Coretic : Nouvelle écriture des règles pour améliorer l'isolation et la composition en SDN

par

Moussa KABA

MÉMOIRE PRÉSENTÉE À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
COMME EXIGENCE PARTIELLE À L'OBTENTION DE LA MAÎTRISE
AVEC MÉMOIRE EN TECHNOLOGIE DE L'INFORMATION
M. SC. A.

MONTRÉAL, LE 5 AVRIL 2018

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

M. Talhi Chamseddine, directeur de mémoire
Département de génie logiciel et des TI à l'École de technologie supérieure

M. Ségla Kpodjedo président du jury
Département de génie logiciel et des TI à l'École de technologie supérieure

M. Gherbi Abdelouahed, membre du jury
Département de génie logiciel et des TI à l'École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 12 MARS 2018

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

Je tiens tout d'abord à remercier l'école ÉTS qui m'a accordé une admission pour pouvoir effectuer ma maîtrise au sein de l'établissement. Les premiers remerciements vont aussi à l'encontre de mon directeur d'étude, Monsieur Talhi Chamseddine, qui m'a fait confiance en me confiant un projet qui demande beaucoup d'efforts. Il a trouvé tout au long du projet des mots pour m'encourager et me guider en vue de surmonter des difficultés et aussi de bien cerner les contextes du travail effectué. J'aimerais aussi remercier Monsieur Ledjiar Abderrahmane, étudiant en doctorat, qui a travaillé avec moi tout au long du projet de maîtrise. Il a apporté sa contribution à l'aboutissement de ce travail en m'aidant à trouver des pistes de solution. J'aimerais aussi remercier Enrique Torrealba et Emmanuel Sampin qui furent des collègues de travail en maîtrise. J'ai su bénéficier de certaines de leurs expériences en réseau informatique, tests de performance en SDN et de l'utilisation de certains outils. Mes remerciements aussi vont à l'encontre de Madame Christine Richard du service aux étudiants. Elle a été d'une aide considérable pour pouvoir obtenir les premières aptitudes de rédaction de mémoire. Mes reconnaissances vont à l'encontre de mon épouse qui m'a toujours encouragé dans les moments difficiles et qui a cru en mes capacités d'atteindre mes objectifs dans ce travail. Pour finir, j'aimerais remercier Messieurs Alain Lavoie et Vladimir Aristizabal, respectivement directeur et gestionnaire au département de sécurité informatique de Vidéotron pour m'avoir aidé à terminer cette maîtrise en me donnant la latitude de me rentre à l'école pendant mes rencontres et à faires des impressions. J'aimerais aussi remercier tous ceux que j'aurais oublié de citer.

CORETIC : NOUVELLE ÉCRITURE DES RÈGLES POUR AMÉLIORER L'ISOLATION ET LA COMPOSITION EN SDN

Moussa KABA

RÉSUMÉ

SDN (*Software Defined Networking*) est une nouvelle architecture réseau qui permet d'apporter une solution à la complexité des tâches des équipements réseaux. Cette architecture permet de séparer le plan de données, chargé d'acheminer les données, du plan de contrôle qui fournit les règles de gestion des flux. Au-dessus du plan de contrôle, s'ajoute le plan de gestion qui permet d'envoyer des instructions au plan de données en utilisant un langage évolué. Les travaux de recherche apportent des solutions pour permettre l'isolation des trafics, la composition des plans de contrôle et pour la gestion des flux au niveau du plan de gestion en SDN. Les solutions actuelles d'isolation et de composition présentent des techniques de gestion des entrées de flux ayant des faiblesses. Certaines utilisent des informations de la couche 2 du modèle OSI (*Open System Interconnection*) (Ahmed, Mohamed Fekih. 2015). Effectuer l'isolation en se servant des informations de la couche 2 entraîne une génération de plusieurs entrées dans les tables de flux, ce qui affecte les performances. D'autres solutions utilisent une seule table de flux pour insérer les règles de plusieurs contrôleurs (Jin, Xin. 2015). Ceci entraîne une cohésion faible des tables concernées. Un couplage élevé est aussi constaté par l'utilisation de plusieurs tables reliées entre elles (Dixit, A. 2014). Quant aux solutions au niveau du plan de gestion, elles ne permettent pas à plusieurs plateformes de programmation de haut niveau d'utiliser un contrôleur commun.

Dans ce travail de maîtrise, il est proposé Coretic qui vise à améliorer l'écriture des règles dans le plan de données pour l'isolation et la composition. Étant donné que le rôle du plan de contrôle est confié au plan de gestion utilisant les plateformes de programmation de haut niveau, ce travail propose une solution permettant d'isoler le trafic des plans de gestion en utilisant un contrôleur commun. Coretic pour atteindre ses objectifs se sert des informations de la couche 3 du modèle OSI et de plusieurs tables de flux. Des tests de performance ont permis de montrer que Coretic offre de meilleures performances dans l'isolation et la composition. La solution d'*hypervision* des plans de gestions ne crée pas une dégradation importante des performances. Coretic à la suite des tests effectués apporte les contributions suivantes :

- Apport d'une nouvelle approche d'isolation et de composition de plans de contrôles basée sur l'utilisation de tables multiples et de l'utilisation d'adresses IP de la couche 3.
- Apport d'une solution d'*hypervision* des plans de gestion.

Ces contributions permettent d'améliorer les performances dans le plan de données. La cohésion des tables de flux est renforcée, car ces tables reçoivent chacune des politiques bien précises. Le couplage devient faible aussi, car Coretic laisse les tables traiter les flux en toute

indépendance. Avec Coretic, il est possible d'utiliser qu'un seul contrôleur pour isoler le trafic de plusieurs plans de gestion.

Mots clés : SDN, Openflow, Isolation, composition, plan de contrôle, plan de données, plan de gestion, *hypervision*, table de flux, mode réactif, mode proactif, mode interprété.

CORETIC: NEW RULES WRITING TO IMPROVE THE ISOLATION AND COMPOSITION IN SDN

Moussa KABA

ABSTRACT

SDN (Software Defined Networking) is a new network architecture that provides a solution to the complexity of network equipment tasks. This architecture allows separating the data plan, responsible for conveying the data, from the control plan that provides rules for flows management. Above the control plan is added the management plan that allows sending instructions to the data plan using advanced languages. The research works provides solutions to enable the isolation of traffics, the composition of control plans and management of flows at the level of the management plan in SDN. Current isolation and composition solutions present flow management techniques with weaknesses. Some of these solutions use Layer 2 information of the OSI (Open System Interconnection) model (Ahmed, Mohamed Fekih, 2015). Performing isolation using Layer 2 information results in the generation of multiple entries in the flow tables, which affects performances. Other solutions use a single flow table to insert the rules of multiple controllers (Jin, Xin, 2015). This results in weak cohesion of the tables concerned. High coupling is also found by the use of several linked tables (Dixit, A. 2014). Concerning management plan solutions, they do not allow several high-level programming platforms to use a common controller. In this mastery work, it is proposed Coretic that aims to improve the writing of rules in the data plane for isolation and composition. Since the role of the control plan is assigned to the management plan using high-level programming platforms, this work provides a solution to isolate traffic of management plans using a common controller. Coretic, to achieve his goals, uses layer 3 informations from the OSI model and multiple flow tables. Performance tests have shown that Coretic offers better performance in isolation and composition. The hypervision solution of management plan does not create a significant degradation of performance. Coretic as a result of the tests carry out the following contributions:

- Providing new approach for isolating and composing control plans based on the use of multiple tables and the use of layer 3 IP addresses.
- Providing a hypervision solution management plans.

