République Algérienne Démocratique Et Populaire

Ministère D'enseignement Supérieur Et De La Recherche Scientifique

Université Mohamed Boudiaf - M'sila

***Faculté** : mathématique et informatique*

***Département** : informatique*

***N°** :……..*

***Domaine** : mathématique et informatique*

***Filière** : informatique*

***Option** : RTIC*

Mémoire présenté pour l'obtention
Du diplôme de Master Académique

Par: Toumiat Soltana

Intitulé

Problème de placement du contrôleur dans les réseaux programmable

Soutenu devant le jury composé de :

.....	Université de M'sila	Président
Dr Lamri Sayad	Université de M'sila	Rapporteur
……	Université de M'sila	Examinateur

…

Année universitaire : 2019 /2020

Remerciement

Avant tout je dis AL HAMDOU LI ALLAH qui ma donne la volonté, la force, la patience et le courage pour réaliser ce modeste mémoire malgré les difficultés rencontrés.

Je voudrais dans un premier temps remercier, mon directeur de mémoire M.SAYAD LAMRI, professeur de l'informatique à l'université de M'sila, pour sa patience, sa disponibilité et surtout ses judicieux conseils, qui ont contribué à alimenter ma réflexion.

Je remercie tous les enseignants qui ont contribué au succès, et à tous ceux qui ont participé de près ou de loin à la réalisation de ce travail.

TABLE DES MATIERES

Chapitre 03: Implémentation et résultats

LISTE DES FIGURES

LISTE DES TABLEAUX

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) est 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. Bien que SDN soit encore à ses débuts, un certain nombre d'appareils et de logiciels réseau basés sur SDN sont déployé par les opérateurs des réseaux SDN, y compris les principaux acteurs de l'industrie tels que B4 de Google, SWAN de Microsoft et « Application Centric Infrastructure » de Cisco.

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.

L'interface de communication représentative du SDN est « OpenFlow », qui suppose initialement un seul contrôleur pour des raisons de simplicité. Mais avec l'expansion du réseau SDN, un seul contrôleur ne peut pas répondre à tous les besoins de gestion existants. Le fait d'avoir plusieurs contrôleurs améliore la fiabilité l'évolutivité du réseau.

Dans le présent travail, nous étudions le problème de placement de contrôleur (CPP) et cherchons à trouver l'emplacement et le nombre optimal des contrôleurs dans un réseau SDN.

Ce mémoire est constitué de trois chapitres structurés comme suit :

- **Le premier chapitre :** fournit une présentation générale des réseaux programmables (SDN), y compris les terminologies, l'architecture, le contrôleur SDN et relation entre le SDN et le NFV.

- **Le deuxième chapitre :** présente les travaux relatifs au placement de contrôleurs dans SDN, un résumé de l'état de l'art pour le CPP et une description détaillée d'un algorithme proposé.

- **Le troisième chapitre :** présente l'environnement du travail, l'implémentation d'algorithme proposés et discute les résultats obtenus.

1. Introduction

Avec l'accroissement constant des infrastructures réseaux actuelles, un nouveau modèle de fonctionnement et de déploiement devient nécessaire. SDN (Software Defined Networking) et la virtualisation des réseaux sont des technologies conçues pour répondre à ces nouvelles problématiques ; elles permettent de rationaliser l'utilisation des ressources disponibles, tout en offrant une flexibilité que des équipements physiques dédiés ne peuvent fournir.

Dans ce chapitre, nous présentons les technologies de virtualisation et le NFV (Network FunctionVirtualization), ensuite nous abordons la technologie du SDN en détail et la relation entre ce dernier et le NFV.

2. La virtualisation

La virtualisation n'est pas une technologie née récemment, elle est apparue aux années soixante à partir de recherches menées par IBM et MIT sur l'utilisation de ressources SI partagées. Cette technologie été abandonnée jusqu'aux années 2000, où elle est revenue en force avec l'explosion des systèmes d'information et le besoin pressant de les gérer plus efficacement.

2.1 Définition

C'est un processus qui va permettre de masquer les caractéristiques physiques d'une ressource informatique de manière à simplifier les interactions entre cette ressource et d'autres systèmes, d'autres applications et les utilisateurs. Elle va permettre de percevoir une ressource physique comme plusieurs ressources logiques et, inversement, de percevoir plusieurs ressources physiques comme une seule ressource logique [1].

L'encyclopédie francophone en ligne Wikipédia définit la virtualisation comme « la virtualisation est un ensemble des techniques matérielles et / ou logicielles qui permettent de faire fonctionner simultanément sur une seule machine plusieurs systèmes d'exploitation ».

Les intérêts de la virtualisation sont:

- économie du matériel par mutualisation.
- évolutivité rapide et flexible.
- utilisation optimale des ressources.
- allocation dynamique de la puissance de calcul.
- facilité d'installation, de déploiement et de migration des machines virtuelles.

2.2 Les domaines d'application

Il existe plusieurs domaines de virtualisation: la virtualisation de stockage, la virtualisation de réseaux, la virtualisation d'applications, et la virtualisation de serveurs.

❖ virtualisation de stockage

La virtualisation de stockage consiste à masquer la disparité physique des unités de stockage, et à les présenter comme un unique volume.

❖ la virtualisation de réseaux

Elle consiste à partager une même infrastructure physique (débit des liens, ressources CPU des routeurs,...) au profit de plusieurs réseaux virtuels isolés [1].

❖ la virtualisation d'applications

La virtualisation permet de séparer l'application du système d'exploitation hôte et des autres applications présentes afin d'éviter les conflits. Cette solution est particulièrement pratique pour simplifier l'administration des ressources informatiques notamment lors de l'installation des nouvelles versions.

❖ la virtualisation de serveurs

La virtualisation de serveur est un principe permettant de faire fonctionner simultanément, plusieurs serveurs virtuels sur un seul serveur physique.

3. Network Function Virtualization (NFV)

Le NFV a été standardisé par l'Industry Specification Group (ISG) de l'ETSI (European Telecommunications Standards Institute). Il est né de la volonté d'accélérer le déploiement de nouveaux services réseau.

3.1 Définition

Le NFV vise à transformer la manière dont les opérateurs de réseau conçoivent les réseaux en développant la technologie de virtualisation informatique standard pour consolider de nombreux types d'équipements de réseau sur des serveurs, commutateurs et stockage à grand volume standard de l'industrie, qui pourraient être situés dans les Datacentres, Nœuds de réseau et dans les locaux de l'utilisateur final. Il s'agit de la mise en œuvre de fonctions de réseau dans des logiciels qui peuvent fonctionner sur une gamme de matériel de serveur standard de l'industrie et qui peuvent être déplacés ou instanciés à divers endroits du réseau, au besoin, sans qu'il soit nécessaire d'installer de nouveaux équipements [2].

3.2 Architecture

L'architecture NFV définie par l'ETSI est représentée sur la figure 1.1:

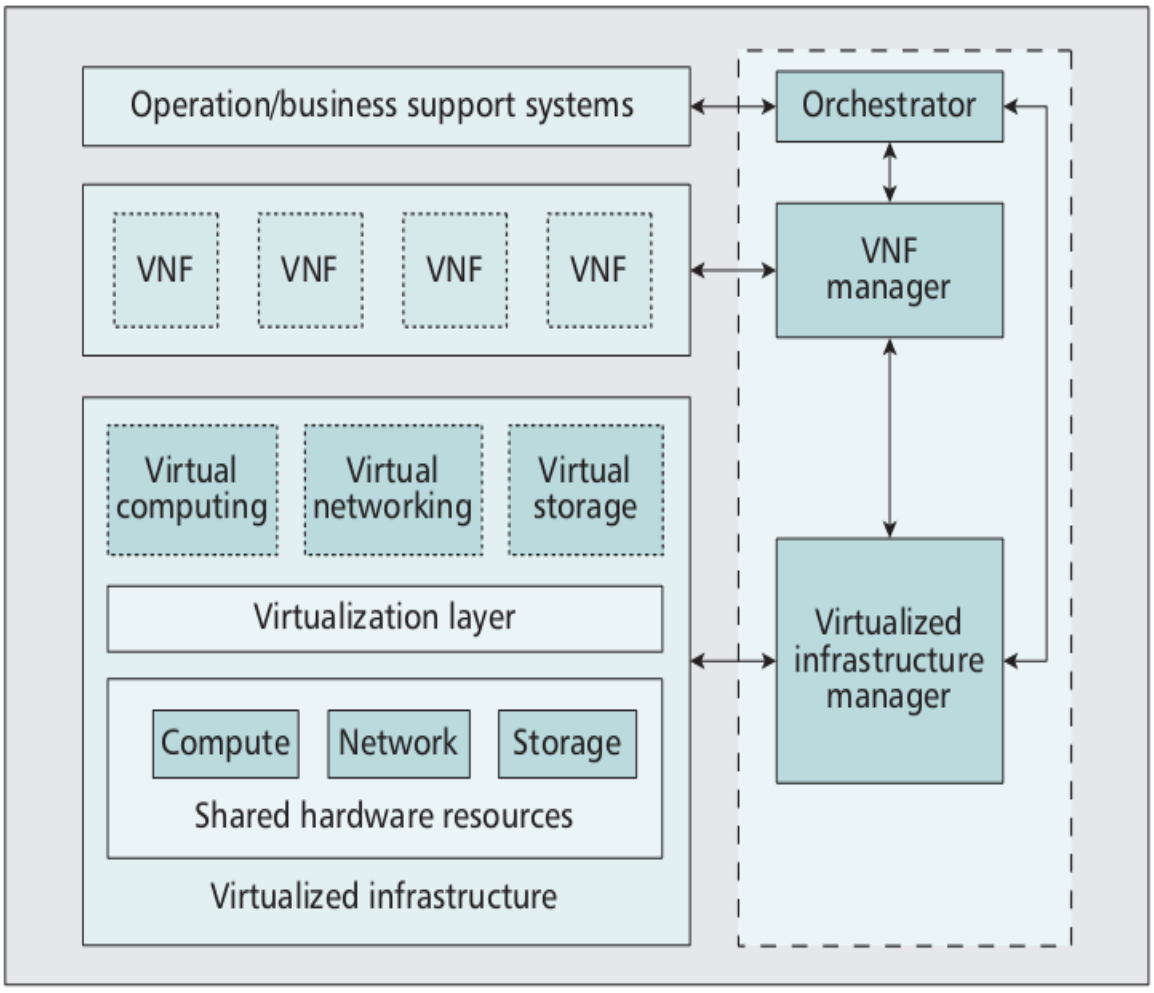

Figure 4.1 - Architecture NFV définit par ETSI [3].

