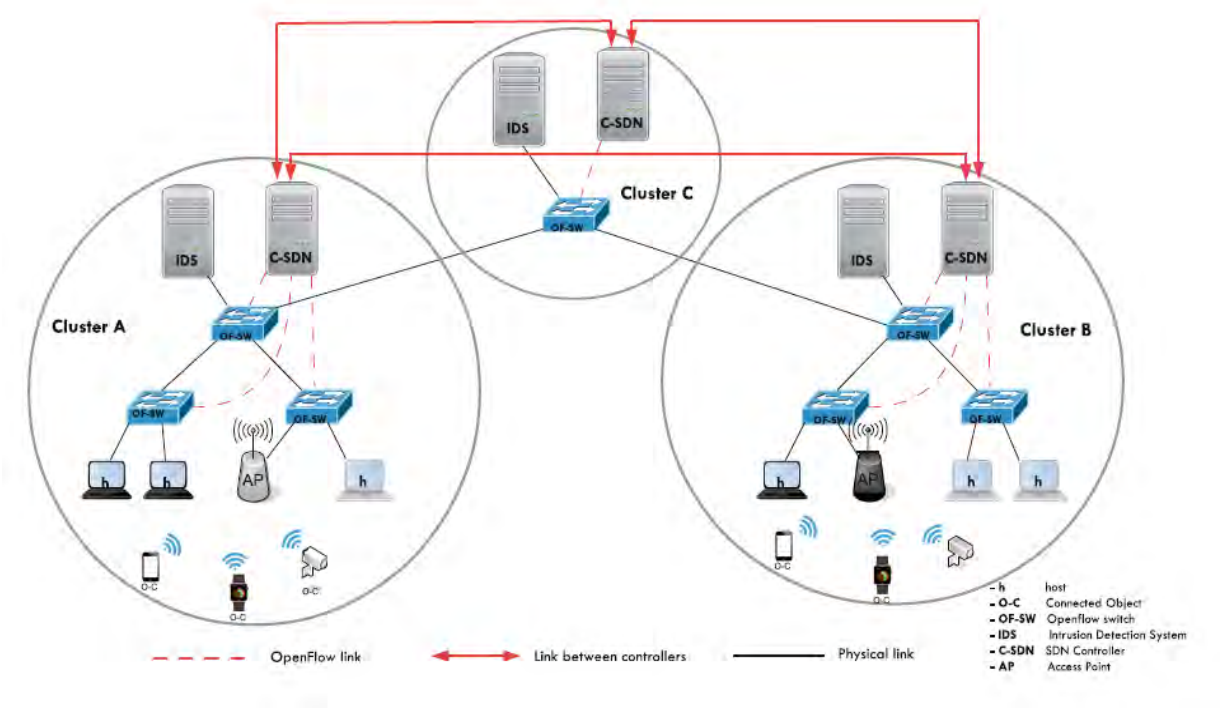


Figure 14 : Architecture basée sur un cluster de réseau programmable [8]

3.6.1 Calcule de la densité

Dans DBCP, nous avons divisé le réseau en regroupant les commutateurs. Pour chaque commutateur si, nous calculons une quantité : sa densité locale ρi. Cette quantité dépend uniquement des distances entre les commutateurs.
La densité locale ρi du commutateur si est définie comme :

$$\rho_i = \sum_j X\left(d_{ij} - d_c\right)$$

Où dij représente la distance entre les commutateurs di et dj. Le dc est une distance seuil.
Seule la distance entre les commutateurs inférieure à dc sera considérée comme un commutateur fermé. Et la fonction χ est définie comme :

$$X(x) = \begin{cases} 1, & \text{si } x < 0\ ; \\ 0, & \text{sinon.} \end{cases}$$

Fondamentalement, ρi est égal au nombre de commutateurs dont les distances au commutateur si sont inférieures à dc. Pour les réseaux à grande échelle, dc est défini comme 0,3 fois le diamètre du graphe.

Algorithme 1 : le processus d'analyse.

1. input : G = (S, L) , dc
2. k = 0
3. for s in S :
4. ρs = get number of nodes within distance (s , dc, G)
5. end for

Tableau 3.2 **le processus d'analyse [8]**

3.6.2 Trouver le commutateur le plus proche avec une densité plus élevée.

Dans les réseaux physiques, la charge des contrôleurs influence également les performances du réseau. Un DBCP capacitif (CDBCP) peut être utilisé pour résoudre le problème CCPP. Compte tenu de la capacité des contrôleurs, le principal problème est que les clusters ne peuvent pas être aussi grands qu'ils devraient l'être. En supposant que chaque contrôleur θ a une capacité L (θ). La charge du commutateur de control (controlling switch) s est indiquée par l (s). L (θ) et l (s) peuvent être considérés comme le nombre de recherches de contrôleurs, de schémas d'installation de flux proactifs et réactifs, ou leur combinaison. La condition de contrainte est alors définie comme suit :

$$L(\theta) \geq \sum_{s \epsilon S(\theta)} l(s),\ \forall \theta \epsilon \Theta$$

Où S (θ) représente la collection de commutateurs du sous-réseau qui sont contrôlés par le contrôleur θ, et Θ représente tous les contrôleurs.

Si un cluster contient suffisamment de charge de contrôle et ne peut étendre aucun commutateur en fonction des contraintes, les autres commutateurs prennent en compte les autres clusters adjacents ou deviennent un nouveau cluster. Afin de conserver l'attribut des commutateurs dans le même cluster sont étroitement connectés, nous considérons prioritaire d'affecter les commutateurs étroitement connectés dans un même cluster, et les commutateurs qui sont à la frontière du cluster en tant que commutateurs candidats pour remplir les clusters qui peuvent augmenter la taille.

Pour trouver les commutateurs à la frontière du cluster, nous calculons une autre quantité pour chaque commutateur : son indexation limite σ, qui représente l'incertitude d'un commutateur appartenant à différents clusters. Pour chaque commutateur, il peut au commutateur lié avec une densité égale ou supérieure. Si les densités de voisins de densité égale ou supérieure sont similaires, le σ du commutateur est plus élevé, ce qui signifie qu'il s'agit d'un commutateur de frontière. σi pour le commutateur si est calculé comme suit:

$$td_i = \sum_j^{s_j \epsilon UL(s_i)} \rho_j$$

$$\sigma_i = \sum_j^{s_j \epsilon UL(s_i)} \frac{\rho_j}{td_i} \log_{|UL(s_i)|} \frac{\rho_j}{td_i}$$

Où UL (si) représente l'ensemble des commutateurs de densité supérieure et égale liés à si.

3.6.3 regroupements (clustering)

Dans notre affectation de clustering, les commutateurs avec une valeur inférieure de σ ont la priorité. Le commutateur est affecté au même cluster que son voisin le plus proche avec une densité plus élevée, sauf si la charge des commutateurs de ce cluster dépasse la capacité du contrôleur. S'il n'y a aucun cluster à affecter, le commutateur devient un nouveau cluster avec lui-même. L'algorithme 2 montre les détails du clustering du réseau en tenant compte de la capacité.

Algorithme 2 : Processus de clustering pour problème de CCPP.

. input : G = (S, L) , ρ
2. Input: the capacity of controllers L (θ).
3. for s in S :
4. σs = get the borderline based on neighbor s (s ,g).
5. End for
6. Sort S according to σ in ascending order.
7. Each switch forms a cluster N (s).
8. for s in S:
9. Get the neighbor switches with higher or equal ρ of s as UL (s).
10. Sort UL (s) according to ρ in descending order.
11. For s' in UL (s)
12. Get the current cluster N (s) and N (s') that contain s and s' respectively
13. If L (θ) ≥ $\Sigma_{si \in (N(s)+N(s'))}$ l(Si):
14. s belongs to the cluster N (s')
15. Break
16. end if
17. end for
18. end for

Tableau 3.3 le processus de regroupement (clustering) [8]

3.6.4 Emplacement du Controleur

Après avoir divisé le réseau, nous devons placer les contrôleurs pour tous les sous réseaux. Notez que dans chaque sous-réseau, un seul contrôleur est déployé. Nous utilisons la combinaison de solutions optimales pour chaque sous-réseau pour approximer la solution optimale pour l'ensemble du réseau. Le placement des contrôleurs peut être décidé en fonction de différentes fonctions objectives. Plus généralement, la latence et la fiabilité sont utilisées.

3.6 Tableau comparatif des différentes solutions liées aux problème de placement du contrôleur

Auteurs de la Solution	Année	Latence	Fiabilité	Coût	Multi-Object	Tolérant aux pannes
Sallahi et M. St-Hilaire [4]	2015	Minimise que la latence moyenne entre le commutateur et le contrôleur	Faible	Faible	Optimal	Non
Y. Hu et al. [5]	2016	Minimise que la latence moyenne entre le commutateur et le contrôleur	Faible	Faible	Optimal	Non
F.J. Ros et al [6]	2016	Non prise en charge	Elevée	Elevé	Moins optimal	Oui
K. S. Sahoo et al [7]	2018	Minimise la latence moyenne entre le commutateur et le contrôleur et la latence inter-contrôleurs	Elevée	Moyen	Optimal	Oui
Cluster de Liao et al [8]	2017	Minimise la latence moyenne entre le commutateur et le contrôleur, la latence inter-contrôleurs et la pire latence entre commutateur et contrôleur	Très Elevée	Moyen	Plus Optimal	Oui

Tableau 2 : Tableau Comparatif des différentes solutions étudiées

3.8 Conclusion

Dans ce chapitre, nous avons présenté un aperçu des travaux connexes et de la littérature sur le problème de placement des contrôleurs et son impact sur les performances dans le SDN. Ensuite, nous avons fourni une description détaillée d'une solution proposée, qui utilise un algorithme de regroupement de commutateurs (clustering) basé sur la densité pour diviser le réseau en plusieurs sous-réseaux.

Chapitre 4 : Mise en œuvre de la solution basée sur la méthode de cluster des contrôleurs SDN

4.1 Introduction

Dans le chapitre précédent, nous avons fait une revue des travaux existants qui traitent le problème de placement des contrôleurs dans SDN et nous avons présenté l'approche de cluster en détail. Dans ce chapitre nous implémentons cette solution basée sur la méthode de clustering permettant le partage de charge et l'optimisation du placement des contrôleurs.
Nous allons donc expliquer dans ce chapitre comment créer un cluster de contrôleurs (trois contrôleurs pour notre cas) afin de synchroniser l'état des contrôleurs et de gérer la haute disponibilité. Et par la suite nous allons faire l'analyse et l'interprétation des résultats sur les performances telle que la latence et la fiabilité du réseau.

4.2 Etude comparative des outils de simulations

Plusieurs outils de simulation et d'émulation comme Mininet ont été développés pour implémenter les réseaux basés sur OpenFlow dans une seule machine, et aussi pour tester les nouvelles applications.

- Mininet (nous présentons l'installation et les commandes d'utilisation dans section suivante) : est un émulateur d'environnement SDN, développé par l'université de Stanford, et peut être utilisé pour construire des réseaux virtuels et estimer les métriques de performance pour différentes topologies avec des configurations différentes.
- NS3 : une autre option est d'utiliser le simulateur NS dans sa version 3 qui supporte la version 0.8.9 du protocole OpenFlow dans son environnement.
- EstiNet : est un simulateur propriétaire supportant la version 1.3.2 du protocole OpenFlow. Il propose deux modes de fonctionnement, simulation et émulation, et peut simuler des milliers de switchs. EstiNet consomme moins de ressources et il est considéré comme étant le plus performant et le plus évolutif.
- OMNET++ : est un projet open source dont le développement en C++ a commencé en 1992 par Andras Vargas à l'université de Budapest. En 2013, Omnet++ support le protocole OpenFlow v1.0.