These contributions improve performance in the data plan. The cohesion of the flow tables is reinforced because each of these tables receives specific policies. The coupling becomes weak too, because Coretic allows the tables to treat flows independently. With Coretic, it is possible to use only one controller to isolate traffic from multiple management plans.

Keywords: SDN, Openflow, Isolation, Composition, Control Plan, Data Plan, Management Plan, Hypervision, Flow Table, reactif mode, Proactive Mode, Interpreted Mode.

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

SDN *Software Defined Networking*

TCP *Transport Control Protocol* (protocole de contrôle de transport)

UDP *User Datagram Protocol* (protocole de datagramme utilisateur)

IP *Internet Protocol*(protocole internet)

VLAN *Virtual Local Area Network* (réseau local virtual)

DST Destination

SRC Source

ARP *Address Resolution* Protocol (protocole de résolution d'adresse)

ICMP *Internet Control Message Protocol* (Protocole de message de contrôle sur Internet)

API *Application Programming Interface* (interface de programmation applicative)

OSI *Open System Interconnection*

OVX Openvirtex

SQL *Structured Query Language*

VLAN *Virtual local arean network* **(**réseau local virtuel**)**

LISTE DES SYMBOLES ET UNITÉS DE MESURE

Ko	Kilo Octet
Pkt	Paquet
Pkts/s	Paquets par seconde
Mbits/s	Mégabits par seconde
s	Seconde
ms	Milliseconde

INTRODUCTION

Les réseaux informatiques sont construits à l'aide d'équipements d'interconnexion comme des commutateurs. Ces équipements remplissent deux rôles qui sont celui de prise de décision et celui d'acheminement de données. Pour acheminer les données, les équipements doivent se servir des règles implémentées en leur sein. Ceci représente une complexité. En plus de la complexité des tâches présentées, les langages de programmation de prise de décision diffèrent d'un modèle d'équipement à l'autre. Dans le but d'apporter une solution à la situation exposée, une nouvelle architecture réseau sous le nom de SDN (*Software Defined Networking*) ainsi que des protocoles de communication ont été créés. L'architecture SDN permet de séparer les équipements qui acheminent les données des équipements qui prennent les décisions. Le plan de contrôle fournit les directives de traitement des flux. Le plan de données achemine les flux. Openflow, le protocole le plus utilisé, permet au plan de contrôle et le plan de données de communiquer (Kreutz, D., et al. 2014). Il existe un troisième plan appelé plan de gestion qui permet de faire une abstraction du plan de contrôle. L'architecture SDN incluant le protocole Openflow présente des avantages par rapport à l'architecture traditionnelle (Kreutz, D., et al. 2014). Nous avons entre autres l'amélioration du délai de traitement des données, car les équipements du plan de données n'ont plus à exécuter des algorithmes complexes pour traiter les données. En raison des avantages de SDN, différents projets de recherche ont été menés. Comme exemples de projets, il y a Flowvisor (Sherwood, et al. 2009), Sdnms (Ahmed, Mohamed Fekih, et al. 2015).

Il faut pouvoir mettre en place des mécanismes permettant l'isolation des flux des contrôleurs. Dans le domaine de la recherche en SDN, un des premiers à pouvoir fournir une solution est Flowvisor (Sherwood, et al. 2009). Il permet de subdiviser les équipements du plan de données entre les différents clients utilisant des contrôleurs différents. Par la suite d'autres travaux ont suivis. Par exemple : Sdnms (Ahmed, Mohamed Fekih, et al. 2015) qui propose l'isolation des flux de plusieurs contrôleurs en se servant des informations de la couche 2 du modèle OSI et de plusieurs tables. Il y a aussi OVX (Openvirtex) (Al-Shabibi et al. 2014) qui propose la

gestion des flux des contrôleurs à l'aide d'une seule table de flux et en se servant des informations de la couche 2 et 3 du modèle OSI. D'autres recherches proposent des solutions à la composition des équipements du plan de contrôle dans le but de distribuer les prises de décision. Comme exemple, il y a Flowbricks (Dixit, A, et al. 2014) qui propose une solution permettant la composition parallèle et séquentielle en utilisant des tables de transition. Covisor (Jin, Xin, et al. 2015) propose une composition parallèle et séquentielle, avec l'utilisation d'une seule table dans le plan de donnée. Outre l'isolation et la composition, l'élaboration des plateformes pour la programmation de haut niveau au niveau du plan de gestion a fait l'objet de recherches. L'une des premières à avoir été développée est Frenetic (Foster, Nate, et al. 2011). Elle permet de faire abstraction du langage Openflow en offrant un langage de haut niveau basé sur le langage SQL (*Structured Query Language*). Elle permet de composer plusieurs modules tout en gérant les conflits entre les règles soumises. D'autres travaux se sont inspirés de Frenetic. Parmi eux, Pyretic (Reich, Joshua, et al. 2013) qui offre plusieurs modes d'exécution des politiques avec une composition plus élaborée entre les modules de gestion de flux. Les solutions d'isolation, de composition et des plateformes de programmation de haut niveau citées précédemment présentent des faiblesses. Ces faiblesses sont expliquées par les points qui suivent.

- **Au niveau de l'isolation**

Pour l'isolation, il y a utilisation des informations de la couche 2 du modèle OSI comme dans Sdnms. Ce qui fait en sorte qu'il y a génération de plusieurs règles de flux causant un problème de performance. Il y a aussi l'utilisation d'une seule table de flux comme dans OVX. Ceci fait baisser la cohésion de ces tables.

- **Au niveau de la composition**

L'utilisation d'une seule table est faite avec Covisor. La cohésion est ainsi faible. Avec Flowbricks, il y a augmentation de couplage avec l'utilisation de tables intermédiaires.

- **Au niveau des plateformes de programmation de haut niveau**

L'*hypervision* qui consiste à gérer et centraliser le partage des ressources pour plusieurs utilisateurs est prise en compte dans ce projet. Aucune solution d'*hypervision* pour les plateformes de programmation de haut niveau n'est connue à ce jour. Tout comme Frenetic, Pyretic offre un seul espace de programmation. C'est-à-dire qu'il ne peut s'utiliser avec plusieurs plateformes à la fois.

Vus les problèmes identifiés ci-haut, les objectifs principaux sont les suivants : améliorer les solutions d'écriture des règles pour l'isolation et la composition entre les plans de contrôles, permettre l'*hypervision* entre les plateformes de programmation de haut niveau. Les objectifs principaux se traduisent par les sous objectifs ci-après :

- **Amélioration des solutions d'écriture des règles pour l'isolation et la composition**

Cet objectif se traduit par :

- Une amélioration des performances pour le traitement des flux.
- Une amélioration de la cohésion des tables de flux.
- Une réduction de couplage entre les tables de flux.

- ***Hypervision* entre les plateformes de programmation de haut niveau :**
  - Permettre à plusieurs plateformes de programmation de haut niveau d'utiliser un même contrôleur pour écrire leurs règles dans le plan de données.
  - Faire l'isolation entre les flux appartenant à des plateformes différentes.
  - Réduire les coûts en performance générés par l'hyperviseur spécialisé.

Pour atteindre les objectifs, la solution Coretic est proposée. Elle utilise les segments d'adresse IP, plusieurs tables dans le plan de données et des mandataires. Elle se divise en trois sous solutions : CoreticIsol, CoreticComp et CoreticHip.