L'architecture NFV est constituée de :

❖ Network Function Virtualisation Infrastructure (NFVI)

Fournit les ressources matérielles et le logiciel de virtualisation. Le NFV Infrastructure se compose de :

- une interface matérielle (stockage, réseau, calcul).
- une couche de virtualisation entre le matériel et le logiciel.
- une interface virtuelle (stockage, réseau, calcul).

❖ *Virtualised Network Function (*VNF)

Correspond aux fonctions réseaux virtualisées. Il s'agit de machines virtuelles fonctionnant sur l'infrastructure NFV (NFVI).

❖ NFV Management and Orchestration

Permettant de gérer les services réseaux de bout en bout est composé de trois blocs :

- L'orchestrateur (NFV Orchestrator) : assure l'orchestration des ressources de l'IVNF sur plusieurs VIM et la gestion du cycle de vie des services de réseau.
- un gestionnaire (VNFM) en charge du cycle de vie des VNFs (la création, l'exploitation et la terminaison).
- un gestionnaire (VIM) a une relation avec NFVI pour la virtualisation des ses ressources, la création et la suppression des ressources virtuelles et le suivi de leur bon fonctionnement.

❖ Les services OSS/BSS

Elles doivent pouvoir transmettre au bloc NFV management and orchestration des informations sur le profil des utilisateurs, la facturation, les politiques de qualités, …

3.3 Avantages et inconvénients du NFV

3.3.1 Avantages

✓ Réduction des coûts de l'équipement et réduction de la consommation d'énergie grâce à la consolidation de l'équipement.

✓ Allongement du cycle de vie des équipements matériels du réseau.

✓ Le NFV permet aux opérateurs d'ajouter, de supprimer, de configurer et de mettre à niveau des services et fonctionnalités réseau en temps réel et en fonction des besoins des clients, plutôt que d'exiger la commande d'Appliance réseau et leur livraison aux succursales et sites distants ou d'imposer des services identiques à tous les clients. Et la vitesse de service est améliorée.

✓ les ressources de réseau sont accessible A tout moment (Disponibilité).

3.3.2 Inconvénients

× Les environnements NFV sont plus dynamiques que les environnements traditionnels, ce qui peut nécessiter une mise à l'échelle avec des fonctionnalités supplémentaires pour faire face.

× NFV exige également un réalignement des processus afin que les infrastructures traditionnelles et virtuelles puissent être gérées simultanément [4].

× Le NFV s'avère complexe et difficile pour de nombreux opérateurs à déployer à grande échelle. L'étendue de l'architecture et le nombre de composants distincts rendent difficile la conception, la construction et le support.

4. Software Defined Network (SDN)

4.1 Définition

Le SDN est une architecture qui découple les fonctions de contrôle et de transfert des données du réseau afin d'avoir une infrastructure physique complètement exempte de tout service réseau [5].

4.2 Architecture

Le SDN présente une architecture réseau où le plan de contrôle est totalement séparé du plan de données, cela est illustré par la figure 1.2 :

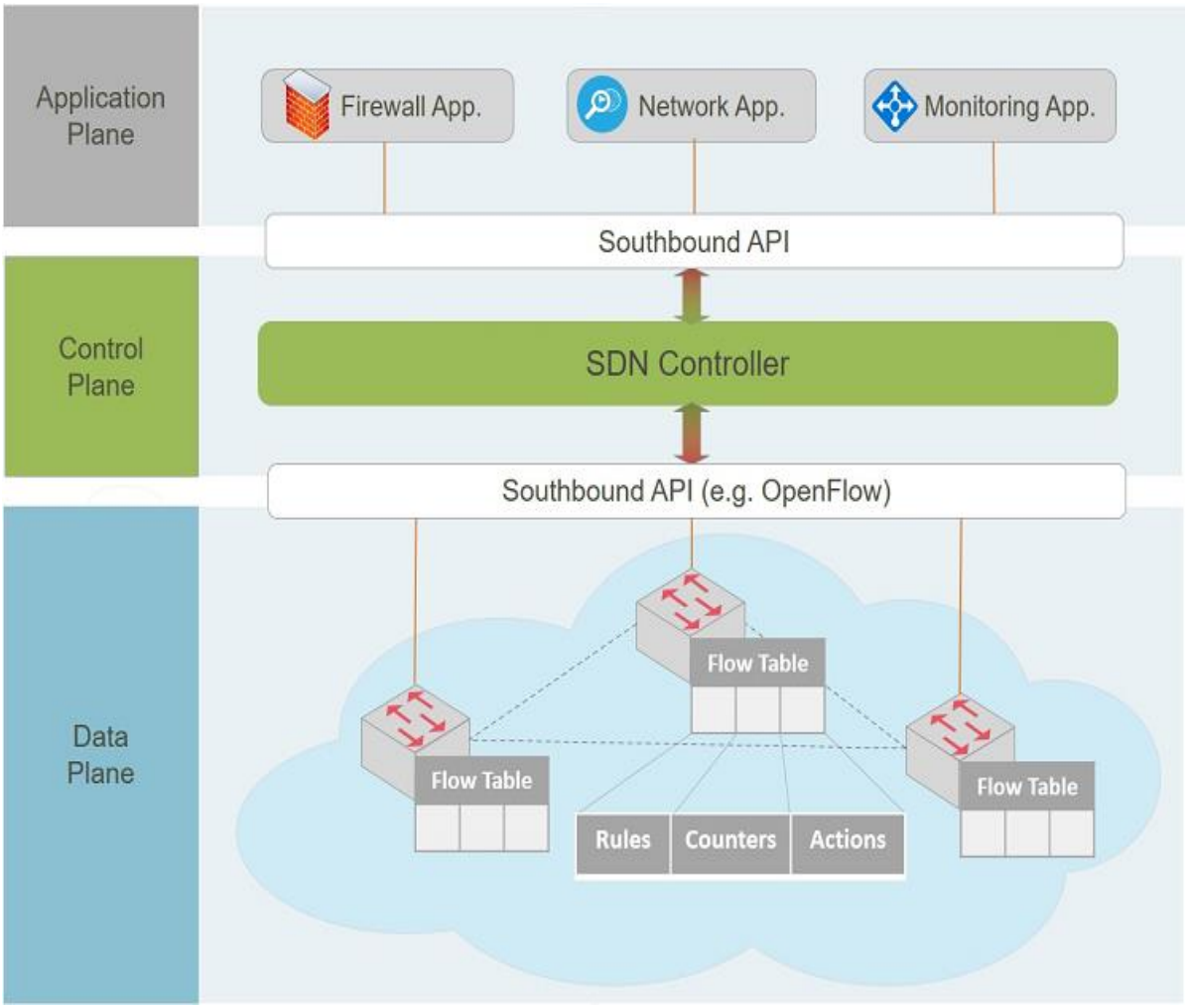

Figure 1.5 - Architecture du SDN [6].

➢ La couche application

Contient diverses applications SDN pour diverses fonctionnalités telles que le routage, les services de sécurité et l'équilibrage de charge. Elle communique avec le contrôleur SDN via l'interface Northbound.

➢ La couche contrôle

Prend des décisions sur la destination du trafic. Elle est responsable de la maintenance logique d'une vue globale du réseau, de la fourniture d'abstraction matérielle aux applications SDN et de l'exécution de tâches de contrôle pour gérer les périphériques réseau dans la couche infrastructure via des interfaces Southbound.

➢ La couche infrastructure

Elle est composée d'équipements de réseau tels que des commutateurs et routeurs qui transmettaient les flux de trafic en fonction des règles installées par les contrôleurs. Elle permet le transfert de données vers ou depuis les hôtes finaux.

4.3 Le contrôleur SDN

SDN est un nouveau paradigme qui réduit la gestion du réseau à définir des programmes de contrôle et les déployer sur une ou plusieurs entités connues sous le nom de contrôleurs.

4.3.1 Définition

Certaines solutions SDN reposent sur la mise en place de contrôleurs. Ces derniers ont pour mission de fournir une couche d'abstraction du réseau et de présenter ce dernier comme un système. Le contrôleur SDN permet d'implémenter rapidement un changement sur le réseau en traduisant une demande globale (par exemple: prioriser l'application X) en une suite d'opérations sur les équipements réseau (requêtes Netconf, ajouts d'états Openflow, configuration en CLI) [5].