Outils de simulation	OpenSource	Versions openflow supportées	Langage utilisée
Mininet	Oui	Toutes versions	Python
NS3	Oui	V0.8.9	C++, python
EstiNet	Oui	V1.3.2	Python
Omnet++	Oui	V1.0	C++

Tableau 3 : Tableau comparatif des outils de simulation

4.3 Présentation de l'émulateur de réseau mininet

Mininet est un émulateur de réseau, il permet de créer une topologie réseau qui se compose d'un ensemble de hosts, de switchs, de contrôleurs et liens virtuels. Il fournit la capacité de créer des hôtes, les commutateurs et contrôleurs via :

- Ligne de commande,
- Interface interactive,
- Script Python.

4.3.1 Configuration utilisée

Afin de bien visualiser le fonctionnement du SDN, nous avons choisi d'utiliser une machine virtuelle téléchargée depuis :
https://github.com/mininet/mininet/releases/download/2.2.2/mininet-2.2.2-170321-ubuntu-14.04.4-server-amd64.zip
et accessible via VirtualBox, elle utilise le système d'exploitation Linux (Ubuntu-14.04.4).

4.3.1 Fonctionnement de base de mininet

Mininet permet avec un simple jeu de commande de réaliser des réseaux virtuels en utilisant la commande **mn**.

4.3.2 Création d'une topologie avec la ligne de commande

Mininet fournit, en plus de la topologie « minimal », la topologie « single », « linear » et « tree » comme montre dans la figure 15. Pour charger l'une de ces topologies, on utilise l'option **« --topo »**, Par exemple : $ sudo mn –topo single « single » tout court donne la même topologie que minimal, c'est-à-dire créer un hôte relie avec un switch, mais on peut ajouter également comme argument un chiffre, qui indique le nombre d'hôtes à créer.
Par exemple : $ sudo mn –topo single,4
Si nous souhaitons personnaliser une topologie déjà créée, nous appliquons une des commandes citées ci-dessous :

- Ajouter un switch: self.AddSwitch('s1')
- Ajouter un hôte : self.AddHost('h1')
- Ajouter un lien : self.addLink(h1, s1)

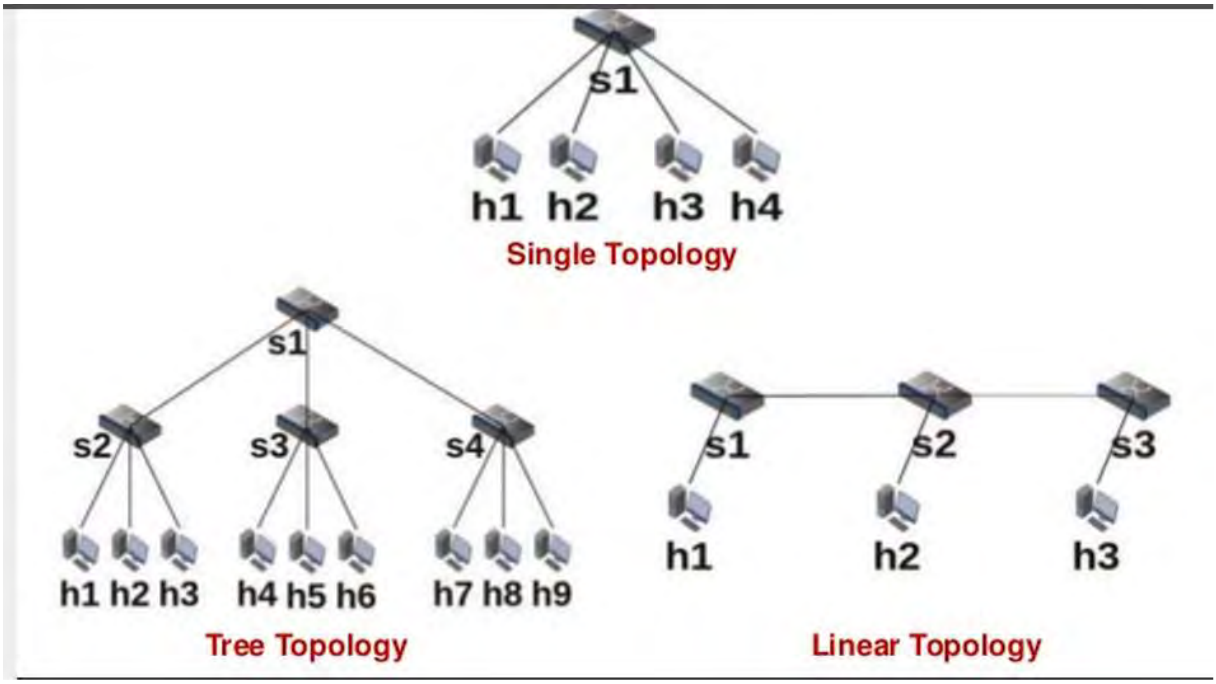


Figure 15 : Différente types de topologie réseau

Apres la configuration, nous devons tester si les paquets sont routés correctement avec la commande **ping** qui est un bon moyen pour verifier la connectivité : Mininet> h1 ping h2 ou bien Mininet> pingall et pour analyser les règles insérées dans chaque commutateur, nous utilisons la commande dpctl : Mininet> dpctl dump-flows, et d'une façon générale, pour accéder à toutes les commandes utilisées sous Mininet, il suffit de taper : Mininet> help

4.4 Architecture Utilisée

En se basant sur notre solution de clustering qui permet non seulement d'optimiser le nombre de contrôleur à déployer dans le réseau mais aussi de permettre l'équilibrage de charge entre les différents contrôleurs, nous allons implémenter une topologie composée de trois contrôleurs Opendaylight et de sept commutateurs. Cette topologie est composée de deux systèmes autonomes (domaines) dont chaque domaine est contrôlé par un contrôleur Opendaylight. Le troisième contrôleur est connecté avec le switch de backbone. Dans chaque switch d'accès c'est-à-dire (S3, S4, S5 et S6), nous avons connecté deux machines hôtes. Les trois contrôleurs sont synchronisés grâce au cluster et nous allons analyser l'impact de l'emplacement de ces trois contrôleurs sur les performances telle que la latence, la fiabilité.

Donc en résumé nous allons utiliser les paramètres suivants pour la simulation :

- trois contrôleurs OpenDayLight ;
- sept Commutateurs Openflow ;
- 8 machines hôtes.

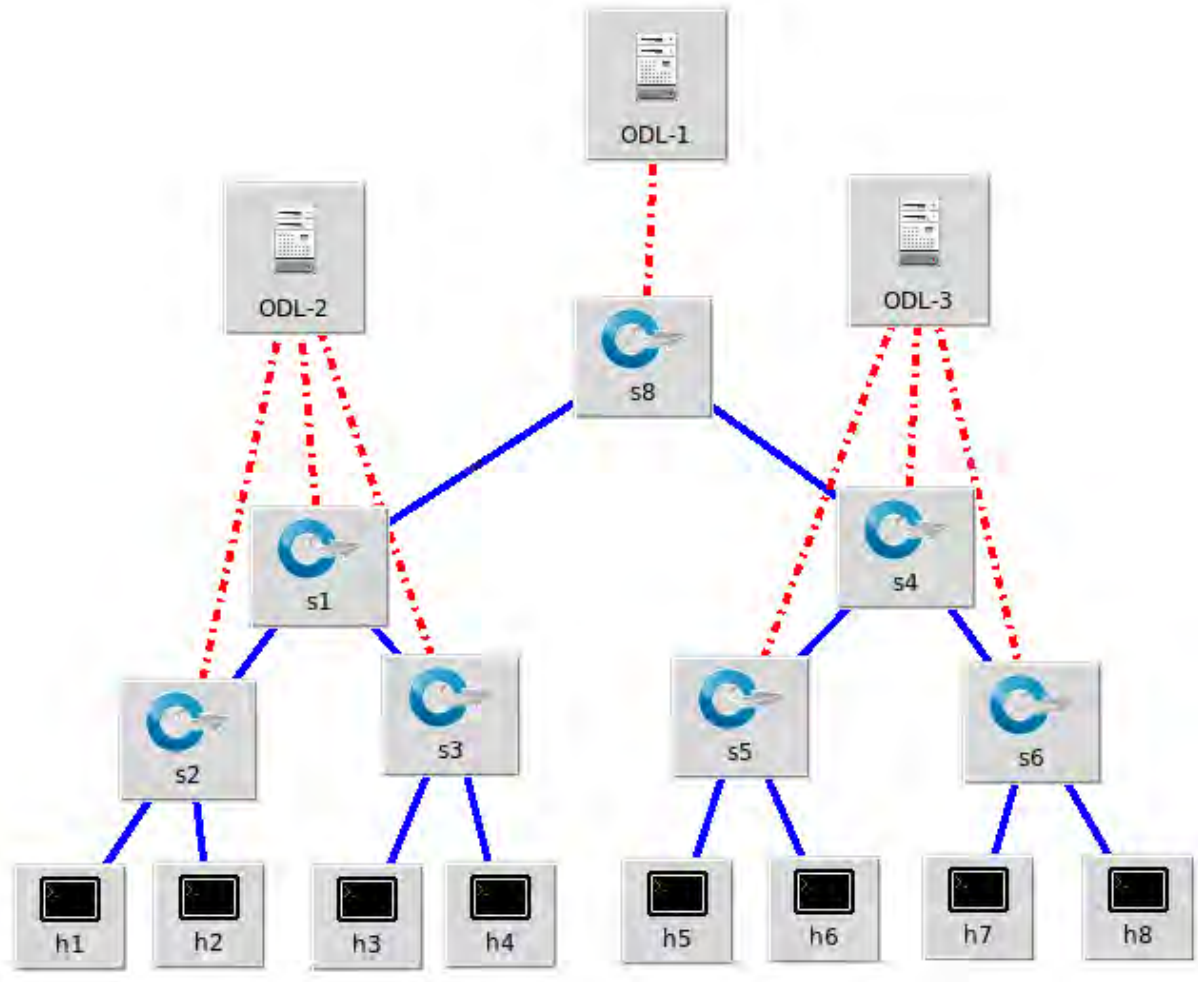


Figure 16 : Architecture Utilisée pour la simulation

4.4 Prérequis ou Méthode d'implémentation

Dans le cas de notre simulation, nous allons installer quatre machines virtuelles répartis comme suit :

- Une machine virtuelle de mininet que nous avons importé sur le lien cité précédemment.
- Trois machines virtuelles constituées chacune d'un système Ubuntu 16.04.1, d'une ram de 2Go et d'un stockage de 20 Go. Dans chacune de ces trois machines nous allons y installer OpenDayLight pour avoir trois contrôleurs OpenDayLight qui vont se synchroniser pour former un cluster.

Chacune de ces quatre machines sera constituée de deux interfaces réseaux dont la première interface va permettre d'aller vers l'internet et la deuxième interface va être un réseau privé d'hôte créé dans VirtualBox qui sera une interface de bouclage sur l'ordinateur hôte qui peut être utilisée pour connecter les quatre machines virtuelles entre eux et à l'ordinateur hôte.

Et après avoir synchronisé les trois contrôleurs OpenDayLight nous allons créer une topologie réseau constituée de deux domaines dans la machine virtuelle mininet dans laquelle chaque domaine sera contrôlé par un contrôleur (OpenDayLight 2 et OpenDayLight 3) et l'autre contrôleur (OpenDayLight 1) sera connecté par un commutateur de backbone.