CoreticIsol, pour isoler les flux se sert des segments d'adresse IP pour rediriger les flux de la table principale vers les tables appartenant à des contrôleurs. CoreticComp se sert de plusieurs tables pour la composition parallèle et séquentielle entre les politiques des contrôleurs. CoreticHip quant à elle utilise deux composants : un étant une extension des plateformes de programmation de haut niveau pour filtrer les paquets et un autre étant l'extension du contrôleur partagé pour s'assurer que les messages des plateformes sont redirigés vers les

bonnes tables. Les tests de performance pour CoreticIsol, CoreticComp et CoreticHip à effectuer permettront de confirmer les contributions suivantes :

- Apport d'une nouvelle approche d'isolation et de composition de plans de contrôles basée sur l'utilisation de tables multiples et de l'utilisation d'adresses du niveau 3.
- Apport d'une solution d'*hypervision* des plans de gestion.

Pour étayer en détail le travail effectué, le reste de ce document est réparti comme suit :

- Le chapitre 1 présente les concepts en SDN et les travaux déjà effectués en lien avec le sujet de recherche de cette maîtrise.
- Le chapitre 2 couvrira les méthodes employées pour atteindre nos objectifs.
- Le chapitre 3 contiendra les méthodes utilisées pour évaluer notre solution.
- Le chapitre 4 présentera les résultats obtenus lors de l'évaluation. Ce chapitre permettra également de faire l'analyse des résultats obtenus.

  À la suite du chapitre 4 suivra une conclusion qui va résumer le travail fait et fournira d'autres axes de recherche.

CHAPITRE 1

CONCEPTS ET REVUE DE LITTÉRATURE

Les réseaux traditionnels informatiques ayant les équipements d'acheminement de données qui assurent les rôles de supervisions, et de transmission de données (voir Figure 1.1) sont concurrencés par une nouvelle architecture. Cette nouvelle architecture, appelée SDN permet de séparer le rôle de contrôle de celui d'acheminement de données. Les algorithmes complexes ou logiques d'acheminement de flux sont retirés des équipements de transmission d'informations, ce qui permet d'augmenter les performances du réseau. En raison des retombées offertes par SDN, les travaux de recherche ont été entrepris. Il existe des travaux commerciaux et des recherches universitaires. Cette recherche se situe dans le cadre des recherches universitaires. Une première grande catégorie de recherches universitaires regroupe les travaux sur l'*hypervision* entre les contrôleurs et le plan de données. Les travaux universitaires d'*hypervision* fournissent des solutions pour la virtualisation des réseaux, l'isolation et la composition. La revue de littérature présentera d'abord les concepts en SDN dans la sous-section 1.1. Par la suite la section 1.2 va explorer les travaux d'*hypervision*. Ensuite la section 1.3 va aborder une documentation des plateformes de programmation de haut niveau. Après la section 1.3, la section 1.4 va aborder les problèmes identifiés. Le chapitre 1 finira par une conclusion qui va résumer les points abordés.

1.1 Concepts en SDN

SDN, la nouvelle architecture réseau sépare la partie qui prend les décisions appelée Plan de contrôle de la partie qui achemine les flux appelée Plan de données. Le protocole de communication le plus répandu permettant d'envoyer des messages entre le plan de contrôle et le plan de données est Openflow. Pour rendre le langage Openflow plus abstrait, une autre couche appelée plan de gestion (Kreutz, D., 2015) est placée au-dessus du plan de contrôle

(voir figure 1.2). Le reste de l'explication des concepts abordera l'architecture SDN et le protocole Openflow.

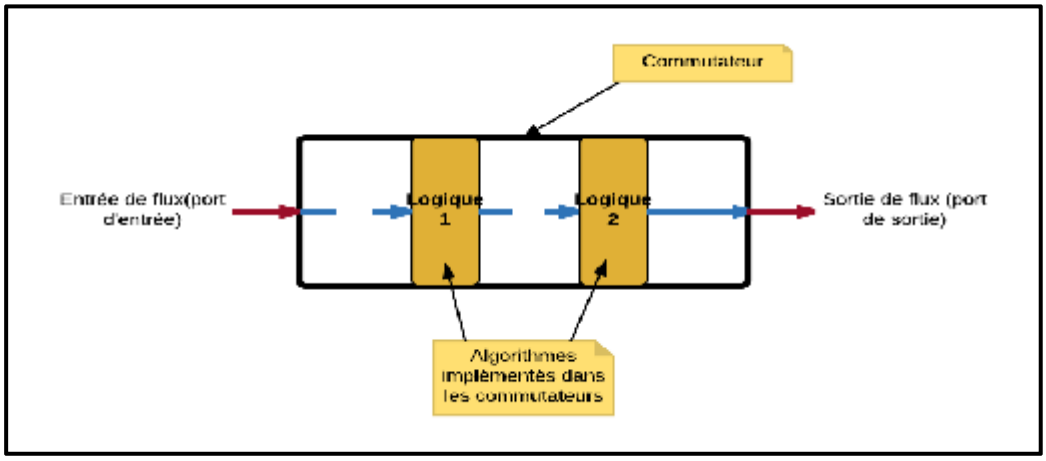


Figure 1.1 Architecture traditionnelle pour l'acheminement de flux

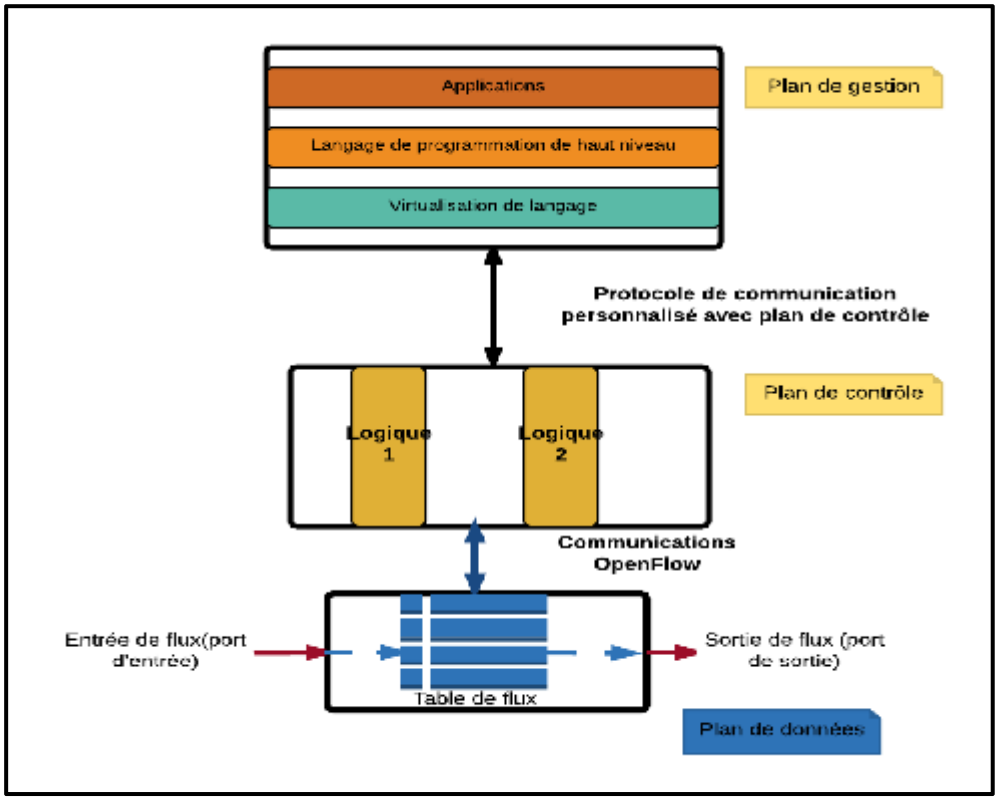


Figure 1.2 Architecture SDN

1.1.1 Architecture

SDN est essentiellement constituée de deux grandes parties qui sont le plan de contrôle et le plan de données. À ces deux parties s'ajoute le plan de gestion qui aide à mieux utiliser le plan de contrôle. Cette section donne les détails des plans dans l'architecture SDN.