Une application donne des ordres au contrôleur via une API « Northbound». Et le contrôleur communique avec les commutateurs via une ou plusieurs API « Southbound ».

4.3.2 Exemples de contrôleurs SDN

Il existe plusieurs contrôleurs SDN, tel que :

- **NOX**

Initialement développé par Nicira, NOX est le premier contrôleur OpenFlow. C'est Open-source et écrit en C ++ et a été utilisé par de nombreux chercheurs. Il est actuellement en baisse: il n'y a pas eu de changements majeurs depuis mi-2012 [7].

- **POX**

Il s'agit d'un contrôleur open source écrit en Python, et comme NOX, fournit un cadre pour développer et tester un contrôleur OpenFlow, mais les performances POX sont nettement inférieures à celles des autres contrôleurs et ne convient donc pas au déploiement d'entreprise [7].

Le contrôleur POX est un contrôleur Python pur, repensé pour améliorer les performances par rapport au NOX Python original.

- **Floodlight**

Le Floodlight est un contrôleur OpenFlow basé sur Java, dérivé du contrôleur Beacon développé à Stanford. Le contrôleur Floodlight est un logiciel open source sous licence Apache, pris en charge par une communauté de développeurs. Il propose une architecture modulaire, facile à étendre et à valoriser [8].

- **Beacon**

Beacon est un contrôleur Java connu par sa stabilité. Il a été créé en 2010 et est toujours maintenu, il a été utilisé dans plusieurs projets de recherche. En raison de ses performances, c'est une solution fiable pour l'utilisation dans des conditions réelles. Ce contrôleur a également été utilisé dans d'autres projets tels que Floodlight ou OpenDaylight [7].

- **OpenDaylight**

Opendaylight est un projet de la fondation Linux. Il s'agit d'un Frameworkopen source qui facilite l'accès aux réseaux programmables (SDN) et à la virtualisation des fonctions réseau (NFV). Il peut également être considéré comme une solution complète.

4.4 Openflow

OpenFlow est un protocole de communication entre un ou plusieurs contrôleurs SDN et les commutateurs qui supportent OpenFlow. C'est le plus couramment utilisé. Il permet d'exécuter des opérations du plan de contrôle dans un contrôleur OpenFlow.

Un switch OpenFlow est déterminé par une ou plusieurs tables de flux (flow table). Chaque table s'assimile à un ensemble de flux, qui consistent chacune en une liste de discriminants relatifs au contenu d'une trame, associée à des actions de traitement à appliquer aux trames correspondantes. Lorsqu'un paquet parvient au switch, les valeurs contenues dans ses en-têtes sont comparées aux différentes règles enregistrées dans la table de flux du switch [9].

4.5 Avantages et inconvénients du SDN

4.5.1 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 [7].

✓ 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.

4.5.2 inconvénients

× 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.

5. SDN et NFV

La virtualisation des fonctions réseau (NFV) est très complémentaire à la technologie des réseaux programmables (SDN), mais elles sont indépendantes. La virtualisation des fonctions réseau peut être implémentée sans qu'un SDN soit nécessaire, bien que les deux concepts et solutions puissent être combinés et potentiellement plus de valeur ajoutée.

NFV et SDN proviennent tous les deux de la demande de flexibilité et d'interopérabilité des clients, plutôt que de l'avènement d'une technologie particulière, ces technologies peuvent être utilisées ensemble et le sont et comme être utilisées conjointement avec d'autres nouvelles technologies. Cela dit, il existe des différences entre ces technologies : SDN sépare le plan de contrôle du plan de données en rendant le réseau entièrement programmable. Tandis que NFV transfère les fonctions réseaux en les virtualisant sur des serveurs standards. SDN permet d'orchestrer le trafic réseau et contribue à son automatisation tandis que NFV se concentre sur les services.

La figure 1.3 représente une architecture IoT basée sur la combinaison du SDN et du NFV :

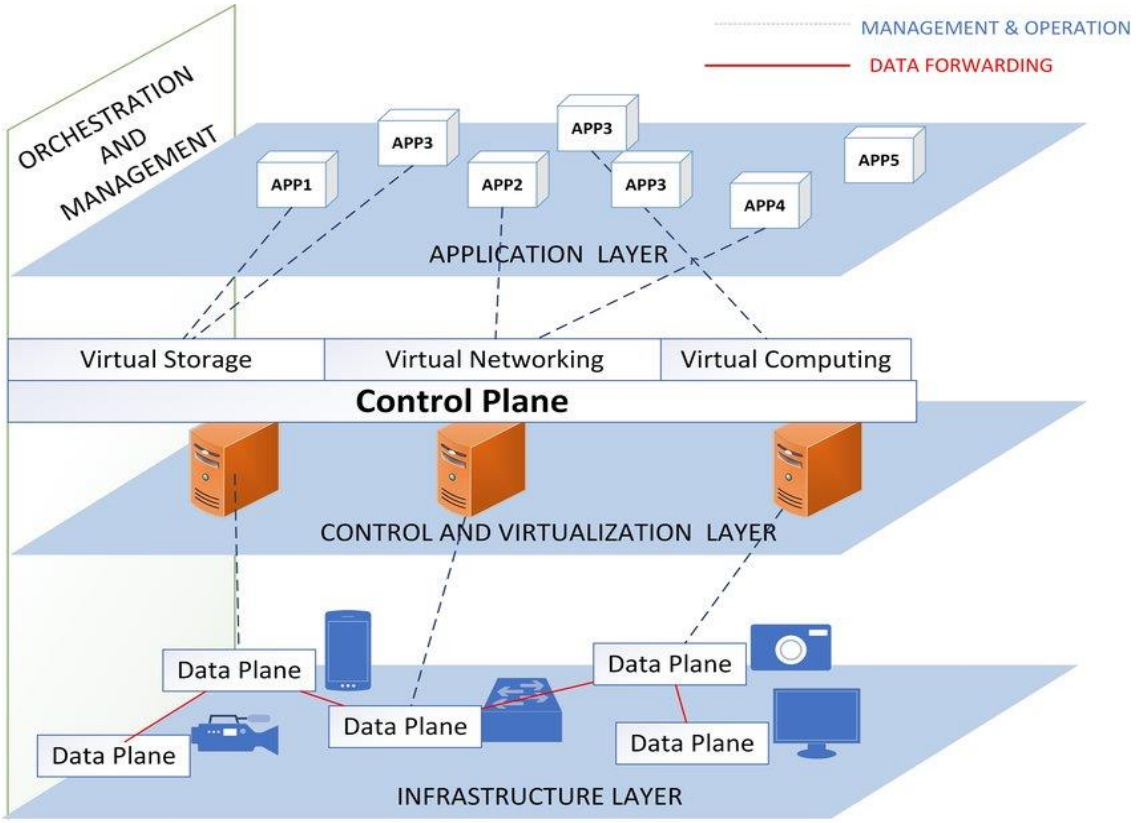

Figure 1.6 - architecture IoT basée SDN et NFV [10].

6. Conclusion

Au cours de ce premier chapitre, nous avons fourni une base théorique sur les technologies des réseaux : la virtualisation, NFV et les réseaux programmables (SDN). Nous avons commencé par introduire la technologie de virtualisation et ses domaines d'applications, puis le NFV, son architecture, ses avantages et ses inconvénients, ensuite nous avons présenté la technologie du SDN en détail et la relation entre ce dernier et le NFV.

Chapitre 02

Le problème de placement de contrôleur dans un réseau SDN

1. Introduction

Les réseaux programmables (SDN) est un nouveau paradigme qui découple le plan de contrôle et le plan de données, cette séparation simplifie la gestion du réseau et améliore son évolutivité. Le contrôleur dans le plan de contrôle gère les commutateurs en leur fournissant des règles qui dictent leur comportement de transfert des paquets. Dans un réseau à grande échelle, un bon placement utilise au mieux la connectivité réseau existante entre les commutateurs. Une réponse rapide et une connexion fiable entre le commutateur et le contrôleur associé est un point clé pour les réseaux SDN. Un seul contrôleur est difficile à contrôler tous les commutateurs dans un réseau SDN à grande échelle, car la capacité du contrôleur est limitée et la latence de propagation entre le contrôleur et les commutateurs est très grande. Actuellement, la plupart des recherches visent à déployer plusieurs contrôleurs à différents endroits pour contrôler l'ensemble du plan de données. Dans ce type d'architectures, le placement de plusieurs contrôleurs devient un problème critique.

Dans ce chapitre, nous présentons des travaux liés au problème de placement de contrôleur (CPP) dans les réseaux programmables, puis, un résumé de l'état de l'art sur CPP, ensuite nous abandonnons la formulation du problème, et enfin nous proposons un algorithme pour résoudre ce problème.

2. Travaux connexes

La première étude sur le sujet est due à Heller et al [11]. Ils ont analysé l'impact du déploiement du contrôleur sur la latence moyenne du réseau et la latence maximale.

Par la suite, d'autres objectifs de performance 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. La figure 2.1 montre la classification du CPP du point de vue de l'objectif optimisé.