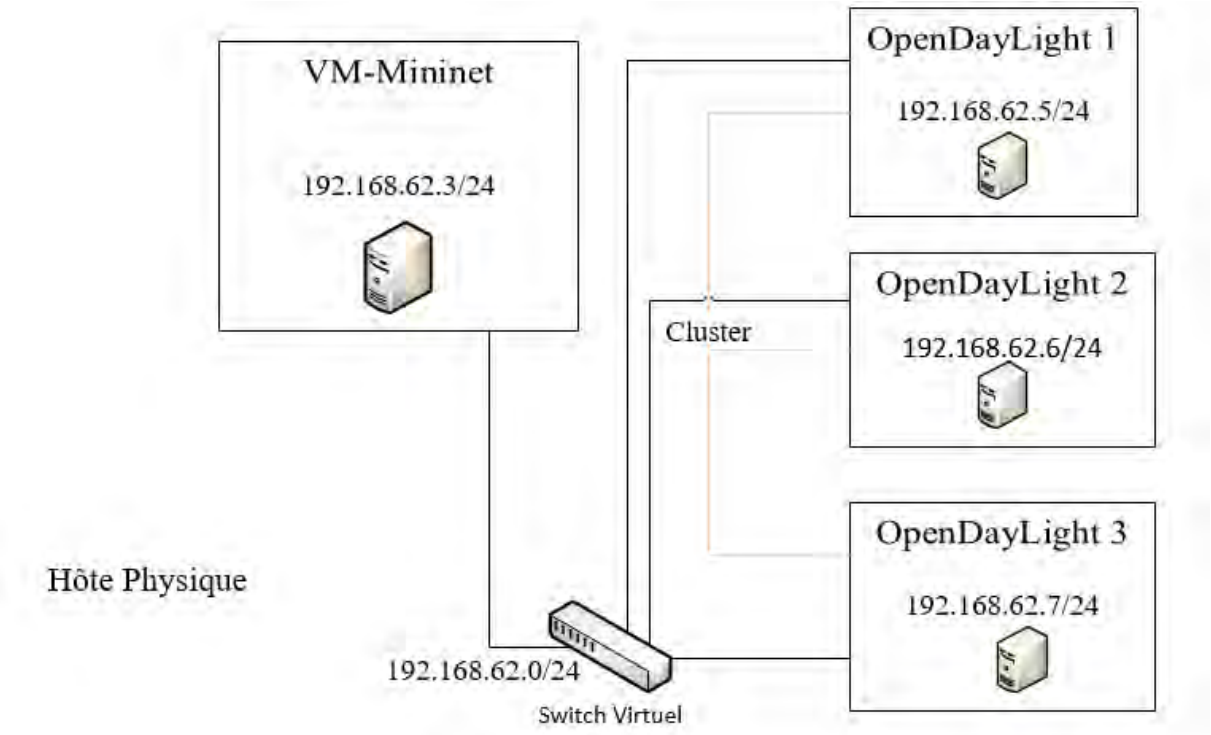


Figure 17 : Prérequis utilisés pour l'implémentation

4.5 Installation des prérequis

4.5.1 Installation de mininet

Lors de linstallation de la machine virtuelle mininet, nous allons utiliser les paramètres par défaut comme le montre la figure 18. Mais nous pouvons les modifier si nous le souhaitons. Après quelques minutes, nous verrons la VM *Mininet que* vous avez importée dans la fenêtre VirtualBox.

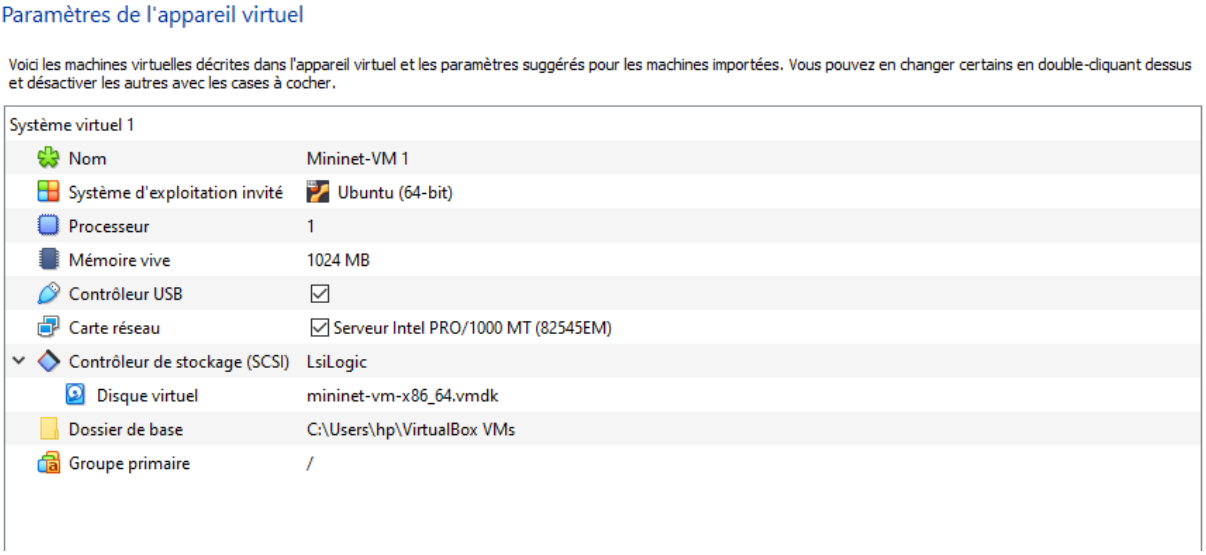


Figure 18 : Importation de la machine virtuelle mininet

Pour utiliser Mininet de la manière recommandée qui est aussi la manière adéquate pour notre implémentation comme le montre la figure 17, nous devons créer une interface réseau (réseau privé hôte) dans VirtualBox. Cela crée une interface de bouclage sur l'ordinateur hôte que nous allons utiliser pour connecter nos quatre machines virtuelles entre eux et à l'ordinateur hôte. Nous allons utiliser la configuration manuelle pour fixer notre adresse ip qui le 192.168.62.1/24 comme le montre la figure 19

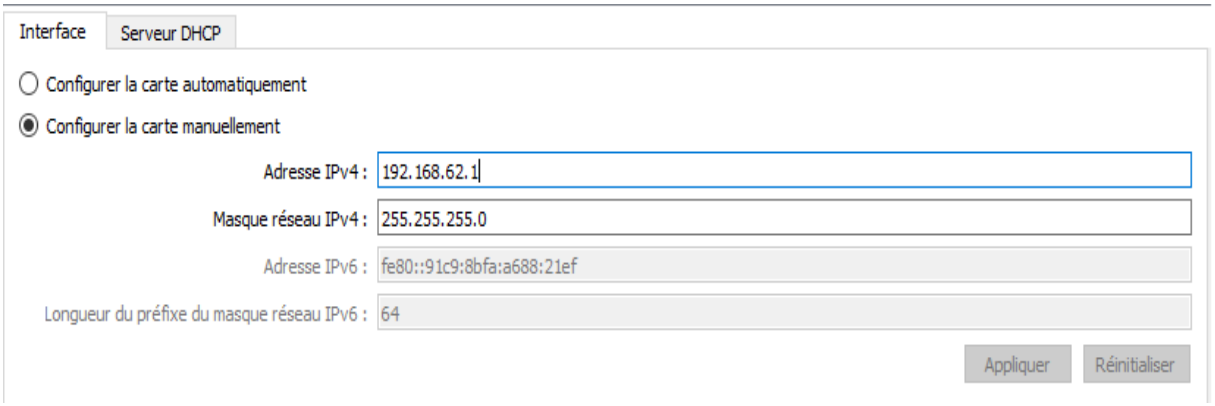


Figure 19 : Création de l'interface réseau privé hôte

Le serveur DHCP est activé sur l'interface et nous voyons que la limite d'adresse inférieure est 192.168.62.3/24 comme le montre la figure 20. Ainsi, nous savons donc que l'adresse IP de l'interface virtuelle connectée au réseau hôte uniquement sur la machine virtuelle se verra attribuer cette adresse IP.

Figure 20 : Activation de la méthode dhcp pour l'interface réseau privé hôte

Ainsi pour une utilisation future, nous devons noter les informations suivantes :

- Adresse réseau de l'hôte uniquement : 192.168.62.0/24
- Adresse IP de l'hôte sur le réseau hôte uniquement : 192.168.62.1/24
- Adresse IP de l'interface virtuelle de la machine virtuelle sur le réseau hôte uniquement : 192.168.56.101/24

La machine virtuelle a déjà une interface définie. Comme le montre la figure 20, sur l'onglet « Adapter 1 », nous voyons cette interface configurée comme un NAT. Cela permettra à la machine virtuelle de se connecter à Internet. Mais pour utiliser Mininet, nous avons toujours besoin d'un moyen pour la machine virtuelle de se connecter directement à l'ordinateur

hôte. Nous devons donc ajouter un autre adaptateur virtuel et le connecter à l'interface réseau hôte uniquement (voir la figure 22) que nous avons créée précédemment.