1.1.1.1 Plan de données

Le plan de données est la couche de l'architecture SDN la plus basse. Elle est constituée d'équipements appelés commutateurs qui acheminent les flux entre les différents ports. Chaque machine appelée *host* est connectée au commutateur par un de ses ports. Il existe deux types de commutateurs qui sont les commutateurs physiques et les commutateurs logiciels (voir tableau 1.1). Chacun des deux types utilise des tables pour acheminer les flux de données. Ces tables sont appelées tables de flux.

1.1.1.1.1 Commutateurs physiques

Les commutateurs physiques sont des équipements physiques. Ils peuvent être utilisés par les petites, moyennes ou grandes entreprises. Ils ont de grandes capacités d'enregistrements d'instructions Openflow.

Tableau 1.1 Liste de commutateurs physiques et logiciels en SDN(Kreutz, D., 2015)

Type de commutateur	Nom du Commutateur	Fabricant	Version OpenFlow
Physique	8200zl and 5400zl	HP	1.0
	Arista 7150 Series	Arista Network	1.0
	BlackDiamond X8	Extreme Networks	1.0
	CX600 Series	Huawei	1.0
	EX9200 Ethernet	Juniper	1.0
	EZ-Chip NP-4	EZchip technologies	1.0
	MLX Series	Brocade	1.0
	NoviSwitch	NoviFlow	1.3
	NetFPGA	NETFPGA	1.0
	RackSwitch G8264	IBM	1.0
	PF5240 et PF5820	NEC	1.0
	Pica8 3920	Pica8	1.0
	Plexxi Switch 1	Plexxi	1.0

Tableau 1.1 (suite)

	V330 Series	Centex Networks	1.0
	Z-Series	Cyan	1.0
Logiciel	Contrail-vrouter	Juniter Networks	1.0
	LINC	FlowForwarding	1.4
	Ofsoftswitch13	Ericsson, CPqD	1.3
	Open vSwitch	Open community	1.0-1.3
	OpenFlow Reference	Stanford	1.0
	OpenFlowClick	Yogesh Mundada	1.0
	Switch Light	Big Switch	1.0
	Pantou/OpenWRT	Stanford	1.0
	XorPlus	Pica8	1.0

1.1.1.1.2 Commutateurs logiciels

Les commutateurs logiciels sont des simulateurs de commutateurs. Ils permettent de façon logicielle de simuler le comportement des commutateurs. Ils sont beaucoup utilisés dans le milieu de la recherche (Al-Shabibi, 2014 et Jin, Xin 2015). Plusieurs outils permettent de déployer ces commutateurs logiciels comme Mininet qui sera utilisé plus tard dans ce travail de recherche.

1.1.1.1.3 Tables de flux

Les tables de flux sont des éléments indispensables aux commutateurs. Elles permettent à ces derniers de traiter les flux. À l'intérieur de ces tables, il y a des entrées ou lignes. Ces entrées sont des instructions de traitement des flux. Un traitement peut être par exemple le blocage d'envoi de paquets à une adresse IP ou l'envoi d'une trame à un port en fonction d'une adresse MAC. Les entrées sont construites à l'aide des messages Openflow reçus du plan de contrôle à l'aide des API (*Application Programming Interface*) appelés API Nord Sud.

1.1.1.2 Plan de contrôle

Lorsque le plan de données n'est pas capable de traiter un flux, celui-ci est envoyé au plan de contrôle pour la prise de décision. Cette couche contient le contrôleur qui se charge du traitement des demandes du plan de données. Il existe deux types de contrôleurs qui sont les contrôleurs centraux et les contrôleurs distribués (voir tableau 1.2). Il y a principalement trois méthodologies d'envoi des instructions du plan de contrôle vers le plan de données qui sont le

mode proactif, le mode réactif et le mode interprété. Ces trois modes d'envoi seront vus un peu plus loin.

1.1.1.2.1 Contrôleurs centralisés et contrôleurs distribués

Les contrôleurs centralisés sont constitués d'une seule instance de contrôleur. Leurs tâches ne sont pas effectuées en parallèle. Le plan de données ne communique pas avec plusieurs contrôleurs à la fois. Il y a qu'un seul contrôleur avec qui il communique. Les contrôleurs distribués quant à eux collaborent ensemble pour répondre aux demandes du plan de données. Lorsqu'un contrôleur distribué reçoit une demande, il la traite et informe les autres contrôleurs à l'aide des API Est et Ouest. Ces API sont des interfaces offertes par les contrôleurs distribués pour une communication entre les plans de contrôle.

Tableau 1.2 Liste des contrôleurs centraux et distribués (Kreutz, D., 2015)

Type de contrôleur	Nom du contrôleur	Version OpenFlow
Centralisé	Beacon	1.0
	Floodlight	1.1
	Maestro	1.0
	Meridian	1.0
	Mul	1.0
	NOX	1.0
	NOX-MT	1.0
	POX	**1.0**
	ProgrammableFlow	1.3
	Rosemary	1.0
	SNAC	1.0
	Trema	1.0
Distribué	DISC0	1.1
	Elasticon	1.0
	Fleet	1.0
	HP VAN SDN	1.0
	HyperFlow	1.0
	Kandoo	1.0

Tableau 1.2 (suite)

	Onix	**1.0**
	NVP Controller	Non précisé
	OpenDaylight	1.0-1.3
	ONOS	1.0
	PANE	Non précisé
	SMaRtLight	1.0
	Yanc	
Non précisé	MobileFlow	1.2
	OpenContrail	1.0

1.1.1.2.2 Méthodes d'envoi des instructions aux plans de données

Le plan de données sollicite le plan de contrôle lorsqu'il ne sait pas quoi faire avec un flux. Le plan de contrôle a trois modes d'envoi d'instructions pour indiquer au plan de données la façon de traiter un flux : le mode proactif, le mode réactif, et mode interprété.

- **Mode proactif**

Le mode proactif permet au contrôleur d'envoyer les règles au plan de données avant même l'arrivée des flux. Ces règles ne sont pas effacées. Elles sont de manière permanente inscrites dans les tables de flux. Le plan de données n'a plus besoin de contacter le plan de contrôle pour un flux identique déjà traité. Les règles envoyées en mode proactif sont de deux types : de façon dynamique et de façon statique. De façon dynamique, le contrôleur attend l'arrivée des flux avant d'envoyer les règles au plan de données. Les règles envoyées au plan de données sont inscrites de façon permanente. De façon statique, toutes les règles sont envoyées pour tous les flux dès que le contrôleur se connecte au plan de données.

- **Mode réactif**

Dans le mode réactif, le contrôleur attend d'être sollicité avant d'envoyer les règles au plan de données. La règle envoyée reste active pour un certain temps dans la table de flux avant d'être

effacée. Au vu d'un flux identique par le plan de données, le contrôleur sera sollicité de nouveau. La même règle sera envoyée au plan de données.