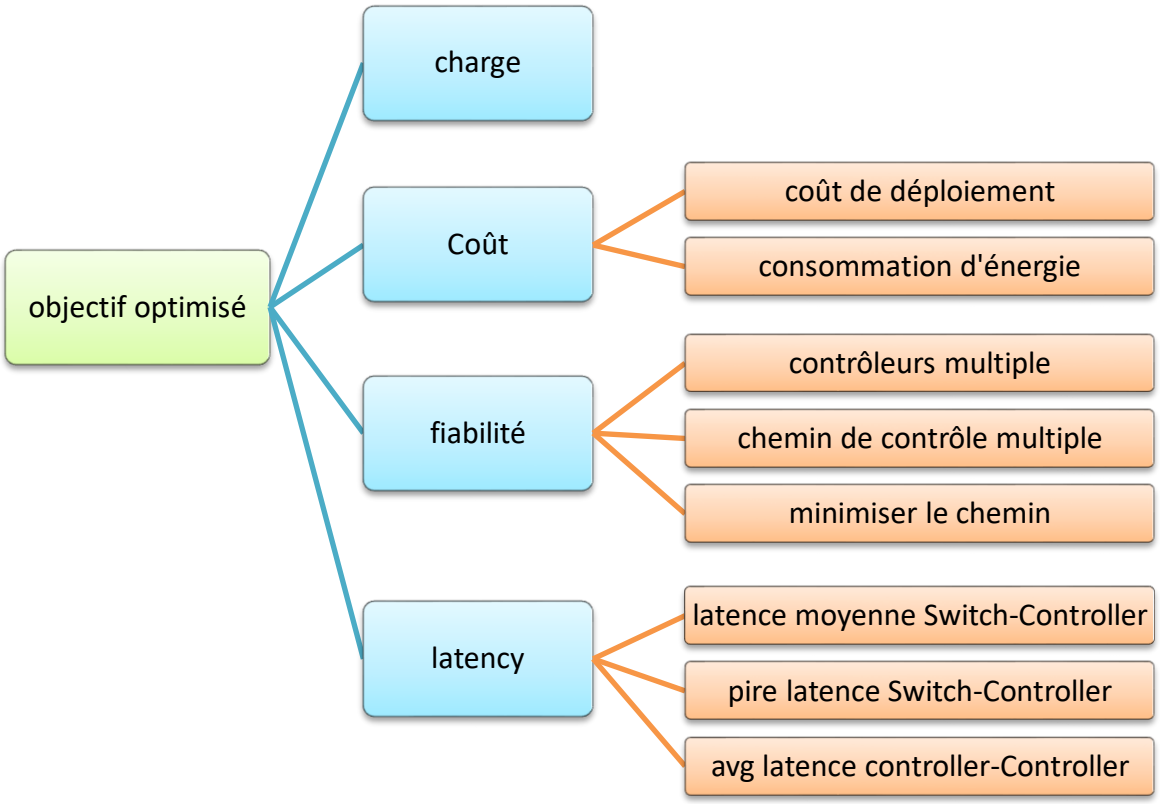

Figure 2.1 Classification des objectifs optimisés

Sallahi et M.St-Hilaire [12]

Sallahi et M.St-Hilaire [12] 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.

Le modèle proposée 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***)***.

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

Tout en tenant compte la latence de la configuration du chemin et la capacité du ou des contrôleurs. Chaque type de contrôleur a un nombre maximum de demandes qu'il peut gérer par seconde, un coût ($), un certain nombre de ports physiques disponibles. Ces informations sur les contrôleurs et d'autres sur les liens ont été utilisées comme des d'entrée pour la simulation, le modèle de programmation linéaire a d'abord été généré à

l'aide du langage de programmation Python, puis envoyé à l'optimiseur (CPLEX). Les résultats de simulation montrent que :

- Seuls les problèmes de petite taille peuvent être optimisés dans un délai raisonnable.
- Environ 10 % des problèmes ne peuvent être résolus dans le délai de 30 heures.

F.J. Ros et al [13]

F.J. Ros et al [13] définissent un problème de placement de contrôleur tolérant aux pannes (FTCP). Ils ont concentré sur la détermination du nombre de contrôleurs qui doivent être instanciés, de l'endroit où ils doivent être déployés et des nœuds de réseau qui sont contrôlés par chacun d'eux, afin d'obtenir une grande fiabilité dans l'interface sud entre les contrôleurs et les nœuds. Les auteures développent un heuristique algorithme permet de calculer les emplacements pour répondre à cette fiabilité et a donné une limite supérieure sur le nombre de contrôleurs.

La fiabilité est calculée comme suite :

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

Où $\prod j_0, t$ l'ensemble des chemins disjoints entre j_0 et t et $P_{u,v}$ 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 Virtual-Box comme hyperviseur. Ces machines virtuelles contrôlent les topologies de réseau créées avec mininet.

Comme prévu, 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 nœud est nécessaire pour se connecter à seulement 2 contrôleurs, qui fournissent généralement plus que le seuil de fiabilité requis.

Y. Hu et al. [14]

Les auteurs de l'article [14] 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) que la consommation d'énergie du réseau est minimisée sous les contraintes de le délai du chemin de contrôle et de charge du contrôleur. L'objectif est défini comme suite :

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

Où ρe 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 utiliser 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 possible en s'assurant que chaque commutateur a un chemin vers un contrôleur et que les charges de contrôleurs sont également équilibrées.

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.

Ishigaki et al. [15]

Basé sur la théorie de l'optimisation des graphes, Ishigaki [15] considère la surcharge de communication entre les contrôleurs et propose le problème de minimiser la distance entre les contrôleurs. L'article considère ce problème comme un problème de couverture d'ensemble minimum pondéré et prouve que le problème est NP-difficile. Après cela, l'auteur propose trois algorithmes gourmands avec des poids différents.

Les résultats montrent que l'algorithme utilisant des poids de distance moyens et des poids fixes a la plus grande efficacité.

K. S. Sahoo et al [16]

K. S. Sahoo et al [16] 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: (a) la latence du contrôleur pour basculer, (b) la latence inter-contrôleur et (c) la connectivité multi-chemins entre le commutateur et le contrôleur.

Le but est de trouver un placement de contrôleur tel qu'il minimise la latence moyenne entre le commutateur et le controleurπ^{avglat} et entre controleurs $\pi^{avgiclat}$ et en même temps maximise le nombre moyen de chemins disjoints $\pi^{avgdispath}$ entre le contrôleur et les commutateurs.

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

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 les algorithmes sont écrits dans la version MatLab R2014a et fonctionnent sur un système doté d'Intel Core i5 avec des processeurs 4 cœurs 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.

Le résultat de la simulation montre que :

- l'approche proposée minimise le délai moyen lors d'une panne de liaison unique par rapport aux travaux existants.

- bien que les deux algorithmes ont des résultats acceptables, CPP_FFA a produit le résultat optimal avec un temps de calcul moindre.

V. Ahmadi et M. Khorramizadeh [17]

L'objectif principal d'article [10] est d'étudier le problème de placement du contrôleur dans les SDN. Diverses mesures telles que la latence entre les nœuds et les contrôleurs, la latence entre les contrôleurs et l'équilibrage de charge sont considérées comme des fonctions objectives qui sont définies comme suite :

- la latence entre les nœuds (***V***) et les contrôleurs (***P***)

$$\pi^{L\_avg\_N2C}(P) = \frac{1}{|V|}\sum_{v\in V} \min_{p\in P} d_{v,p}$$

- la latence entre les contrôleurs

$$\pi^{L\_\text{avg}\_C2C}(P) = \frac{1}{\binom{|P|}{2}}\sum_{p1,pe\in P} d_{p1,p2}$$

- la charge des contrôleurs.

  Afin d'équilibrer la répartition de la charge entre les contrôleurs, une métrique de déséquilibre $\pi^{imbalance}$ est introduite pour être minimisée. Chaque nœud doit être connecté à son contrôleur le plus proche.

$$\pi^{imbalance}(P) = \max_{p\in P} n_p - \min_{p\in P} n_p$$

Pour établir un compromis entre ces objectives contradictoires, V. Ahmadi et M. Khorramizadeh [17] présentent un algorithme heuristique appelé Multi-Start Hybrid Non-dominated Sorting Genetic Algorithm (MHNSGA) pour résoudre efficacement le problème de positionnement du contrôleur multi-objectifs. Ils introduisent une nouvelle variante de ce problème appelé MOCCPP dans laquelle la capacité du contrôleur et la charge des commutateurs sont ajoutées comme des contraintes. MHNSGA est adapté pour résoudre le MOCCPP en introduisant une nouvelle technique de gestion des contraintes.

MHNSGA a été appliqué sur plusieurs topologies d'Internet Topology Zoo et comparé à certains d'autres algorithmes, notamment POCO, PSA et PSO – CGLCPP. Les résultats numériques montrent que MHNSGA a besoin de moins de mémoire et de temps de calcul que les trois autres algorithmes, il est 20 fois plus rapide que l'évaluation exhaustive du cadre POCO pour des scénarios avec plus de 14 millions d'espaces de recherche.

Zhang et al [18]

Zhang et al [18] utilisent l'algorithme Adaptive Bacterial Foraging Optimization (ABFO) pour résoudre le problème de placement du contrôleur multi-objectif (MOCP). La solution proposée permet d'améliorer la fiabilité du réseau (un commutateur peut être contrôlé par plusieurs contrôleurs pour fournir une tolérance aux pannes), d'équilibrer la charge entre les contrôleurs et de réduire la latence entre le contrôleur et le commutateur.

ABFO utilise des stratégies adaptatives pour améliorer le Bacterial Foraging Optimization (BFO) original. Le système BFO classique se compose de trois mécanismes principaux la chimiothérapie, la reproduction ou la dispersion et l'élimination. Les auteurs personnalisent les étapes chimiothérapie de l'ABFO pour effectuer trois variations dans chaque temps d'itération : les changements de position du contrôleur, les changements dans la relation de cartographie entre le contrôleur et le commutateur, et changement de la proportion des demandes de routage à partir des commutateurs traités par les contrôleurs.