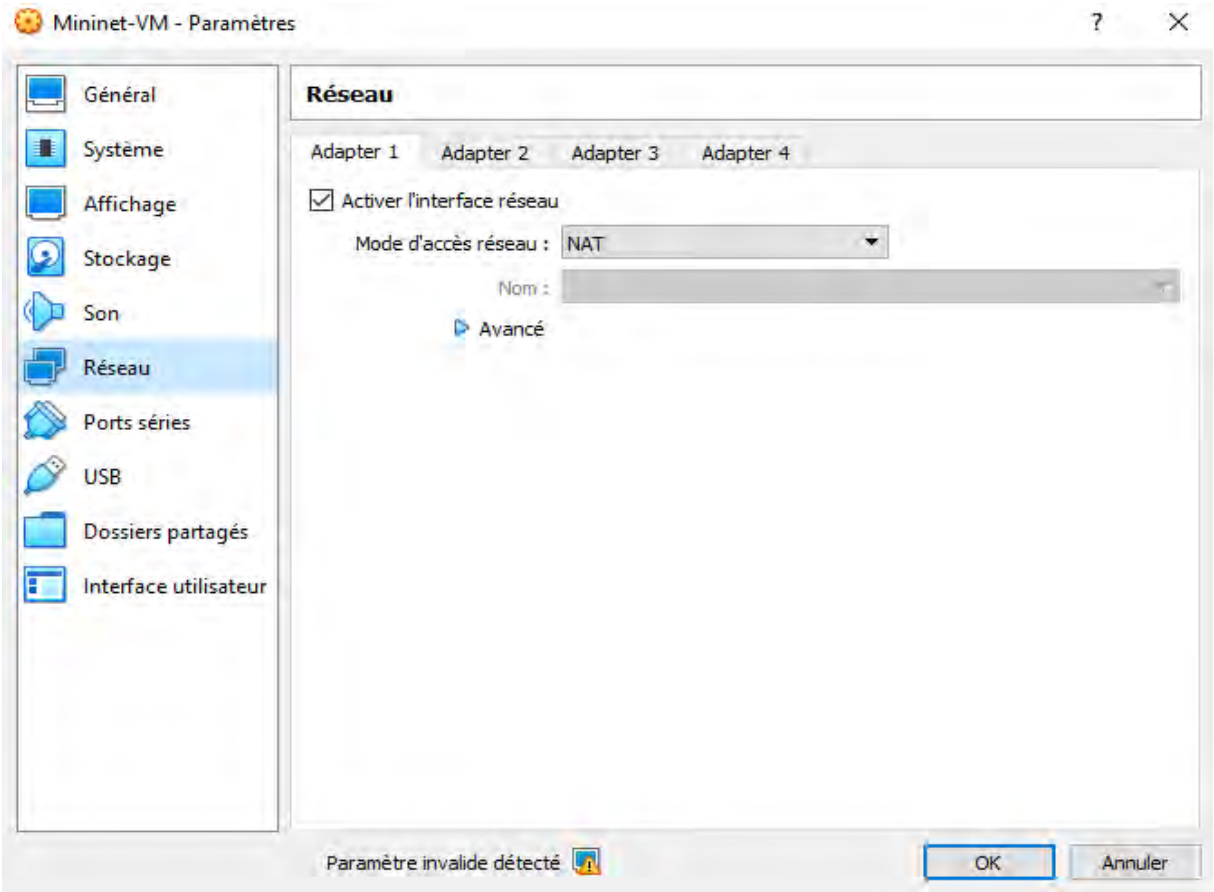


Figure 21 : L'interface réseau nattée par défaut

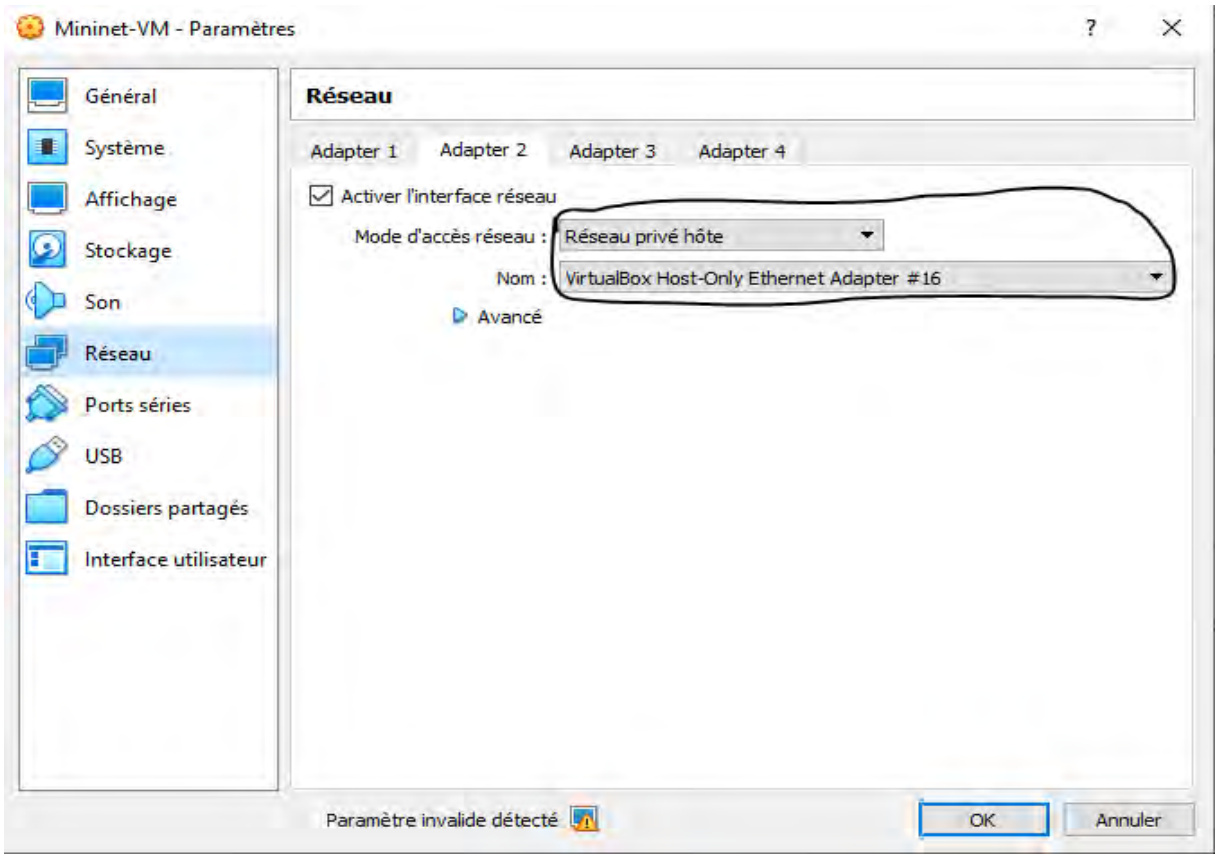


Figure 22 : L'activation de l'interface réseau privé hôte

Après démarrage de la machine virtuelle mininet, nous verrons que nos deux cartes réseaux sont bien installé et que notre interface réseau privé hôte qui est le eth1 a bien pris l'adresse 192.168.62.3.

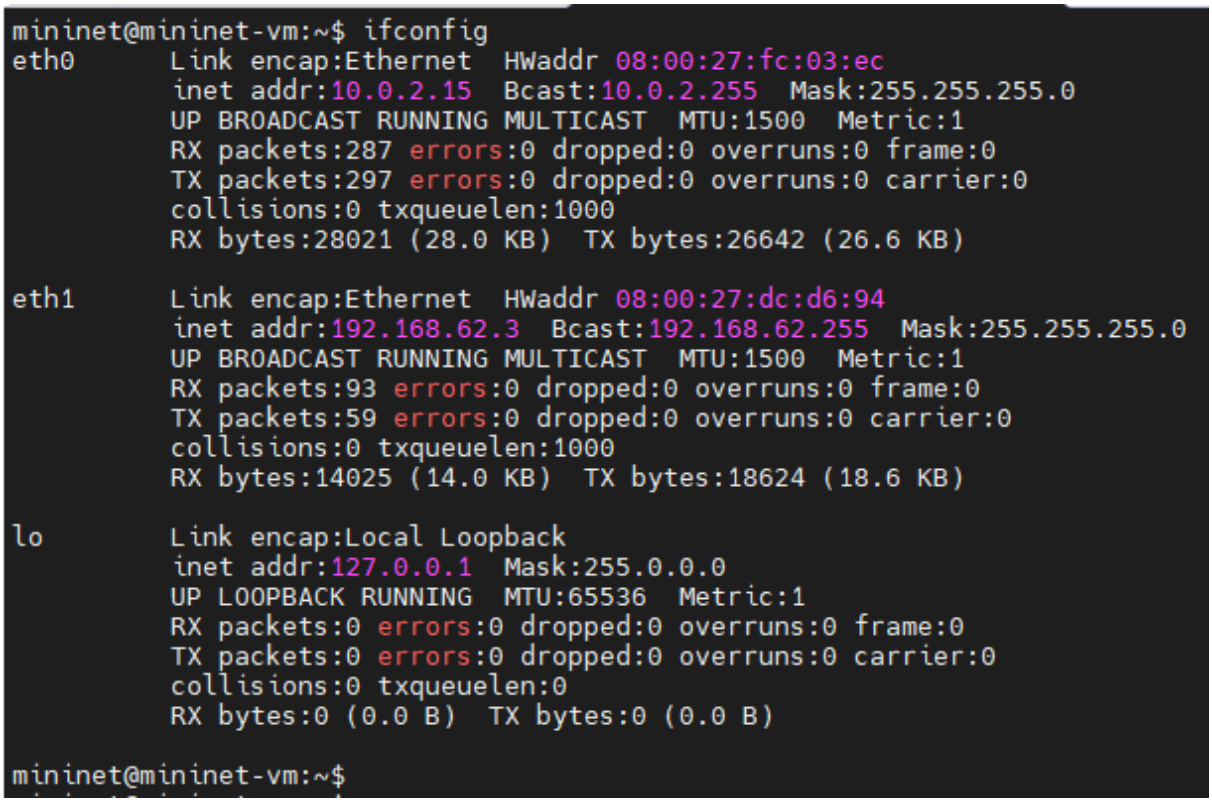


Figure 23 : L'adresse assignée par l'interface réseau privé hôte et l'adresse nattée

4.5.2 Installation d'OpenDayLight

Après installation de nos trois serveurs qui vont héberger chacun OpenDayLight et qui sont des serveurs constitués d'un système Ubuntu 16.04.1, de 2Go de mémoire ram et de 20Go de stockage, nous allons configurer la première interface réseau comme un Nat pour permettre à la machine d'aller vers internet et la deuxième interface sera la même interface réseau privé hôte créée précédemment et qui va permettre aux trois serveurs d'OpenDayLight, à mininet ainsi qu'à la machine hôte de se communiquer.

C'est pourquoi nous voyons dans la figure 24 notre serveur 1 dont l'interface enp0s8 a une adresse ip 192.168.62.5 qui lui est assignée et ceci grâce à notre serveur dhcp que nous avons configuré précédemment sur VirtualBox.

De même, nous voyons l'interface enp0s8 de notre serveur 2 a une adresse ip 192.168.62.6 qui lui est assignée (voir la figure 25).

Ainsi que l'interface enp0s8 de notre serveur 3 qui a aussi une adresse ip 192.168.62.7 qui lui est assignée (voir la figure 26).

```
root@memoire:~# ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:c8:bf:20
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fec8:bf20/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:19745 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7933 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:26559178 (26.5 MB)  TX bytes:489839 (489.8 KB)

enp0s8    Link encap:Ethernet  HWaddr 08:00:27:b8:fc:98
          inet addr:192.168.62.5  Bcast:192.168.62.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:feb8:fc98/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:156 errors:0 dropped:0 overruns:0 frame:0
          TX packets:104 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:20367 (20.3 KB)  TX bytes:18776 (18.7 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:160 errors:0 dropped:0 overruns:0 frame:0
          TX packets:160 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:11840 (11.8 KB)  TX bytes:11840 (11.8 KB)

root@memoire:~#
```

Figure 24 : Adresse assignée par l'interface réseau privé hôte au contrôleur 1

```
root@memoire2:~# ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:02:b3:70
          inet adr:10.0.2.15  Bcast:10.0.2.255  Masque:255.255.255.0
          adr inet6: fe80::a00:27ff:fe02:b370/64 Scope:Lien
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Packets reçus:30 erreurs:0 :0 overruns:0 frame:0
          TX packets:105 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          Octets reçus:6688 (6.6 KB) Octets transmis:9628 (9.6 KB)

enp0s8    Link encap:Ethernet  HWaddr 08:00:27:34:1f:53
          inet adr:192.168.62.6  Bcast:192.168.62.255  Masque:255.255.255.0
          adr inet6: fe80::a00:27ff:fe34:1f53/64 Scope:Lien
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Packets reçus:197 erreurs:0 :0 overruns:0 frame:0
          TX packets:85 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          Octets reçus:23186 (23.1 KB) Octets transmis:16842 (16.8 KB)

lo        Link encap:Boucle locale
          inet adr:127.0.0.1  Masque:255.0.0.0
          adr inet6: ::1/128 Scope:Hôte
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          Packets reçus:160 erreurs:0 :0 overruns:0 frame:0
          TX packets:160 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1
          Octets reçus:11840 (11.8 KB) Octets transmis:11840 (11.8 KB)

root@memoire2:~#
```