- **Mode interprété**

Pour le mode interprété, aucune règle de gestion des flux n'est envoyée au plan de données. C'est le plan de gestion qui traite la demande du plan de données. Après traitement de la requête soumise, le plan de gestion répond au plan de données en lui spécifiant ce qu'il y a lieu de faire. À chaque fois qu'un paquet est reçu par un commutateur dans le plan de données, une demande de traitement de ce paquet est soumise au plan de gestion par l'intermédiaire du plan de contrôle.

1.1.1.3 Plan de gestion

Le plan de gestion est une couche se situant au-dessus du plan de contrôle (voir figure 1.2). Il permet de faire abstraction du plan de contrôle. Ce plan vise plusieurs objectifs. Un des objectifs du plan de gestion est l'abstraction des instructions des API des contrôleurs. La programmation des politiques de traitement des flux en Openflow à l'aide des API des contrôleurs représente des risques d'erreurs (Reich, Joshua, et al 2013). Le plan de gestion permet de réduire ces erreurs en offrant plusieurs fonctionnalités qui sont la virtualisation du langage offert par les contrôleurs, l'utilisation d'un langage de programmation de haut niveau et des applications de gestion des flux.

1.1.1.3.1 Langage de programmation de haut niveau

Pour faciliter le travail des programmeurs SDN, le plan de gestion permet de construire les applications à l'aide des langages évolués comme Python, Java ou C++. Ces langages sont utilisés avec des plateformes de programmation de haut niveau. Ces plateformes visent différents objectifs. Il en existe qui sont spécialisées dans la production des modules ainsi qu'à la résolution de conflits entre les modules. Dans cette famille de plateformes spécialisées, il y a Frenetic et Pyretic. Ces deux plateformes sont des pionniers dans la résolution des conflits entre les modules dans le domaine de la recherche.

1.1.1.3.2 Applications de gestion

Les applications dans le plan de gestion sont de trois sortes. Il y a les applications sans composition, les applications avec composition parallèle et les applications avec composition séquentielle.

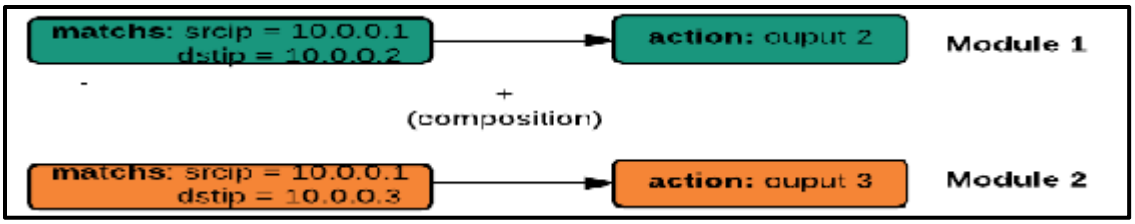
- **Application sans composition**

Une application sans composition est formée d'un seul module. Un module est comparable à une classe en programmation orientée objet ou à une méthode. Il a la responsabilité de fournir les règles pour des cas bien précis de gestion de flux. Il peut comprendre une ou plusieurs règles.

- **Application avec composition parallèle**

La syntaxe qui symbolise la composition parallèle est || ou +. Des modules en parallèle permettent de faire appliquer des règles provenant d'eux même sur les flux identiques. La composition parallèle s'applique de deux façons. Il y a la composition parallèle des conditions d'entête et la composition des règles. Dans la composition des conditions d'entêtes, pour poser une action commune, une des conditions d'entête des modules en composition parallèle est vérifiée (voir Figure 1.3). Dans l'exemple illustré dans la figure 1.3, les paquets ayant comme source l'adresse IP égale à 10.0.0.1 ou comme destination l'adresse IP égale à 10.0.0.2 sont envoyés sur le port 2 du commutateur. Une règle est composée de conditions ou Match, d'une priorité et d'une ou plusieurs actions à poser. La section 1.1.2.1 parlant du protocole Openflow donne les détails des éléments d'une règle. L'autre catégorie de composition parallèle concerne la composition parallèle entre les règles. Chaque module a son ensemble de *Match* et son ensemble d'actions. Les paquets doivent respecter les *Match* d'un module au minimum pour voir une action s'exécuter (voir figure 1.4).

Figue 1.3 Illustration de la composition parallèle avec les entêtes



Figue 1.4 Illustration de la composition parallèle avec les règles (match+actions)

- **Application avec composition séquentielle**

La composition séquentielle implique l'application de règles dans un module avant celles d'un autre module. La syntaxe qui symbolise la composition séquentielle est >>. Dans la composition séquentielle, les actions du module précédent sont exécutées avec celui qui est le suivant.

1.1.2 Protocole Openflow

Openflow est un protocole de communication entre le plan de contrôle et le plan de données. Il permet au plan de contrôle d'envoyer des instructions au plan de données. Ce dernier également fournit des informations au plan de contrôle en utilisant le même protocole. La version la plus à jour est la version 1.4. La version 1.5 n'est pas encore certifiée. Ce travail de recherche s'est limité à la version 1.0 parce qu'elle est la plus supportée par les contrôleurs (Kreutz, D., 2015). Il existe 21 types de messages en Openflow 1.0. Seulement trois types de messages seront décrits, car ce sont eux qui sont les plus récurrents. Ce sont Packetin, Packetout et Flowmod. Mais avant de donner plus de détails sur les types de messages, une description d'un paquet Openflow sera faite.

1.1.2.1 Description d'un paquet OpenFlow

Un paquet Openflow est encapsulé dans un message TCP/IP qui circule entre le plan de données et le plan de contrôle. Comme on le voit dans la figure 1.5 tirée d'une capture Wireshark, le message Openflow est encapsulé dans un paquet TCP/IP.