Le schéma proposé de MOCP basé sur ABFO a est simulé par MATLAB et ses performances sont évaluées sur trois topologies qui sont Internet2 OS3E, Tinet de Internet Topology Zoo et RF-Ii qui est une topologie ISP du référentiel Rocketfuel, l'evaluation s'est effectué avec algorithme heuristique en [10] et Pareto Simulated Annealing (PSA) pour connaître l'influence du nombre de contrôleurs sur la performance du réseau, la fiabilité du réseau de contrôle, l'équilibre de charge du contrôleur et latence de le chemin de contrôle.

Les résultats de la simulation montrent que :

- La probabilité de fiabilité de l'heuristique est d'environ 0,199% supérieure à PSA et de 0,015% supérieure à ABFO, et la probabilité de fiabilité de l'ABFO est d'environ 0,184 % supérieure à celle de PSA en moyenne.
- le facteur d'équilibrage de la charge d'ABFO est respectivement environ 96,908% inférieur à l'heuristique et 85,931% à PSA en moyenne.
- la latence de l'ABFO est respectivement environ 36,114% inférieure à l'heuristique et 15,274% inférieure à la PSA en moyenne.

3. Synthèse

Le tableau 2.1 résume les travaux pertinents qui sont étroitement liés au problème de placement des contrôleurs.

référence	**année**	**Objectifs**				**contraintes**				**algorithme**	**Simulation et resultats**
		latence	Fiabilité	coût	charge	Capacité de contrôleur	Nombre de contrôleurs	latence	Tolérant aux pannes		
Sallahi et M.St-Hilaire [12]	2015			√		√	√	√		Programmation inter linéaire	-Seuls les problèmes de petite taille peuvent être optimisés dans un délai raisonnable. -Environ 10 % des problèmes ne peuvent être résolus dans le délai de 30 heures.
F.J. Ros et al [13]	2016		√				√		√	heuristique algorithme	- pour 75% de la topologie, 8 contrôleurs ou moins suffisent. -Pour obtenir une meilleure fiabilité, chaque nœud doit se connecter à seulement 2 contrôleurs.
Y. Hu et al [14]	2016			√		√	√	√		programme entier binaire (BIP), IGCPA	-Économies d'énergie allant jusqu'à 55 % pendant les heures creuses
Ishigaki	2017	√				√				Greedy	Les algorithmes gourmands utilisant

[15]										algorithm	le poids de distance moyen et le poids constant sont toujours efficaces.
K. S. Sahoo et al [16]	2018	√	√			√	√			l'algorithme de FireFly	Pour la latence de communication, le FFA a amélioré les performances que PSO.
V. Ahmadi et M. Khorramizadeh [17]	2018	√			√	√	√			Hybrid Non-dominated Sorting Genetic Algorithm (MHNSGA)	vec plus de 14 millions d'espaces de recherche, MHNSGA est 20 fois plus rapide que le framework POCO. iliser moins de resources.
Zhang et al [18]	2018	√	√		√	√	√	√	√	Adaptive Bacterial Foraging Optimization (ABFO)	-La probabilité de fiabilité de l'heuristique est d'environ 0,199% supérieure à PSA et de 0,015% supérieure à ABFO. -le facteur d'équilibrage de la charge d'ABFO est respectivement environ 96,908% inférieur à l'heuristique et 85,931% à PSA en moyenne. -la latence de l'ABFO est

											respectivement environ 36,114% inférieure à l'heuristique et 15,274% inférieure à la PSA en moyenne.
M. Tanha et al [19]	2018	√			√	√		√		2 algorithmes: RCCPP-AMC RCCPP-SMC	fournir aux fournisseurs de services des informations utiles sur la conception d'un WAN défini par logiciel résilient.

Table 2.1 – Résumé de l'état de l'art pour le problème de placement du contrôleur.

4. formulation du problème

La topologie de réseau de plan de données G (S, L) contient un ensemble de commutateurs S et un ensemble de liaisons bidirectionnelles L. un lien l_{ij} entre le commutateur s_i et s_j comme:

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

5. DBCP : Approche basée sur le cluster de densité

Liao et al [20] proposent une approche appelée Density Based Controller Placement (DBCP), qui utilise un algorithme de regroupement de commutateurs basé sur la densité pour diviser le réseau en plusieurs sous-réseaux,

Cette approche analyse la structure topologique et divise les réseaux en plusieurs sous-réseaux. Évidemment, les commutateurs devraient avoir des connexions étroites au sein du même sous-réseau, alors qu'ils devraient avoir relativement peu de connexions aux commutateurs dans d'autres sous-réseaux. Le sous-réseau doit être le réseau le moins séparable. Après la division, le sous-réseau présente des performances de tolérance aux pannes élevées. Dans le même temps, les performances de latence du sous-réseau ne peuvent pas être considérablement augmentées par une séparation plus fine [20].

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 pouvoir 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.

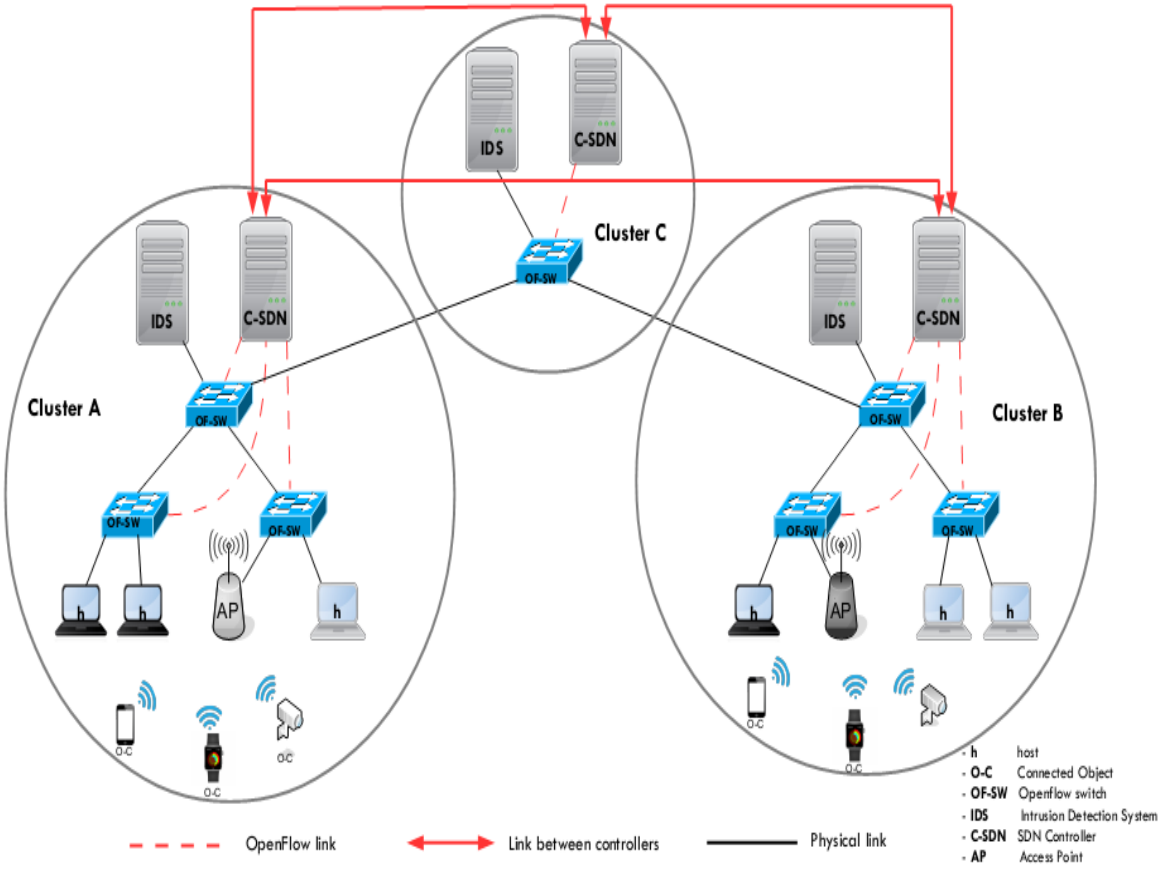

Figure 2.2 architecture basée sur un cluster de réseau programmable [21]

5.1.calculer la densité

Dans DBCP, nous avons divisé le réseau en regroupant les commutateurs. Pour chaque commutateur s_i, nous calculons une quantité: sa densité locale ρ_i. Cette quantité dépend uniquement des distances entre les commutateurs.

La densité locale ρ_i du commutateur s_i est définie comme:

$$\rho_i = \sum_j X (d_{ij} - d_c) \qquad (2)$$

Où d_{ij} représente la distance entre les commutateurs d_i et d_j. Le d_c est une distance seuil. Seule la distance entre les commutateurs inférieure à d_c sera considérée comme un commutateur fermé. Et la fonction χ est définie comme:

$$X (x) = \begin{cases} 1, & \text{si } x < 0 ; \\ 0, & \text{sinon.} \end{cases} \qquad (3)$$

Fondamentalement, ρ_i est égal au nombre de commutateurs dont les distances au commutateur s_i sont inférieures à d_c. Pour les réseaux à grande échelle, d_c est défini comme 0,3 fois le diamètre du graphe [20].

Algorithme 1 : le processus d'analyse.