Figure 25 : Adresse assignée par l'interface réseau privé hôte au contrôleur 2

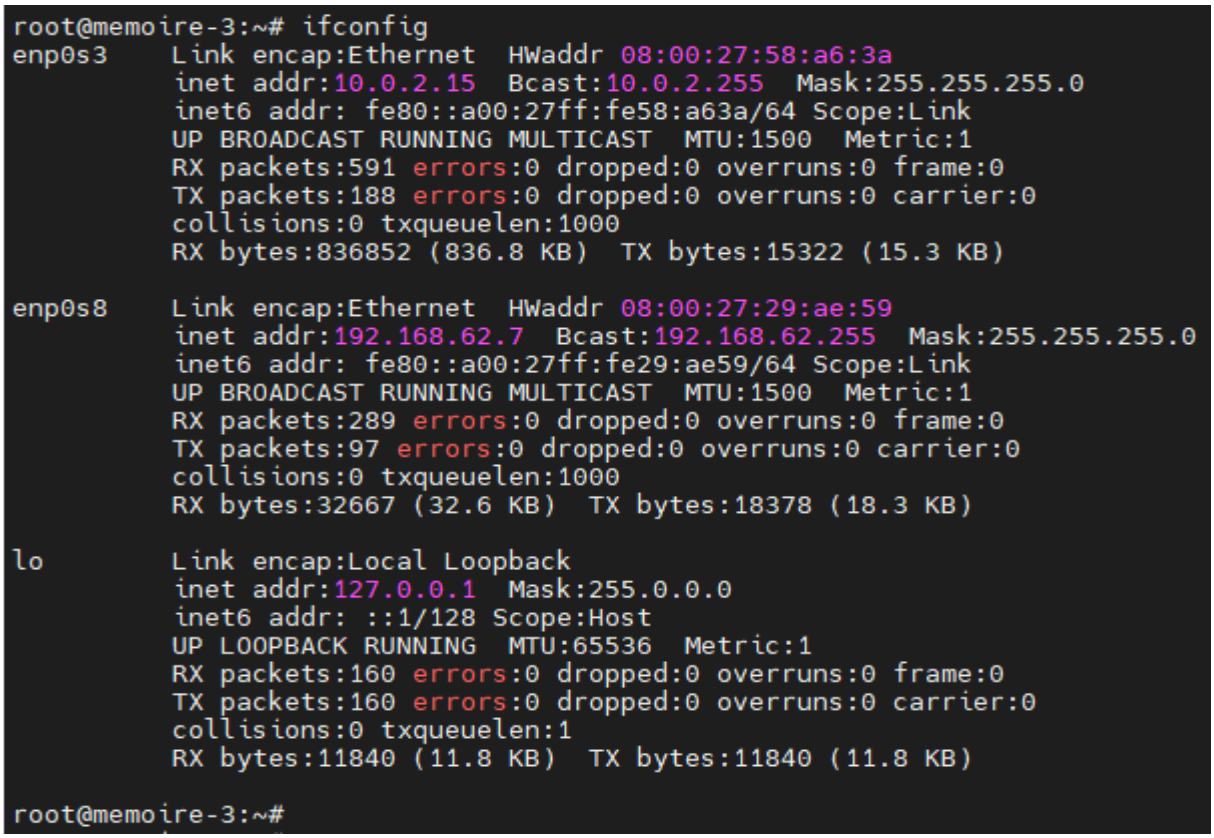


Figure 26 : Adresse assignée par l'interface réseau privé hôte au contrôleur 3

Maintenant que nos trois machines virtuelles sont prêtes à héberger OpenDayLight, nous allons commencer l'installation d'OpenDaylight dans chacune des trois machines virtuelles.

- Installation de java :

Puisque le contrôleur OpenDaylight SDN est un programme Java, commençons d'abord par installer l'environnement d'exécution de java avec les commandes suivantes :

```
sudo apt-get update
sudo apt-get install default-jre-headless
```

Après exécution de ces commandes ci-dessus, nous devons définir la variable d'environnement JAVA_HOME. Et pour le faire, modifions le fichier *bashrc* avec la commande suivante :

```
nano ~/.bashrc
```

Puis ajoutez la ligne suivante au fichier *bashrc :*

```
export JAVA_HOME=/usr/lib/jvm/default-java
```

Et exécutons le fichier avec la commande : `source ~/.bashrc`

- Installation d'OpenDayLight

Maintenant l'installation s'est terminée. Il ne nous reste qu'à télécharger le logiciel OpenDayLight dans chacune de nos trois machines virtuelles Ubuntu 16.04.1 à partir du site web d'OpenDayLight. Et nous utilisons la commande **wget** pour télécharger le fichier tar.

```
Wget
https://nexus.opendaylight.org/content/groups/public/org/opendaylight/integration/distribution-karaf/0.4.0-Beryllium/distribution-karaf-0.4.0-Beryllium.tar.gz
```

Après avoir téléchargé le fichier tar, nous allons le décompresser en utilisant la commande tar :

tar -xvf distribution-karaf-0.4.0-Beryllium.tar.gz

Cela crée un dossier nommé distribution-karaf-0.4.0-Beryllium qui contient le logiciel OpenDaylight et les plugins.
OpenDaylight est emballé dans un conteneur karaf. Karaf est une technologie de conteneur qui permet aux développeurs de mettre tous les logiciels requis dans un seul dossier de distribution.
Cela facilite l'installation ou la réinstallation d'OpenDaylight en cas de besoin, car tout est dans un seul dossier.

- Démarrage d'OpenDayLight :

Pour exécuter OpenDaylight, exécutons la commande karaf dans le dossier de distribution du package.

cd distribution-karaf-0.4.0-Beryllium
./bin/karaf

Maintenant, le contrôleur OpenDaylight est en cours d'exécution.

Figure 27 : Démarrage d'OpenDayLight sur le contrôleur 1

- Installation des fonctionnalités d'OpenDayLight :

Ensuite, nous allons installer l'ensemble minimum de fonctionnalités requises pour tester OpenDaylight et l'interface graphique d'OpenDaylight grâce à la commande suivante :

opendaylight-user@root> feature:install odl-restconf odl-l2switch-switch odl-mdsal-apidocs odl-dlux-all :

Dont chaque terme a la fonctionnalité définit ci-dessous :
odl-restconf : permet d'accéder à l'API RESTCONF
odl-l2switch-switch : fournit des fonctionnalités réseau similaires à celles d'un commutateur Ethernet

odl-mdsal-apidocs : permet d'accéder à l'API Yang
odl-dlux-all : interface utilisateur graphique OpenDaylight

OpenDayLight est maintenant bien installé dans notre serveur Linux. Et nous allons faire ce travail pour l'ensemble des trois serveurs qui seront nos contrôleurs OpenDayLight.

- L'interface utilisateur graphique d'OpenDaylight :

Ouvrons un navigateur sur notre système hôte et entrons l'URL de l'interface utilisateur OpenDaylight (DLUX UI). Il s'exécute sur la VM OpenDaylight, donc l'adresse IP est 192.168.62.5 et le port défini par l'application est 8181 :

Donc, l'URL est : http://192.168.62.5:8181/index.html
Le nom d'utilisateur et le mot de passe par défaut sont tous deux admin

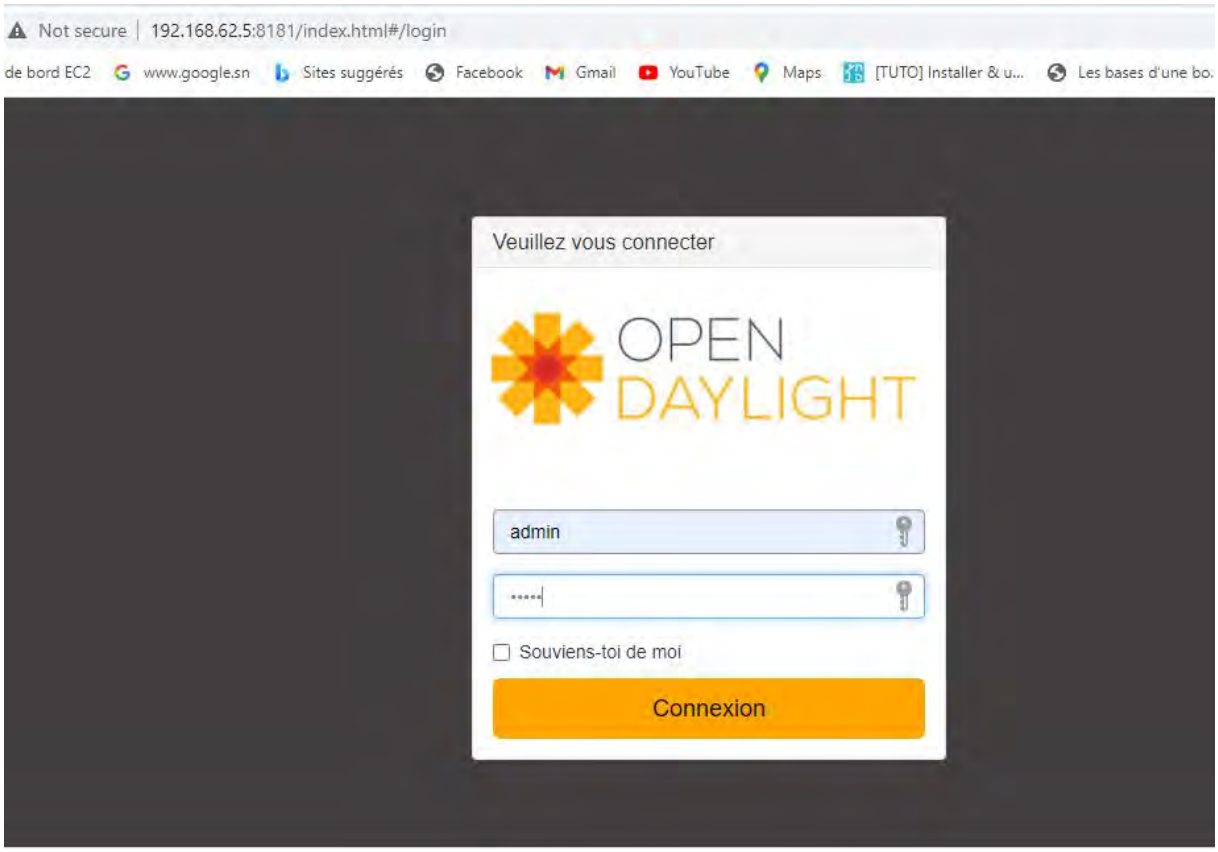


Figure 28 : L'interface de connexion d'OpenDayLight

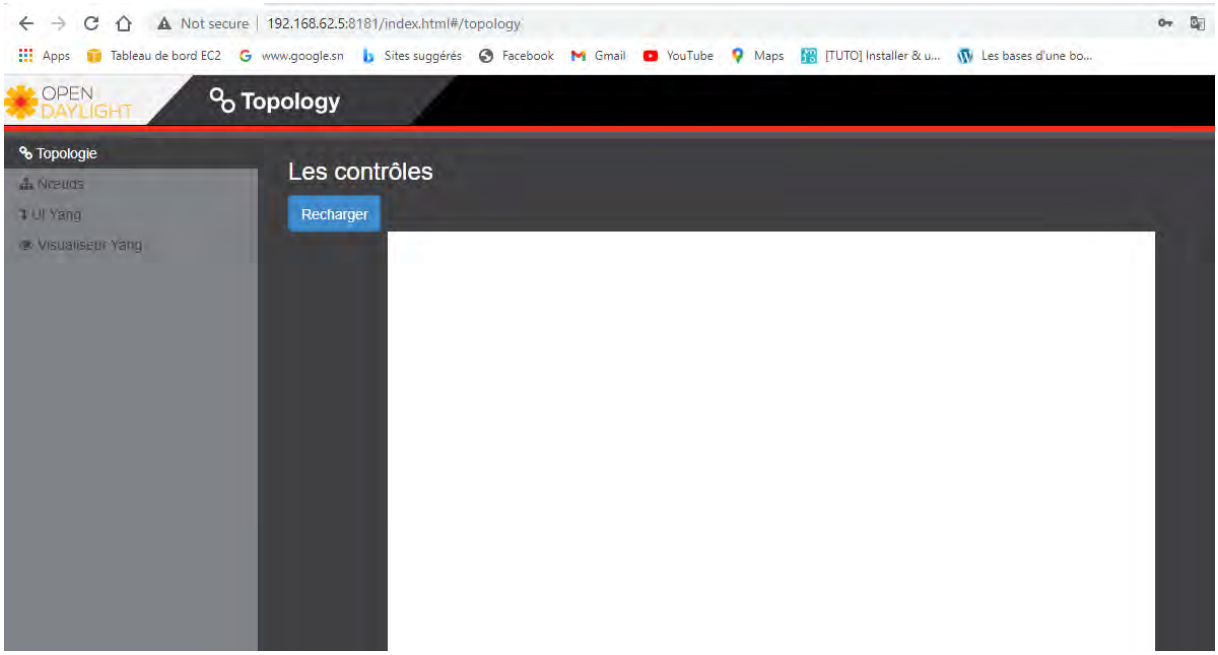


Figure 29 : L'interface graphique d'OpenDayLight

4.6 Création du cluster composé de trois contrôleurs OpenDayLight

4.6.1 Présentation du clustering

Le clustering est un mécanisme qui permet à plusieurs processus et programmes de fonctionner ensemble comme une seule entité. Par exemple, lorsque vous recherchez quelque chose sur google.com, il peut sembler que votre demande de recherche soit traitée par un seul serveur Web. En réalité, votre demande de recherche est traitée par de nombreux serveurs Web connectés dans un cluster. De même, vous pouvez avoir plusieurs instances d'OpenDaylight travaillant ensemble comme une seule entité.