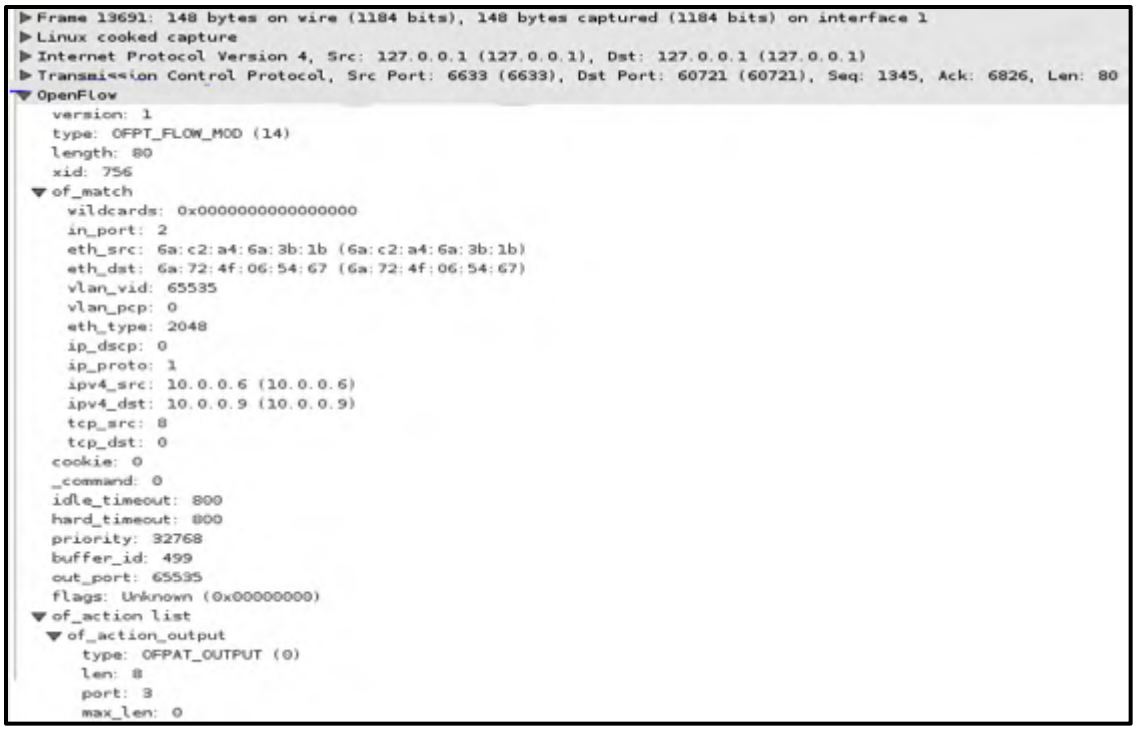


Figure 1.5 Message OpenFlow encapsulé dans un paquet TCP

Dans sa structure générale, un paquet OpenFlow composé de cinq parties : version, type, *lenght*, XID, *payload* (voir figure 1.6). La version indique la version du protocole Openflow. Le champ "type" indique le type de message (Packetout, Flowmod, Portmod etc). *Lenght* indique la taille totale du message Openflow (nombre d'octets). Dans l'exemple de la figure 1.6 si le *Lenght* est égal à 8 alors la taille totale du message Openflow est 64 bits. Dans ce cas, il n'y a pas de *Payload*. Xid est un identifiant qui permet de lier une réponse à une requête. *Payload* est un champ qui varie en fonction du type de message.

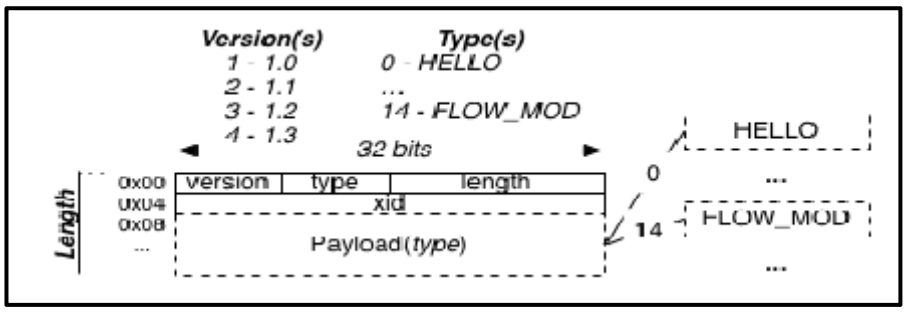


Figure 1.6 Structure générale d'un message OpenFlow tiré de http://flowgrammable.org/sdn/openflow/message-layer/

1.1.2.1.1 Packetin

Un Packetin est un message Openflow allant du commutateur au contrôleur. Il permet de rapporter un évènement au contrôleur. Par exemple lorsqu'un commutateur ne sait pas quoi faire d'un paquet qui arrive sur un port, il l'envoie au commutateur. Les champs qui décrivent un Packetin sont représentés dans la figure 1.7. Le champ "*header*" se trouve au-dessus de tous les messages comme expliqué dans la section précédente. "*buffer_id*" est un identifiant du paquet auprès du commutateur. "*total_len*" est la taille totale des données dans le champ "*data*". "*in_port*" est le port par lequel le paquet est arrivé au commutateur en question. "*reason*" indique la raison pour laquelle le paquet est envoyé au contrôleur. "*data*" représente les données à transmettre.

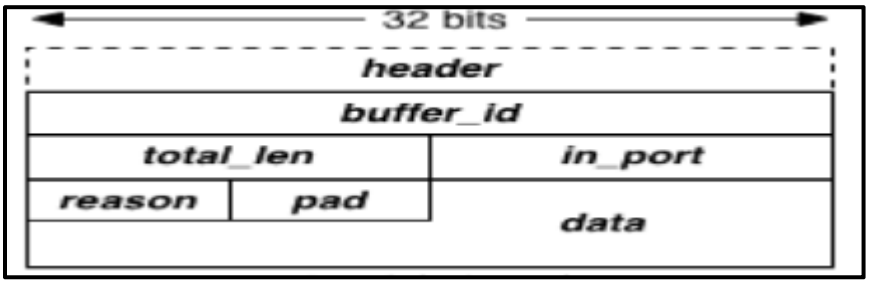


Figure 1.7 Structure générale d'un message OpenFlow Packetin
tirée de http://flowgrammable.org/sdn/openflow/message-layer/packetin/

1.1.2.1.2 Packetout

Un message Packetout est un message allant du contrôleur au commutateur. Il permet d'indiquer au contrôleur les actions à poser pour traiter un paquet donné. Un Packetout contient 6 champs (voir figure 1.8). L'entête est l'information générale qui se trouve au-dessus de chaque message Openflow. "*buffer_id*" indique l'emplacement du paquet brut. "*in_port*" indique le port d'entrée au commutateur à considérer pour le traitement du paquet. "*actions_len*" indique la taille du tableau d'actions. "action[]" est le tableau qui contient les actions. "*data*[]" est le tableau qui contient le paquet brut.

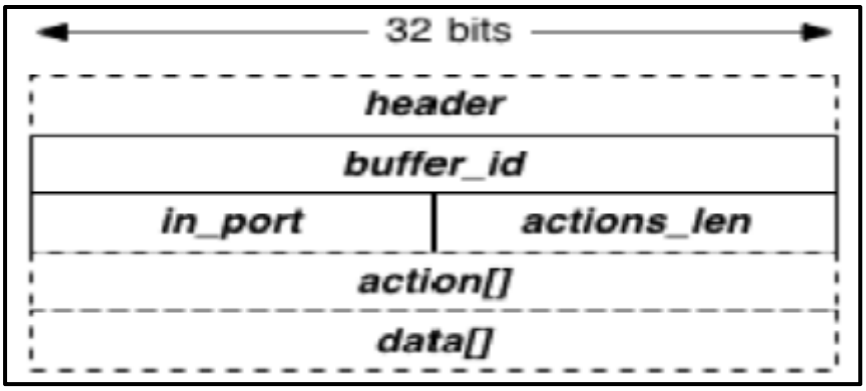


Figure 1.8 Structure un message Packetout tirée de
http://flowgrammable.org/sdn/openflow/message-layer/packetout/

1.1.2.1.3 Flowmod

Flowmod un est message qui permet au contrôleur d'écrire des règles dans les tables de flux. Le message contient plusieurs champs (voir figure 1.9) Le champ "*match*" indique la correspondance d'entête des paquets à traiter. Le champ "*cookie*" permet d'identifier le contrôleur. "*Command*" indique s'il faut soit ajouter cette nouvelle règle ou la remplacer par une autre. "*Idle_timeout*" indique le temps d'attente maximal entre deux traitements de flux avant que la règle soit effacée. Une valeur indiquant 0 signifie que la règle ne va pas s'effacer jusqu'au délai indiqué par "*Hard_timeout*". Quant à "*Hard_timeout*", il indique la durée maximale de temps d'existence de la règle dans le commutateur. Avec une valeur égale à 0, la règle reste en permanence dans le commutateur si le "*Idle_timeout*" a aussi une valeur égale à 0. "*Priority*" indique le niveau de priorité de l'application de la règle. Plus la priorité est haute et plus elle sera utilisée pour traiter des paquets. Le champ "*buffer_id*" permet de faire la correspondance entre un Packetout et un Packetin précédent. "*Out_port*" intervient dans les cas où le contrôleur envoie un message de suppression dans la table. La valeur contenue dans ce champ doit être un port configuré dans une des actions. Le champ "*flags*" sert à configurer certaines actions dans le commutateur comme le fait d'aviser le contrôleur si une ligne d'entrée est supprimée. Le tableau "action[]" contient la liste des actions à appliquer.