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

Table 2.2 le processus d'analyse [20]

5.2. 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 :

$$L(\theta) \geq \sum_{s \epsilon S(\theta)} l(s), \ \forall \theta \epsilon \Theta \quad (4)$$

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. σ_i pour le commutateur s_i est calculé comme suit:

$$td_i = \sum_j^{s_j \in UL(s_i)} \rho_j \qquad (5)$$

$$\sigma_i = \sum_j^{s_j \in UL(s_i)} \frac{\rho_j}{td_i} \log_{|UL(s_i)|} \frac{\rho_j}{td_i} \qquad (6)$$

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

5.3.regroupement (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 3 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. σ_s = 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) ≥ $\sum_{s_i \in (N(s)+N(s'))} l(Si)$:
14. s belongs to the cluster N (s')
15. Break
16. end if

17. end for
18. end for

Table 2.3 le processus de regroupement (clustering) [20].

5.4. l'emplacement du contrôleur

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.

5.4.1. Latence contrôleur-à-commutateur

La fonction objective de la latence la plus défavorable par rapport au contrôleur θ est calculée comme suit :

$$\pi^{\text{maxlatency}}\left(\square(\theta)\right) = \min_{v\in\square(\theta)} \max_{s\in\square(\theta)} d(v,s) \qquad (7)$$

Où d (v, s) est la latence entre l'emplacement du contrôleur v et le commutateur s, qui est défini comme le nombre de sauts entre eux dans ce document. S (θ) représente la collection de commutateurs qui doit être contrôlée par le contrôleur θ.

5.4.2. Latence entre contrôleurs

Pour réduire les coûts de communication entre les contrôleurs, les contrôleurs doivent être déployés aussi étroitement que possible. Intuitivement, nous devons minimiser la latence entre eux. Cependant, ils sont déployés indépendamment. La latence de la communication entre contrôleurs ne peut pas être mesurée directement en calculant la latence entre eux.

Pour minimiser la latence d'un contrôleur vers d'autres contrôleurs, nous devons déployer ce contrôleur aussi étroitement que vers d'autres sous-réseaux. La latence d'un contrôleur au commutateur d'autres sous-réseaux peut être mesurée. Ainsi, la latence minimale d'un contrôleur à d'autres contrôleurs pourrait être mesurée en calculant la latence de son placement à tous les autres commutateurs du sous-réseau. La fonction objective de la latence inter-contrôleur par rapport au contrôleur θ est calculée comme suit:

$$\pi^{\text{inter}-\text{controller}}\left(\square(\theta)\right) = \min_{v\epsilon\square(\theta)} \frac{1}{|S|}\sum_{s\epsilon(S-\square(\theta))} d(v,s) \qquad (8)$$

Où S représente tous les commutateurs du réseau.

6. Conclusion

Dans ce chapitre, nous avons présenté un aperçu des travaux connexes et de la littérature sur le problème de placement des contrôleurs 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 03

Implémentation et résultats

1. Introduction

Dans le chapitre précédent, nous avons fait une revue des travaux existants qui traitent le problème de placement de contrôleurs dans SDN et nous avons présenté l'approche DBCP en détail. Dans ce chapitre, nous expliquerons l'implémentation de l'algorithme proposé, nous avons opté pour un langage de programmation orienté objet, rapide et offrant la possibilité de manipuler des interfaces graphiques de haut niveau. Pour cela, nous avons choisi le langage C++ avec l'environnement de développement Embarcadero C ++ Builder. Enfin, nous exposerons les résultats des tests effectués pour évaluer les performances de l'algorithme.

2. L'environnement de Développement :

Embarcadero C++ Builder

C ++ Builder est un environnement de développement rapide d'applications (RAD) (Figure 3.1), initialement développé par Borland et à partir de 2009 détenu par Embarcadero Technologies, pour l'écriture de programmes dans le langage de programmation C ++ ciblant Windows NT (IA-32 et x64), OS X, IOS et Android. C ++ Builder combine la bibliothèque de composants visuels et l'EDI écrits en Object Pascal avec plusieurs compilateurs C ++. La nouvelle version C ++ de RAD Studio est fournie avec le Framework VCL primé pour les applications Windows natives hautes performances et le puissant Framework FireMonkey (FMX) pour les interfaces utilisateur multiplateformes, elle permet de traiter plus rapidement les bogues grâce au débogage natif multiplateforme intégré « IDE » [22].

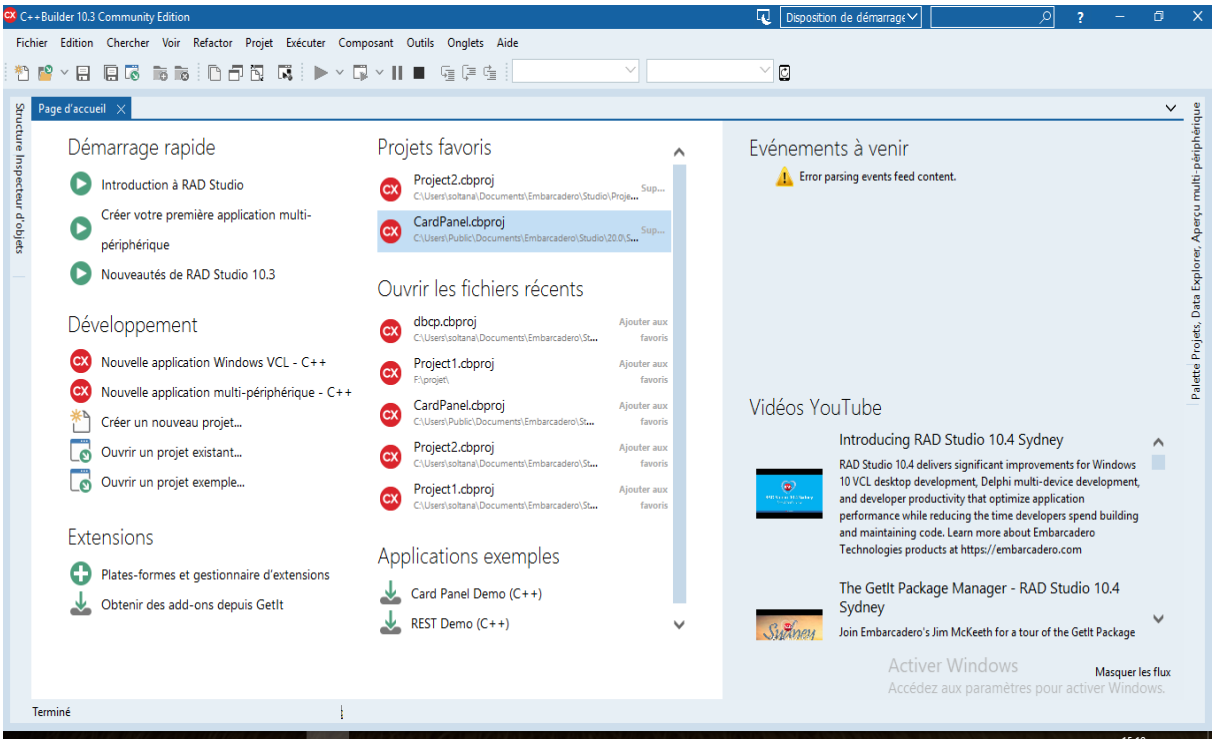

Figure 3.1- Embarcadero C++ Builder.

3. Implémentation

Dans ce travail, nous voulons résoudre le problème de placement des contrôleurs SDN et nous nous appuierons sur l'approche DBCP. Nous avons conçu notre programme en C++. Afin que nous puissions représenter tous les éléments et appliquer les algorithmes de manière appropriée, nous avions tellement d'éléments, de variables et de fonctions, donc nous parlerons juste des éléments essentiels:

- Le nombre de commutateurs n.
- Le nombre de contrôleurs (le nombre de clusters).
- La distance seuil d_c.
- La densité ρ.
- Borderline index σ.
- Latence.

Calcul_density ()

La fonction Calcul_density calcule le nombre de commutateurs dont les distances aux autres commutateurs sont inférieures à d_c. . La figure 3.2 montre le code de la fonction Calcul_density().

```
int calcul_density ( vector <Switch> v2, int dc,int s)
{
int densitymax=0;
  for (int j=0; j<v2.size(); j++){
     if (((v2[s].tab_distance[j] - dc) < 0) && (v2[s].tab_distance[j] != 0) )
        v2[s].density = v2[s].density + 1;
     if( densitymax < v2[s].density)
        densitymax=v2[s].density;
  }
  return v2[s].density;
}
```

Figure 3.2 – la fonction calcul_density

Calcul_borderIndex ()

Cette fonction permet de calculer le borderline index pour chaque commutateur s. La figure 3.3 montre le code de la fonction Calcul_borderIndex().

```
double calcul_borderIndex (vector <Switch> v3, int s)
{
 for (int j=0; j<v3.size() ;j++)
   if(v3[s].tab_distance[j] == 1 )
     if(v3[j].density >= v3[s].density){
       v3[s].td = v3[s].td + v3[j].density;
        v3[s].bordIndex = v3[s].bordIndex + (v3[j].density / v3[s].td) * log( v3[j].density / v3[s].td);
     }
 return v3[s].bordIndex ;
}
```

Figure 3.3 - la fonction calcul_borderIndex.