Les avantages du clustering sont :

- Mise à l'échelle : si vous avez plusieurs instances d'OpenDaylight en cours d'exécution, vous pouvez potentiellement faire plus de travail et stocker plus de données qu'avec une seule instance. Vous pouvez également diviser vos données en petits morceaux (fragments) et soit distribuer ces données sur le cluster, soit effectuer certaines opérations sur certains membres du cluster.
- Haute disponibilité : si plusieurs instances d'OpenDaylight sont en cours d'exécution et que l'une d'entre elles plante, les autres instances seront toujours opérationnelles et disponibles.
- Persistance des données : vous ne perdrez aucune donnée stockée dans OpenDaylight après un redémarrage manuel ou une panne

4.6.2 Configuration du cluster

La configuration du cluster se commence d'abord sur l'enregistrement des membres du cluster sur le fichier configure_cluster.sh du répertoire bin de la distribution OpenDayLight. Et l'ensemble de ces enregistrements des membres du cluster vont aussi s'enregistrer sur les fichiers akka.conf et module-shards.conf qui sont des scripts qui contiennent l'ensemble des membres du cluster

Voici la syntaxe à suivre pour configurer le fichier configure_cluster.sh :

```
bin/configure_cluster.sh <index> <seed_nodes_list>
```

- Index : Entier compris entre 1..N, où N est le nombre de nœuds de départ. Cela indique quel nœud de contrôleur (1..N) est configuré par le script.
- seed_nodes_list: Liste des nœuds d'amorçage (adresse IP), séparés par une virgule ou un espace.

L'adresse IP à l'index fourni doit appartenir au membre exécutant le script. Nous allons exécuter ce script sur l'ensemble des nœuds de départ, tout en conservant la même liste de nœuds de départ et en faisant varier l'index de 1 à N.

Nos configurations sur la figure 30 montrent la déclaration des membres du cluster avec le contrôleur avec l'adresse ip 192.168.62.5 qui est le membre 1 du cluster, le contrôleur avec l'adresse ip192.168.62.6 qui est le membre 2 du cluster et en fin le contrôleur avec l'adresse ip 192.168.62.7 qui est le membre 3 du cluster.

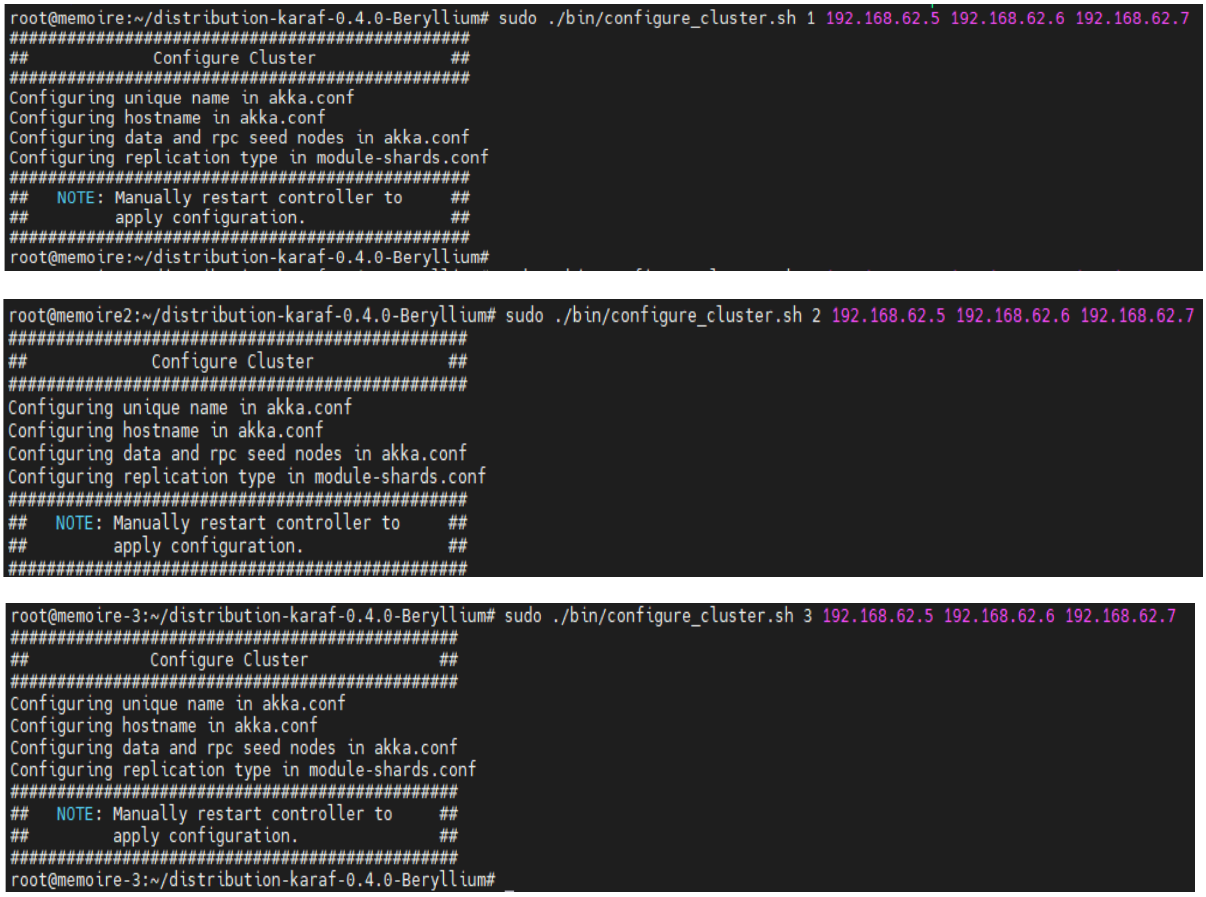


Figure 30 : Déclaration des membres du cluster

Pour vérifier que nos trois contrôleurs OpenDayLight sont exécutés en tant que cluster, nous allons ouvrir les fichiers .conf suivants :

- configuration/initial/akka.conf
- configuration/initial/module-shards.conf

Les figures 31, 32 et 33 montrent que dans le fichier de configuration (configuration/initial/akka.conf), nous apportons des modifications en recherchant chaque instance des lignes où il y'a « hostname » et nous remplaçons l'adresse ip par l'adresse de la machine dans laquelle ce fichier réside et OpenDayLight s'exécutera. La figure 31 représente le fichier de configuration (configuration/initial/akka.conf) du contrôleur numéro 1 et qui a comme rôle membre-1 du cluster. La figure 32 représente celui du contrôleur numéro 2 et qui est désigné comme le membre-2 du cluster. Et en fin la figure 33 représente le fichier de configuration (configuratio/initial/akka.conf) du troisième contrôleur qui est désigné comme membre-3 du cluster.

Aussi comme le montrent toujours les figures 31, 32 et 33, tous les adresses ip des trois contrôleurs membre du cluster doivent être enregistrées dans l'instance « seed-nodes ».

```
    default-mailbox {
      # When not using a BalancingDispatcher it is recommended that we use the SingleConsumerOnlyUnboundedMailbox
      # as it is the most efficient for multiple producer/single consumer use cases
      mailbox-type="akka.dispatch.SingleConsumerOnlyUnboundedMailbox"
    }
  }
  remote {
    log-remote-lifecycle-events = off
    netty.tcp {
      hostname = "192.168.62.5"
      port = 2550
      maximum-frame-size = 419430400
      send-buffer-size = 52428800
      receive-buffer-size = 52428800
    }
  }

  cluster {
    seed-nodes = ["akka.tcp://opendaylight-cluster-data@192.168.62.5:2550",
                  "akka.tcp://opendaylight-cluster-data@192.168.62.6:2550",
                  "akka.tcp://opendaylight-cluster-data@192.168.62.7:2550"]

    seed-node-timeout = 12s

    auto-down-unreachable-after = 30s

    roles = ["member-1"]
  }
```

Figure 31 : Configuration du fichier akka.conf du contrôleur 1

```
    default-mailbox {
      # When not using a BalancingDispatcher it is recommended that we use the SingleConsumerOnlyUnboundedMailbox
      # as it is the most efficient for multiple producer/single consumer use cases
      mailbox-type="akka.dispatch.SingleConsumerOnlyUnboundedMailbox"
    }
  }
  remote {
    log-remote-lifecycle-events = off
    netty.tcp {
      hostname = "192.168.62.6"
      port = 2550
      maximum-frame-size = 419430400
      send-buffer-size = 52428800
      receive-buffer-size = 52428800
    }
  }

  cluster {
    seed-nodes = ["akka.tcp://opendaylight-cluster-data@192.168.62.5:2550",
                  "akka.tcp://opendaylight-cluster-data@192.168.62.6:2550",
                  "akka.tcp://opendaylight-cluster-data@192.168.62.7:2550"]

    seed-node-timeout = 12s

    auto-down-unreachable-after = 30s

    roles = ["member-2"]
  }

  persistence {
```

Figure 32 : Configuration du fichier akka.conf du contrôleur 2

```
GNU nano 2.5.3                    File: configuration/initial/akka.conf

    default-mailbox {
      # When not using a BalancingDispatcher it is recommended that we use the SingleConsumerOnlyUnboundedMailbox
      # as it is the most efficient for multiple producer/single consumer use cases
      mailbox-type="akka.dispatch.SingleConsumerOnlyUnboundedMailbox"
    }
  }
  remote {
    log-remote-lifecycle-events = off
    netty.tcp {
      hostname = "192.168.62.7"
      port = 2550
      maximum-frame-size = 419430400
      send-buffer-size = 52428800
      receive-buffer-size = 52428800
    }
  }

  cluster {
    seed-nodes = ["akka.tcp://opendaylight-cluster-data@192.168.62.5:2550",
                  "akka.tcp://opendaylight-cluster-data@192.168.62.6:2550",
                  "akka.tcp://opendaylight-cluster-data@192.168.62.7:2550"]

    seed-node-timeout = 12s

    auto-down-unreachable-after = 30s

    roles = ["member-3"]
  }
```

Figure 33 : Configuration du fichier akka.conf du contrôleur 3

Ouvrons maintenant le fichier configuration/initial/module-shards.conf de chaque contrôleur et mettons à jour les répliques afin que chaque partition soit répliquée sur les trois nœuds. Comme le montre la figure 34

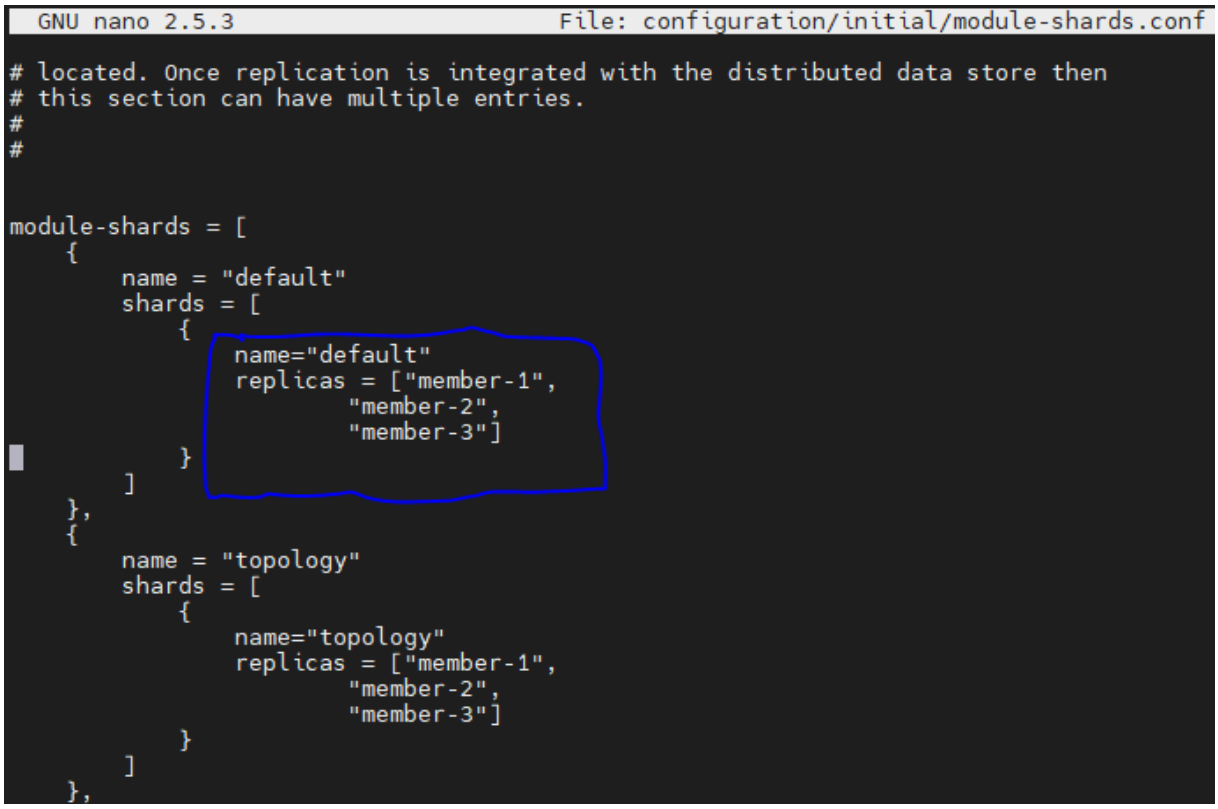


Figure 34 : Configuration du fichier module-shards.conf

Arrivé à ce stade, nous constatons que les configurations du cluster sont terminées et maintenant il ne reste qu'à créer une topologie sur mininet dont les nœuds vont être contrôlés par les contrôleurs de notre cluster que nous venons de créer.

4.7 Création de la topologie et analyse des résultats

Après avoir créé notre topologie sur miniedit qui notre architecture présentée précédemment dans la figure 17, nous avons sauvegardé le fichier qui est un fichier python sous le nom de controllers2.py dans la machine virtuel mininet. C'est pourquoi les figures 35 et 36 désignent des portions du code python créé lors de la création de notre topologie. Et évidemment nous voyons que les adresses assignées précédemment aux différents contrôleurs sont aussi les adresses que nous voyons dans le code source python ce qui est tout à fait logique. Car ce fichier d'écrit le code source python de l'ensemble de la topologie : les nœuds utilisés, les liaisons entre les nœuds, et les adresses pour chaque nœud.

```
#!/usr/bin/python

from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='192.168.62.0/24')

    info( '*** Adding controller\n' )
    ODL1=net.addController(name='ODL1',
                      controller=RemoteController,
                      ip='192.168.62.5',
                      protocol='tcp',
                      port=6633)

    ODL3=net.addController(name='ODL3',
                      controller=RemoteController,
                      ip='192.168.62.7',
                      protocol='tcp',
                      port=6633)

    ODL2=net.addController(name='ODL2',
                      controller=RemoteController,
```

Figure 35 : Portion du code source de notre topologie

```
                    protocol='tcp',
                    port=6633)

    info( '*** Add switches\n')
    s3 = net.addSwitch('s3', cls=OVSKernelSwitch)
    s6 = net.addSwitch('s6', cls=OVSKernelSwitch)
    s2 = net.addSwitch('s2', cls=OVSKernelSwitch)
    s8 = net.addSwitch('s8', cls=OVSKernelSwitch)
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch)
    s5 = net.addSwitch('s5', cls=OVSKernelSwitch)
    s4 = net.addSwitch('s4', cls=OVSKernelSwitch)

    info( '*** Add hosts\n')
    h2 = net.addHost('h2', cls=Host, ip='192.168.62.11', defaultRoute=None)
    h3 = net.addHost('h3', cls=Host, ip='192.168.62.12', defaultRoute=None)
    h1 = net.addHost('h1', cls=Host, ip='192.168.62.10', defaultRoute=None)
    h6 = net.addHost('h6', cls=Host, ip='192.168.62.16', defaultRoute=None)
    h4 = net.addHost('h4', cls=Host, ip='192.168.62.13', defaultRoute=None)
    h5 = net.addHost('h5', cls=Host, ip='192.168.62.15', defaultRoute=None)
    h8 = net.addHost('h8', cls=Host, ip='192.168.62.18', defaultRoute=None)
    h7 = net.addHost('h7', cls=Host, ip='192.168.62.17', defaultRoute=None)

    info( '*** Add links\n')
    net.addLink(s1, s2)
    net.addLink(s1, s3)
    net.addLink(s1, s8)
    net.addLink(s8, s4)
    net.addLink(s4, s5)
    net.addLink(s4, s6)
    net.addLink(h1, s2)
    net.addLink(h2, s2)
    net.addLink(h3, s3)
    net.addLink(h4, s3)
```

Figure 36 : Portion du code source de notre topologie

Nous allons maintenant exécuter le code source de notre topologie qui est sauvegardé dans le fichier controllers2.py. Et nous voyons que mininet a bien exécuté le code en créant l'ensemble des composants de notre topologie que nous allons voir dans chacun de nos trois contrôleurs puisqu'ils se forment un cluster. En plus les pings montrent que les différents hôtes se communiquent correctement.

```
root@mininet-vm:~/mininet/examples# sudo python controllers2.py
*** Creating (reference) controllers
*** Creating switches
*** Creating hosts
*** Creating links
*** Creating links
*** Starting network
*** Configuring hosts
h3 h4 h5 h6 h7 h8 h9 h10
*** Testing network
*** Ping: testing ping reachability
h3 -> X X h6 h7 h8 h9 h10
h4 -> h3 h5 h6 h7 h8 h9 h10
h5 -> h3 h4 h6 h7 h8 h9 h10
h6 -> h3 h4 h5 h7 h8 h9 h10
h7 -> h3 h4 h5 h6 h8 h9 h10
h8 -> h3 h4 h5 h6 h7 h9 h10
h9 -> h3 h4 h5 h6 h7 h8 h10
h10 -> h3 h4 h5 h6 h7 h8 h9
*** Results: 3% dropped (54/56 received)
*** Running CLI
*** Starting CLI:
mininet>
```

Figure 37 : Exécution de notre de notre topologie sous mininet

Et comme nous l'avions dit précédemment les différentes comtroleurs sont synchronisés. C'est pourquoi quand nous allons dans chaque interface graphique de chaque contrôleur, nous voyons le contrôleur ne s'affiche pas seulement le domaine qu'il contrôle mais il affiche toute la topologie avec tous les domaines car le cluster permet aux différents contrôleurs de partager la charge.

D'où la figure 38 qui est l'interface graphique du contrôleur 1 avec son adresse 192.168.62.5. Ce contrôleur est relié au commutateur de backbone mais maintenant c'est comme si ce contrôleur est relié avec l'ensemble des domaines du réseau.

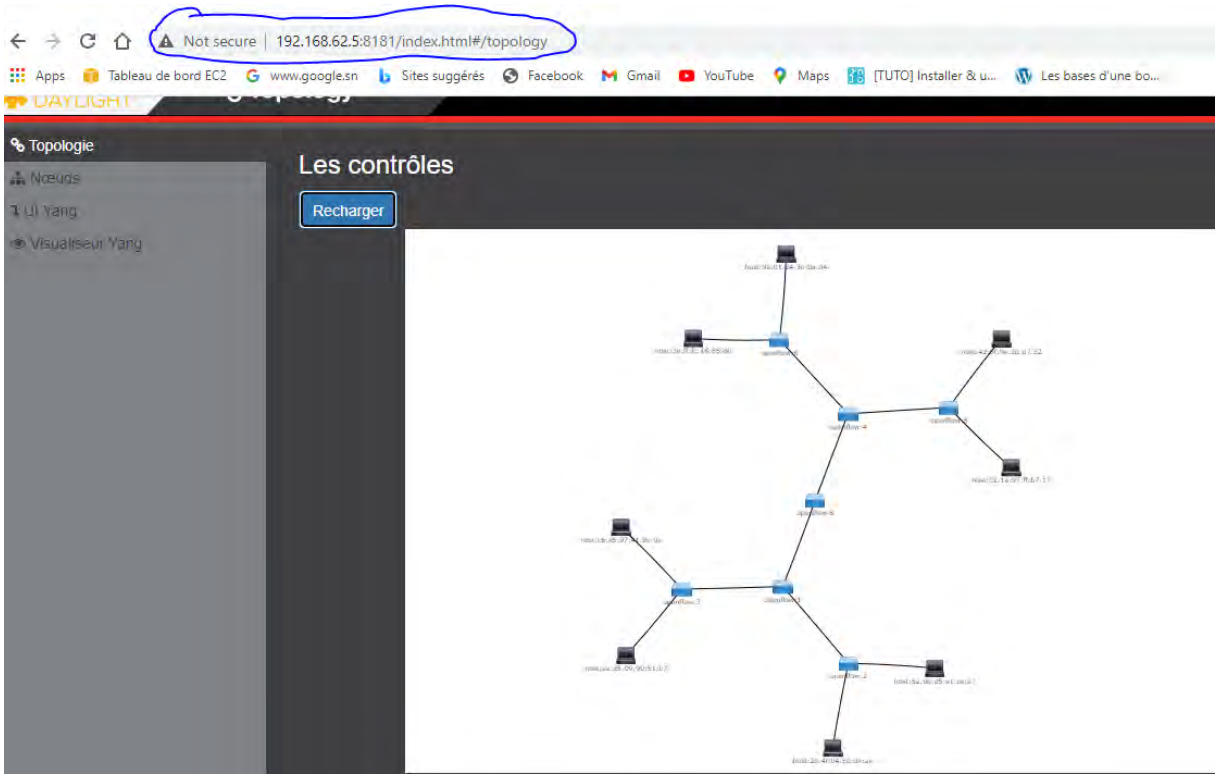


Figure 38 : Visualisation de la topologie dans le contrôleur OpenDayLight-1

De même, nous voyons que la figure 39 s'affiche la même chose car c'est le contrôleur 2 avec son adresse 192.168.62.6. Ce contrôleur est relié au domaine 2 avec les switchs S4, S5 et S6 mais comme les contrôleurs se sont synchronisés c'est pourquoi il affiche la topologie comme si c'est le seul qui contrôle le réseau.