Sort_switchs ()

Dans le processus de clustering, les commutateurs avec une valeur inférieure de borderline index ont la priorité. La fonction sort_switchs trier tous les commutateurs selon le borderline index par ordre croissant. La figure 3.4illustre le code de cette fonction.

```
bool sort_switchs(const Switch &lhs, const Switch &rhs)
 { return lhs.bordIndex < rhs.bordIndex; }
```

Figure 3.4 - la fonction sort_switchs ()

DBCP ()

Il permet de trouver le nombre optimal de contrôleurs en fonction du clustering de densité, puis place les contrôleurs. Le placement est déterminé en fonction de la latence (distance entre les commutateurs). La figure 3.5 montre le code de cette fonction.

```
    for(int k=0; k<n; k++) {
    for(int i=0; i<n; i++)
       for(int j=0; j<n; j++)
          if(switchs[i].tab_distance[k] + switchs[k].tab_distance[j] < switchs[i].tab_distance[j])
            switchs[i].tab_distance[j] = switchs[i].tab_distance[k] + switchs[k].tab_distance[j];
}

dc=4;

for (int i=0; i < switchs.size(); i++)
   switchs[i].density = calcul_density (switchs, dc,i);

for (int i=0; i<n; i++)
    switchs[i].bordIndex = calcul_borderIndex(switchs, i);

for (int i=0; i < n; i++)
 for (int k=0; k<n; k++)
    if ((switchs[i].tab_distance[k] == 1)&&(switchs[i].density <= switchs[k].density))
      switchs[i].UL.push_back(k);

int temp;
 for (int k=0; k<n; k++)
 for(int i=0; i<switchs[k].UL.size(); i++)
      for(int j=i+1 ; j<switchs[k].UL.size(); j++)
          if(switchs[switchs[k].UL[j]].density < switchs[switchs[k].UL[i]].density)
          {
              temp = switchs[k].UL[i];
              switchs[k].UL[i] = switchs[k].UL[j];
              switchs[k].UL[j] = temp;
          }
```

```
sort(switchs.begin(), switchs.end(), sort_switchs);

for (int i=0; i<n; i++)
{  Cluster cluster;
   clusters.push_back(cluster);
}

 for (int i=0; i<n; i++)
    clusters[i].list_switchs.push_back(switchs[i].index);

for (int i=0; i < n; i++){           //s
for (int j=0; j<switchs[i].UL.size(); j++) //s'
{    int n1, n2,p;
  for (int d=0; d < clusters.size(); d++)
   for (int f=0; f < clusters[d].list_switchs.size(); f++)
     if( switchs[i].index == clusters[d].list_switchs[f])
      { n1=d;  p=f;  }    //s
       //
  for (int d=0; d < clusters.size(); d++)
    for (int f=0; f < clusters[d].list_switchs.size(); f++)
      if( switchs[i].UL[j] == clusters[d].list_switchs[f])
        n2=d;
        //
 if (capacité_ctrl >= clusters[n2].list_switchs.size()+1){
   clusters[n2].list_switchs.push_back(switchs[i].index);
   clusters[n1].list_switchs.erase(clusters[n1].list_switchs.begin()+p);
    break;}
 }
}
}
```

```
for (int i=0; i<clusters.size() ; i++){
    if (clusters[i].list_switchs.size() == 0)
      { clusters.erase(clusters.begin()+i); i=i-1;}
}

nbr_ctrl= clusters.size();

int list_controllers[nbr_ctrl];  int ctrl;
int min_latences [nbr_ctrl];
for (int i=0; i<nbr_ctrl; i++) min_latences[i]=999;
            int x;
 for(int k=0; k<nbr_ctrl; k++) {

    for(int i=0; i<clusters[k].list_switchs.size(); i++){
            int maxd=0;
            for (int f=0; f<n; f++)
            if( clusters[k].list_switchs[i]==switchs[f].index)
            x= f;
       for(int j=0; j<clusters[k].list_switchs.size(); j++){
          if(maxd <= switchs[x].tab_distance[clusters[k].list_switchs[j]])
            {maxd = switchs[x].tab_distance[clusters[k].list_switchs[j]];
            ctrl= switchs[x].index;}
       }
       if(maxd < min_latences[k])
       {min_latences[k] = maxd;
        list_controllers [k]= ctrl;
       }
    }
}
```

Figure 3.5- la fonction main

4. Résultats

La figure 3.6 nous montre de résultat obtenu d'appliquer le DBCP sur un réseau contient 23 commutateurs, avec une valeur de d_c égale à 4 et de capacité égale à 5.

```
-------------------------------------------------
The clusters:

Cluster 0:  - 13 - 19 - 16 - 12
Cluster 1:  - 14 - 15
Cluster 2:  - 21 - 22
Cluster 3:  - 5 - 2 - 0 - 1 - 6
Cluster 4:  - 4 - 3 - 8 - 9 - 20
Cluster 5:  - 10 - 7 - 11
Cluster 6:  - 18 - 17

-------------------------------------------------

The number of controllers: 7

-------------------------------------------------

Controller placement:

 13  14  21  5  4  11  18
```

Figure 3.6 – résultat obtenu d'appliquer le DBCP.

La figure 3.7 montre les emplacements des contrôleurs.

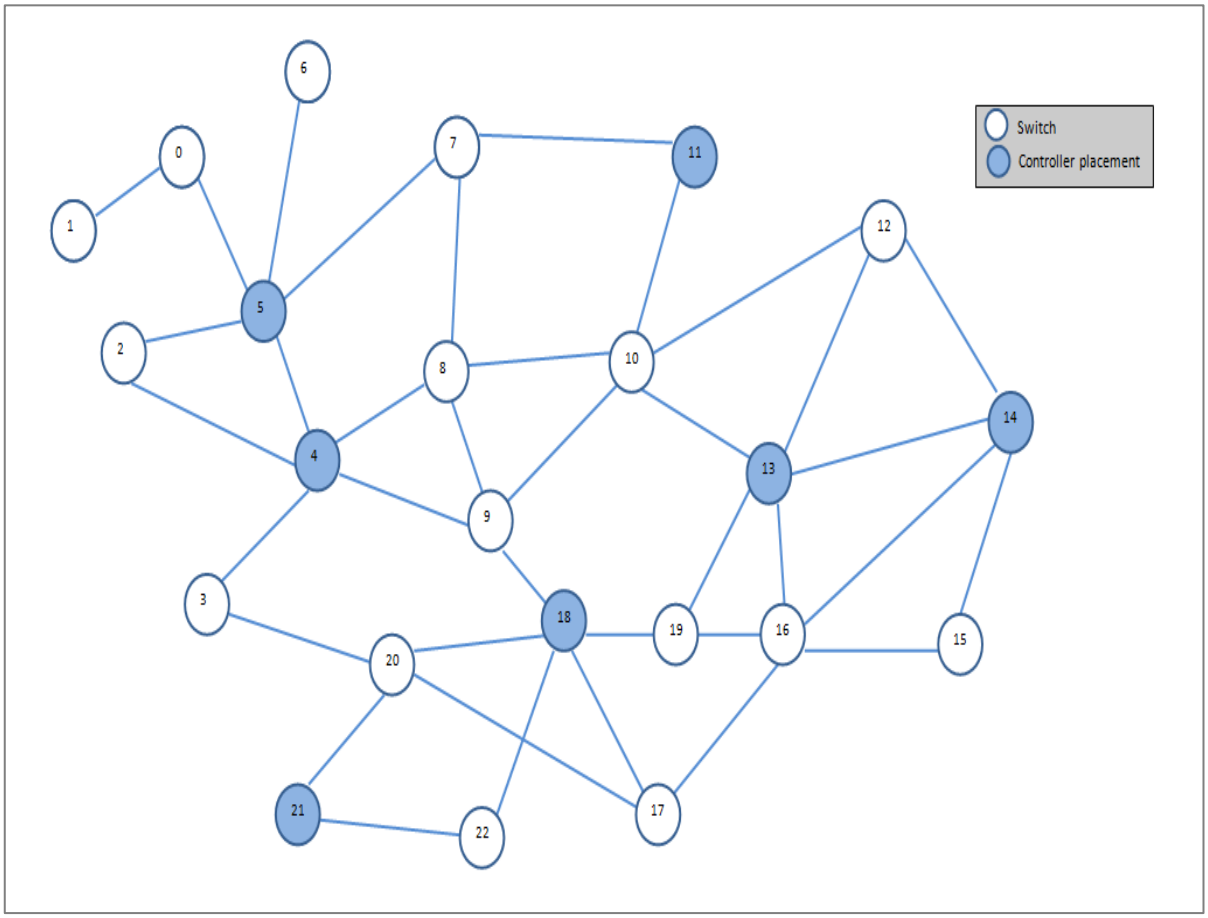

Figure 3.7 – l'emplacement des contrôleurs.

5. Évaluation de d_c

La valeur de d_c est le seul paramètre de notre algorithme. Pour estimer la relation entre d_c et les performances de clustering, nous définissons d_c comme étant les valeurs comprises entre 1 et le diamètre du réseau (la plus grande distance possible) afin de comparer les performances de DBCP. La figure 3.8 montre comment le nombre de clusters varie sous différents d_c.

Le nombre des commutateurs dans les 4 différentes topologies est : topologie 01 :14 nœuds, topologie 02 :15 nœuds, topologie 03 :17 nœuds, topologie 04 :20 nœuds.