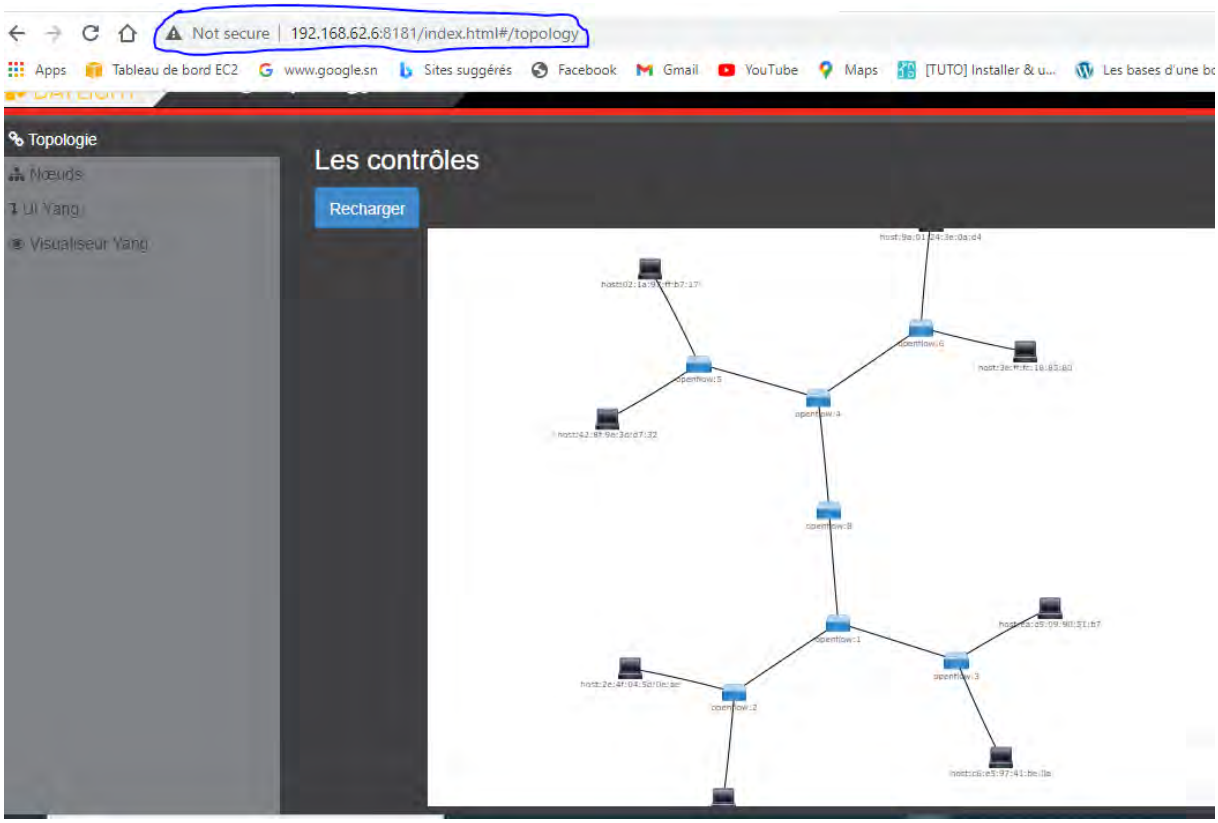


Figure 39 : Visualisation de la topologie dans le contrôleur OpenDayLight-2

4.8 Interprétation des résultats

Pour évaluer les performances du réseau nous avons utilisé l'outils cbench qui permet de tester la latence des contrôleurs.
Mais comme nous utilisons la méthode de clustering c'est pourquoi la latence entre les différents contrôleurs est presque nulle et c'est d'ailleurs ce qui permet aux contrôleurs de se synchroniser très rapidement.

En plus de la latence entre les contrôleurs, nous voyons aussi que la latence entre les contrôleurs et les commutateurs est faible (voir la figure 40). Et aussi le nombre d'emplacement possible pour ce cas est égal à trois emplacements.

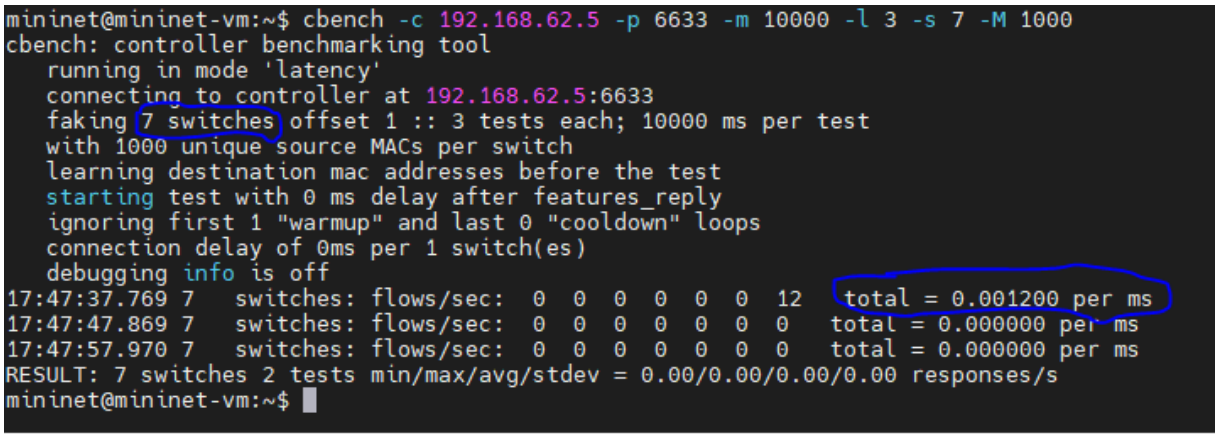


Figure 40 : Test sur la latence du réseau

En plus de la latence, nous avons que si un des contrôleurs tombe en panne, ça ne va pas impacter sur les performances du réseau car ce sont les deux autres contrôleurs qui vont toujours continuer à contrôler l'ensemble du réseau.
Donc nous pouvons dire que la fiabilité est bien assurée car le réseau est disponible et tolérant aux pannes.

4.9 Conclusion

Dans ce dernier chapitre nous avons présenté l'implémentation de la solution basée sur la méthode de cluster pour optimiser le nombre et l'emplacement des contrôleurs d'un réseau SDN.
L'analyse des résultats obtenus montre que cette solution permet de déterminer le nombre de contrôleur nécessaire et leur emplacement dans le réseau.
Et l'évaluation des performances montre que cette solution minimise la latence entre les contrôleurs et assure la fiabilité du réseau.

Conclusion générale et Perspectives

Les réseaux programmables sont des réseaux complexe dont tout l'intelligence du réseau est centralisée sur un seul élément appelé contrôleur. Des études sur leurs premières architectures utilisées montre la présence d'un point de défaillance unique causé par l'utilisation d'un seul contrôleur qui entraine une incapacité du réseau à gérer une grande quantité de demande de flux et une incapacité d'évoluer lorsque la taille du réseau augmente.
Pour résoudre ce problème causé par l'utilisation d'un seul contrôleur, il est nécessaire de penser à l'utilisation de plusieurs contrôleurs.
Mais l'utilisation de ces contrôleurs multiples a provoqué un problème appelé : « Le problème de placement des contrôleurs (CPP) » qui est l'un des principaux défis des réseaux programmables car il a un lourd impact sur les performances des réseaux SDN.

Dans ce mémoire, nous avons proposé une étude détaillée de quelques solutions optimisées de placement de contrôleurs dans les systèmes SDN. Nous avons aussi proposé un déploiement d'une solution multi-contrôleurs dans l'environnement de simulation Mininet.
Nous avons implémenté l'algorithme par le langage de programmation python pour exécuter la topologie. Les expériences faites sur certains nombres de réseaux de différentes topologies nous ont permis de penser à la méthode clustering.
Après avoir inséré les valeurs d'entrée, nous avons déployé le cluster, le nombre optimal des contrôleurs et leurs emplacements. Cette solution minimise la latence entre les commutateurs et les contrôleurs, mais aussi elle assure le partage de la charge et la synchronisation des données entre les contrôleurs qui sont des atouts majeurs de cette méthode de clustering.

Dans nos travaux futurs, l'optimisation du placement des contrôleurs est un point à considérer. Il serait intéressant d'étudier les problèmes liés à la relocalisation des contrôleurs, l'ajout, la suppression de contrôleurs, les défis liés à la migration des commutateurs et aussi étudier les problèmes liés à la mobilité et au CPP pour d'autres domaines de réseau tels que l'IoT et les réseaux de capteurs.

BIBLIOGRAPHIE

[1] D. Kreutz, F.M.V. Ramos, P.E. Verissimo, C.E. Rothenberg, S. Azodolmolky et S. Uhlig,"Software-defined networking: A comprehensive survey", *Proceedings of the IEEE*,IEEE
xplore, 103(1), pp. 14–76, 2015.

[2] F. Benamrane, M. Ben mamoun, et R. Benaini, « Performances of OpenFlow-Based Software-Defined Networks: An overview », *J. Netw.*, vol. 10, no 6, juin 2015

[3] J. Stribling, Y. Sovran, I. Zhang, X. Pretzer, J. Li, M. F. Kaashoek, and R. Morris, 'Flexible, Wide-area Storage for Distributed Systems with WheelFS', *Proceedings of the 6th Symposium on Networked Systems Design and Implementation*. NSDI'09, USENIX, pp. 43–58, 2009.

[4] A. Sallahi et M. St-Hilaire, "Optimal model for the controller placement problem in software defined networks", IEEE communications letters, vol. 19, no. 1, pp.30-33, 2015.

[5] Y. Hu, T.Luo, N.C.Beaulieu, et C.Deng, "The Energy-Aware Controller Placement Problem in Software Defined Networks", IEEE Communications Letters, pp.01-04, 2016.

[6] F.J. Pos, P.M. Ruiz, "On reliable controller placement in software defined networks", Computer Communications 77 (2016) 41–51, pp.43-50, 2016.

[7] K. S. Sahoo et al, "On the placement of controllers in software-Defined-WAN using meta-heuristic approach", The Journal of Systems & Software 145 (2018) 180–194, pp.182-192, 2018.

[8] J. Liao, H. Sun, J. Wang, Q. Qi, K. Li et T. Li, "Density cluster based approach for controller placement problem in large-scale software defined networkings", Computer Networks 112 (2017) 24–35, pp.26-33, 2017.

[9] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann, "Heuristic Approaches to the Controller Placement Problem in Large Scale SDN Networks", IEEE Transactions on Network and Service Management, Vol. 12, No. 1, pp. 4–17, March 2015.

[10] B. Heller, R. Sherwood, and N. McKeown, "The Controller Placement Problem", in Proc. of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networks (HotSDN'12), pp. 7–12, 2012.

WEBOGRAPHIE

[11] Gestion dynamique et évolutive de règles de sécurité pour l'Internet des Objets, [En ligne]. Disponible: https://www.theses.fr/2019REIMS011.pdf, (consulté le 10/12/2020).

[12] Software-Defined Networking (SDN) Definition, [En ligne]. Disponible: https://www.opennetworking.org/sdn-resources/sdn-definition, (consulté le 12/09/2020)

[13]. https://tools.ietf.org/html/rfc5810 (consulté le 15/12/2020)
[14]. http://tools.ietf.org/pdf/draft-smith-opflex-00.pdf (consulté le 17/12/2020)
[15]. https://www.theses.fr/2019REIMS011.pdf (consulté le 18/01/2021)
[16]. https://tools.ietf.org/html/rfc6690 (consulté le 20/01/2021)
[17]. http://python.proceranetworks.com/current/api_reference.html (17/09/2018 consulté le 24/01/2021)
[18]. https://github.com/frenetic-lang/pyretic/wiki/Dynamic-Policies (consulté le 27/01/2021)
[19]. https://www.researchgate.net/figure/OpenFlow-architecture_fig1_272579006 . (consulté 10/01/2021)
[20]. https://www.semanticscholar.org/paper/Efficient-load-balancing-for-multi-controller-in-H%E1%BA%A3i-Kim/9ac990613933bfdfe1d76b4d9b0c9a9bf9a2e3d3 (consulté le 14/01/2021)
[21] Akka : Build powerful concurrent & distributed applications more easily. [En ligne]. Disponible: http://akka.io/, (consulté le 07/01/2021)
[31] Inter-SDN Controller Communication-Using Border Gateway Protocol TCS. [En ligne]. Disponible : http://www.tcs.com/resources/white_papers/Pages/. (Consulté le 09/01/2021)