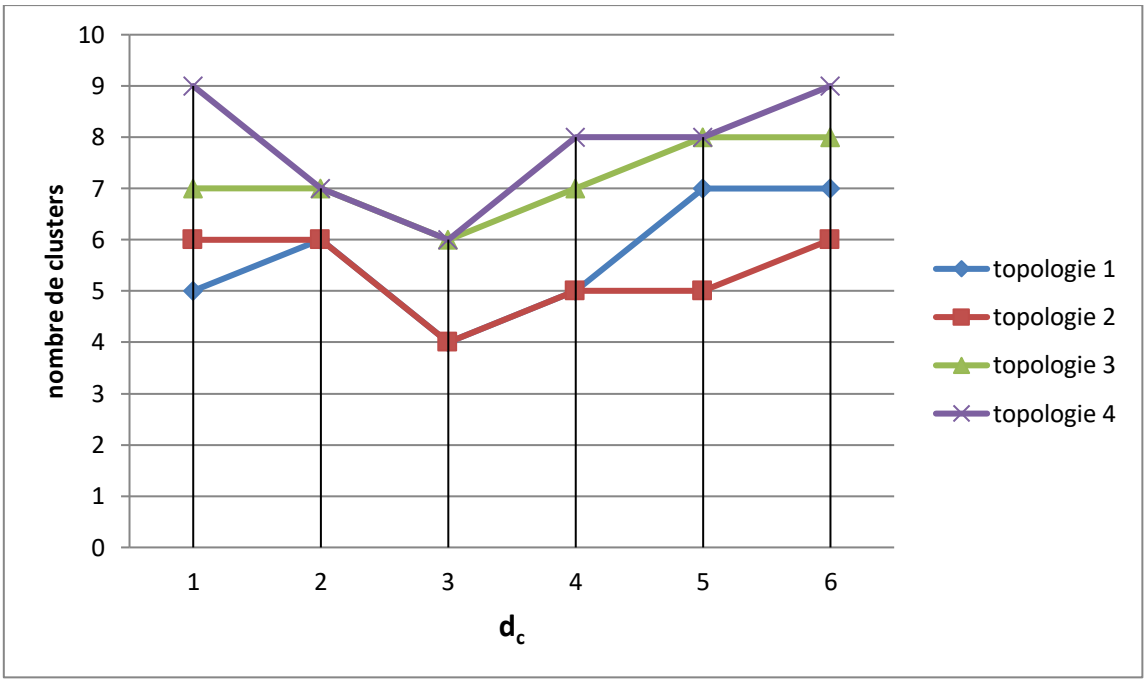

Figure 3.8 - la variation du nombre de clusters en fonction de d_c.

La figure .38 montre comment le nombre de clusters varie sous différents valeurs de d_c. Chaque ligne représente une topologie et le nombre de clusters est le même que le nombre de contrôleurs.

Le nombre de clusters est grand lorsque d_c est trop grand ou trop petit. En effet, si d_c est trop petit, la densité de chaque commutateur est limitée dans un rayon court, ce qui fait que les commutateurs ont des densités similaires et réduit la taille du cluster et donc un nombre de clusters grand. Si elle est trop grande, la densité des commutateurs se rapprochera du nombre de tous les commutateurs, ce qui rend également la taille du cluster beaucoup plus petite.

6. Conclusion

Dans ce dernier chapitre, nous avons présenté l'implémentation de l'algorithme proposé qui est basé sur la densité pour trouver le nombre optimal et le placement des contrôleurs d'un réseau SDN. Nous avons exposé les résultats des travaux effectués pour évaluer les performances de cet algorithme.

Les résultats obtenus montrent que l'algorithme DBCP permet de déterminer le nombre de contrôleurs nécessaires et d'identifier leurs positions sur le réseau. DBCP n'a besoin que de configurer un paramètre d_c, il peut donc être implémenté facilement. En outre, nous pouvons implémenter pour différentes applications en modifiant la fonction de distance (fonctions d'objectif). Ainsi, pour obtenir la solution de placement du contrôleur, DBCP n'a besoin que d'exécuter l'algorithme en une seule fois. Au contraire, la plupart des approches conventionnelles obtiennent les solutions en utilisant une méthode itérative.

Conclusion générale

Le problème de placement du contrôleur (CPP) est l'un des principaux défis des réseaux programmables pour augmenter les performances.

Pour résoudre ce problème nous avons choisi le DBCP (Density Based Controller Placement) qui utilise un algorithme de clustering de commutateurs basé sur la densité pour diviser le réseau en plusieurs sous-réseaux.

Dans ce travail, nous avons implémenté l'algorithme par le langage de programmation c++, et les expériences faites sur certains nombre de réseaux de différentes topologies. Après avoir inséré les valeurs d'entrée, nous avons obtenu l'ensemble du cluster, le nombre optimal des contrôleurs et les emplacements des contrôleurs qui minimisent la latence entre les commutateurs et les contrôleurs.

Dans les travaux futurs, l'optimisation de plusieurs objectifs sera considérée pour le placement du contrôleur. 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] Amel Haji, "Architecture de Cloud Orientée Service", doctorat en technologies de l'information et de la communication, L'Ecole Supérieure des Communications de Tunis, disponible : http://www.supcom.mincom.tn/Fr/upload/1522930084.pdf.

[2] M.Chiosi et all. "Network Functions Virtualisation - An Introduction, Benefits, Enablers, Challenges and Call for Action", SDN and OpenFlow World Congress, Darmstadt-Allemagne, vol. 48, pp.5, October 22-24, 2012.

[3] ETSI, "Network Functions Virtualisation (NFV), Architectural Framework", GS NFV 002 (v. 1.1.1), 2013.

[4] Ayankoya F.Y et al, "Appraisal on Cloud Computing and Network Functions Virtualization", IJCSNS International Journal of Computer Science and Network Security, VOL.19, No.7, pp. 42, 2019.

[5] Jérôme Durand, "Le SDN pour les nuls", Montpellier, pp.02, JRES 2015.

[6] M. Latah et L. Toker, "Artificial Intelligence Enabled Software Defined Networking: A Comprehensive Overview", arXiv:1803.06818v3, pp.02, 2018.

[7] Thomas Paradis, "Software-Defined Networking", mémoire de Master, School of Information and Communication Technology KTH Royal Institute of Technology Stockholm, Suède, pp.05-11, 2014.

[8] Marcial P. Fernandez, "Evaluating OpenFlow Controller Paradigms", La douzième conférence internationale sur les réseaux 2013 Barcelone, Espagne, pp.153, 2013.

[9] T. Issa et al, "Etude du nomadisme dans un Cloud éducatif administré par la technologie SDN/OpenFlow", Institut de recherches mathématique, Université Félix Houphouët-Boigny, conférence WACREN 2016, pp.04, 2016.

[10] L. Valdivieso et al, "SDN / NFV Architecture for IoT Networks", 2018.

[11] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem" inProc. 1st ACM Workshop HotSDN , 2012.

[12] 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.

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

[14] 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.

[15] G. Ishigaki et al, "Cluster Leader Election Problem for Distributed Controller Placement in SDN," in GLOBECOM 2017-2017 IEEE Global Communications Conference, pp.01-06, 2017.

[16] 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.

[17] V. Ahmadi et M. Khorramizadeh, "An adaptive heuristic for multi-objective controller placement in software-defined networks", Computers and Electrical Engineering 66(2018) 204–228, pp.208-224, 2018.

[18] B. Zhang, X. Wang et M. Huang, "Multi-objective optimization controller placement problem in internet-oriented software defined network", Computer Communications 123 (2018) 24–35, pp.26-34, 2018.

[19] M. Tanha et al, "Capacity-aware and delay-guaranteed resilient controller placement for software-defined WANs", IEEE Transactions on Network and Service Management, pp.04-13, 2018.

[20] 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.

[21] O. Flauzac et al, "Secure Exchanges Activity in Function of Event Detection with the SDN", pp.05, 2019.

[22] Embarcadero, disponible: https://www.embarcadero.com/products/cbuilder, consulté 18/08/2020.

ملخص:

يعتبر هذا العمل وضع وحدات التحكم للشبكات المبرمجة. الهدف هو العثور على العدد الأمثل وكذا مواقع وحدات التحكم التي تقلل من زمن الانتقال بين المحولات وأجهزة التحكم. لتحقيق هذا الهدف ، اخترنا طريقة وضع وحدة التحكم المعتمدة على الكثافة "DBCP ". تظهر النتائج التجريبية أن DBCP يقدم أداء أفضل ويمكن تنفيذه بسهولة وللتطبيقات المختلفة.

الكلمات المفتاحية: الشبكات المبرمجة ، وحدة تحكم ، مشكلة وضع وحدة التحكم ، وضع وحدة التحكم القائمة على الكثافة(DBCP) .

Abstract:

This work considers the placement of controllers for Software Defined Networks (SDN). The goal is to find the optimum number and placements of controllers that minimize latency between switches and controllers. To achieve this goal, we have chosen the Density Based Controller placement (DBCP) approach. The experimental results show that DBCP offers better performance; it can be implemented easily and for different applications.

Keywords: Software Defined Networks, Controller, Controller placement problem (CPP), Density Based Controller placement (DBCP).

Résumé:

Ce travail considère le placement des contrôleurs pour les réseaux programmable (SDN). L'objectif est de trouver le nombre et les emplacements optimaux des contrôleurs qui minimisent la latence entre les commutateurs et les contrôleurs. Pour atteindre cet objectif, nous avons choisi l'approche de placement du contrôleur basé sur la densité (DBCP). Les résultats expérimentaux montrent que DBCP offre de meilleures performances, il peut être implémenté facilement et pour différentes applications.

Mots-clefs : réseau programmable, Contrôleur, Le problème de placement des contrôleurs, le placement du contrôleur basé sur la densité (DBCP).