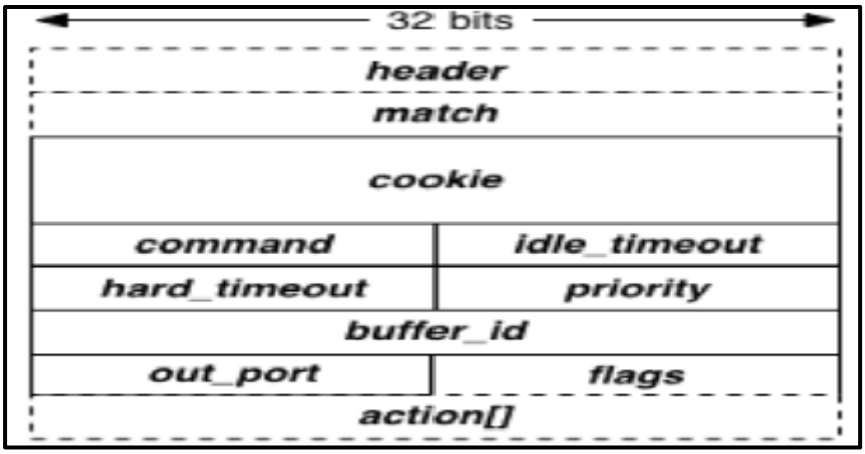


Figure 1.9 Structure générale d'un message OpenFlow Flowmod
tirée de http://flowgrammable.org/sdn/openflow/message-layer/flowmod/

1.1.2.2 Traitement du trafic dans les tables de flux

Les commutateurs dans le plan de données se servent des tables pour acheminer des paquets. Cela est possible par l'utilisation des entrées dans les tables de flux et d'un processus de traitement des paquets.

1.1.2.2.1 Les entrées dans les tables de flux

Une entrée dans la table de flux est constituée de trois sections : *Match*, compteur et action(s) (voir figure 1.10).

Figure 1.10 Trois sections d'une entrée dans la table de flux

Le *Match* est l'entête qui permet de reconnaître les paquets à traiter. Au niveau des entêtes, OpenFlow 1.0 utilise 12 champs pour faire correspondre un paquet à une entrée. Ces 12 champs d'entête sont représentés dans le tableau 1.3. Le compteur permet de connaître le nombre total des paquets traités pour une entrée. Action permet de connaître le traitement à appliquer pour les paquets dont l'entête respecte la colonne *Match*.

Tableau 1.3 Liste des 12 champs d'entête OpenFlow 1.0 tirée de http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf

Champ	Description
***Ingress Port* (in_port)**	Indique le port par lequel arrive le paquet.
***Ether* SRC**	Indique l'adresse MAC source du host qui envoie le paquet.
***Ether* DST**	Indique l'adresse MAC de destination du host qui reçoit le paquet.
Ether type	Indique le type de protocole utilisé pour une certaine trame.
VLAN ID	Champs pour indiquer un réseau virtuel.
VLAN *priority*	Indique la priorité d'une trame qui a un identifiant (id) dans le champ Vlan id.
IP SRC	L'adresse IP du host qui émet un paquet.
IP DST	L'adresse IP du host qui reçoit un paquet.
IP PROTO	Indique le protocole IP utilisé.
IP ToS bits	Indique le type de service ayant ainsi une influence sur la priorité du paquet.
TCP/UDP src port	C'est un champ qui contient deux informations à la fois. Les huit bits les moins significatifs contiennent l'information sur le type de requête ICMP. Les huit autres bits concernent le port source en se basant sur la couche transport du modèle OSI.
TCP/UDP DST port	Comme le précédent les huit bits moins significatifs contiennent le code pour une requête ICMP. Les huit autres bits contiennent le port de destination pour la couche transport du modèle OSI.

1.1.2.2.2 Processus de traitement d'un paquet

Lorsqu'un paquet ou trame arrive dans le commutateur, il y a des processus qui sont suivis. Il y a un processus à haut niveau et un autre qui est plus complexe qui est inclus dans celui qui est de haut niveau. Le processus de haut niveau permet de valider l'entête d'un paquet (voir figure 1.11)) et le processus de validité est représenté par la figure 1.12. Dans la figure 1.11, la deuxième étape est optionnelle. Elle représente le cas où il y a un *Spanning Tree*. Ensuite, le processus d'analyse d'entête est fait (voir figure 1.12). Une recherche est faite dans les différentes tables afin de trouver la bonne entrée et appliquer les actions correspondantes. Pour sélectionner une entrée, il faut que celle-ci ait son *Match* qui correspond à l'entête du paquet. Si plusieurs entrées sont trouvées dans la table de flux, la plus prioritaire est utilisée. Si deux entrées ont la même priorité, le commutateur est libre d'en choisir une. Il faut noter qu'il est

possible d'avoir plusieurs tables dans un commutateur. En OpenFlow 1.0, plusieurs tables peuvent être manipulées en utilisant un module Nicira avec le contrôleur POX.

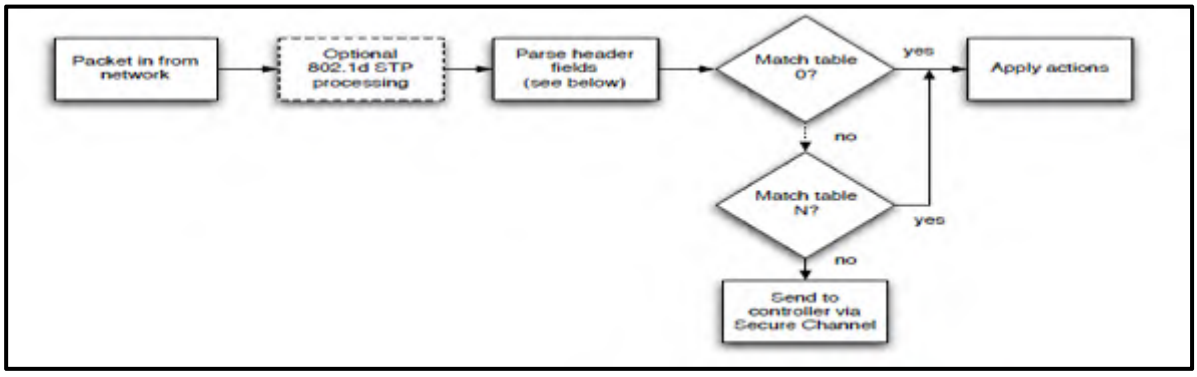


Figure 1.11 Processus de traitement de flux à haut niveau de paquet dans un commutateur tiré de http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf

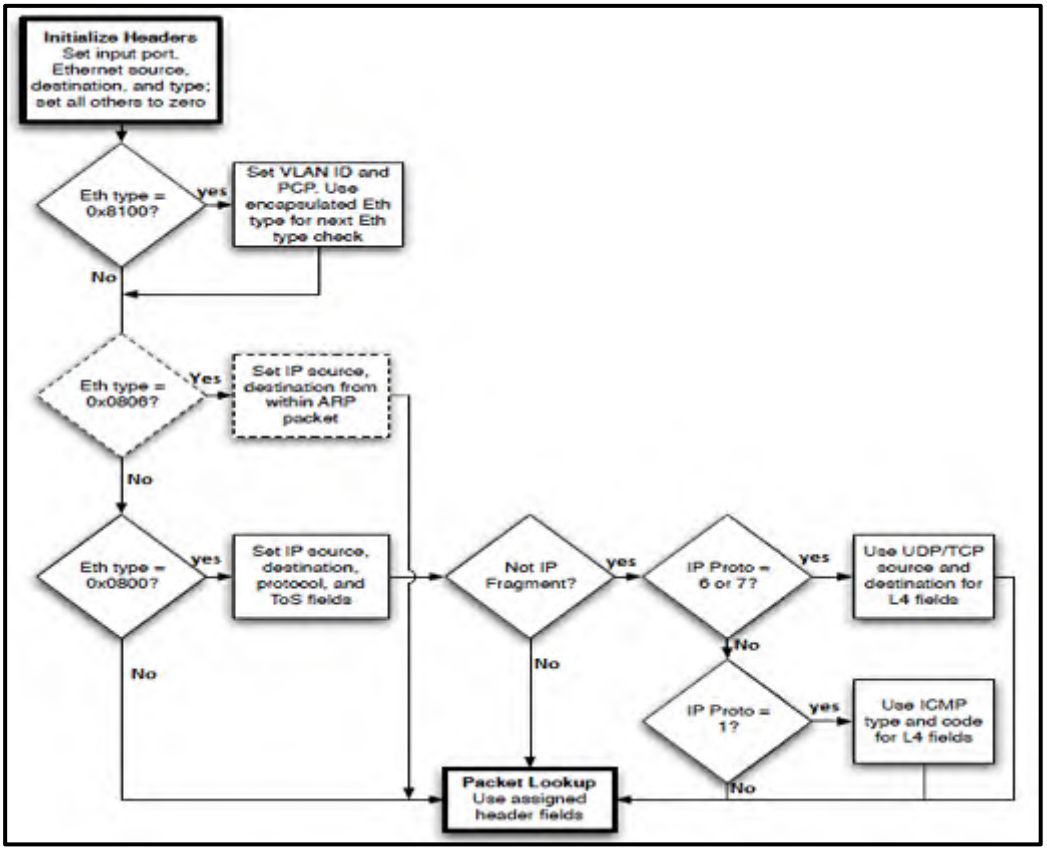


Figure 1.12 Processus de validation d'un entête, tiré de http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf

Après avoir vu l'architecture SDN ainsi que le protocole OpenFlow 1.0, la section qui suit va expliquer les solutions d'*hypervision* SDN.

1.2 Solutions d'*hypervision*

Les solutions d'*hypervision* supervisent les actions des contrôleurs vers le plan de données en SDN. Les solutions d'*hypervision* sont de deux types qui sont les solutions d'isolation et les solutions de composition. La figure 1.13 illustre le lien d'hiérarchisation entre les concepts d'*hypervision*, d'isolation et de composition. L'isolation permet d'empêcher l'envoi d'un paquet au mauvais contrôleur et à la mauvaise machine virtuelle. La composition permet à plusieurs plans de contrôle de travailler sur les mêmes trafics.

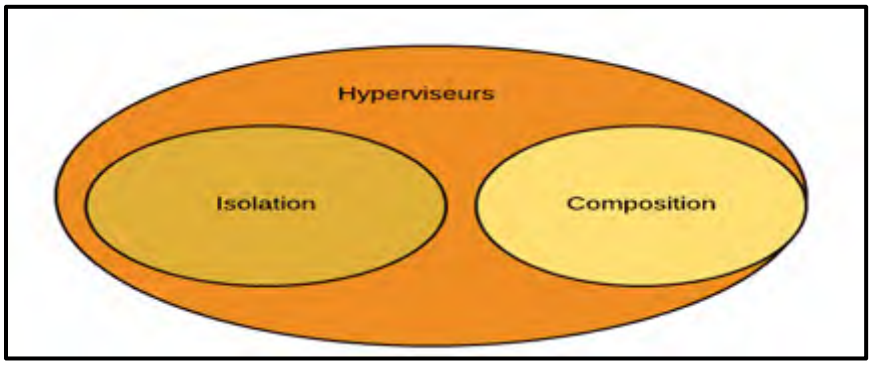


Figure 1.13 Illustration de la hiérarchisation des hyperviseurs.

Avant d'aborder l'isolation et la composition en détails, il est pertinent d'expliquer les modèles et les architectures disponibles pour faire de la virtualisation.

1.2.1 Virtualisation

La virtualisation permet de créer des réseaux virtuels au-dessus d'un autre réseau plus global. Dans un centre de données par exemple où les ressources du réseau physique sont partagées entre plusieurs clients, la virtualisation des réseaux devient indispensable. Les clients n'ont pas à se soucier des détails de la disposition des équipements ou de la topologie du réseau physique. Il existe des modèles et des architectures pour effectuer la virtualisation. Les modèles sont des techniques aidant à la virtualisation. Ils aident à faire le lien entre les réseaux virtuels et le

réseau physique. Les architectures sont les composants qui sont mis ensemble et sur lesquels sont appliqués les modèles.

1.2.1.1 Les modèles

Pour effectuer la virtualisation, des travaux de recherche proposent des modèles (voir figure 1.14). Six modèles sont présentés ici, ce sont Flowvisor, l'utilisation d'une ou de plusieurs instances Openflow, l'utilisation d'instances Openflow ayant chacune un *Datapath*, la virtualisation à niveaux multiples et l'utilisation d'un contrôleur maître (figure 1.15).

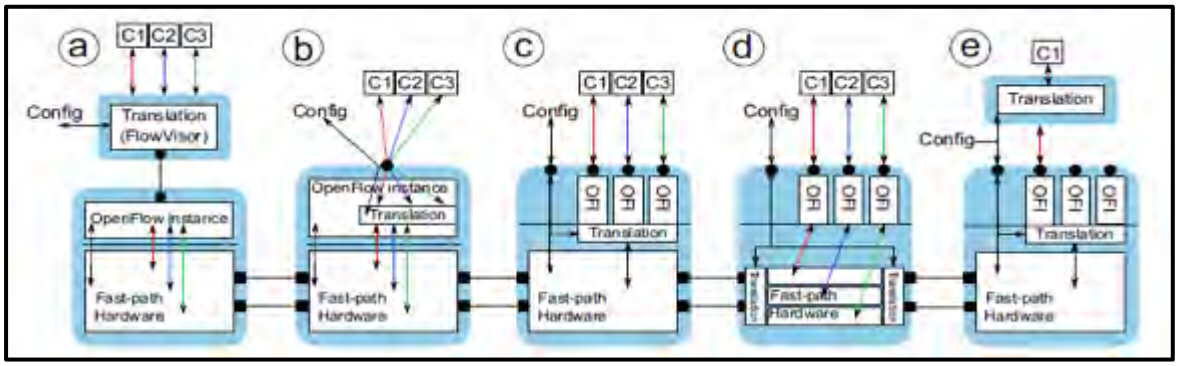


Figure 1.14 Modèles de virtualisation de la couche physique**.** (Sköldström, Pontus. 2012)

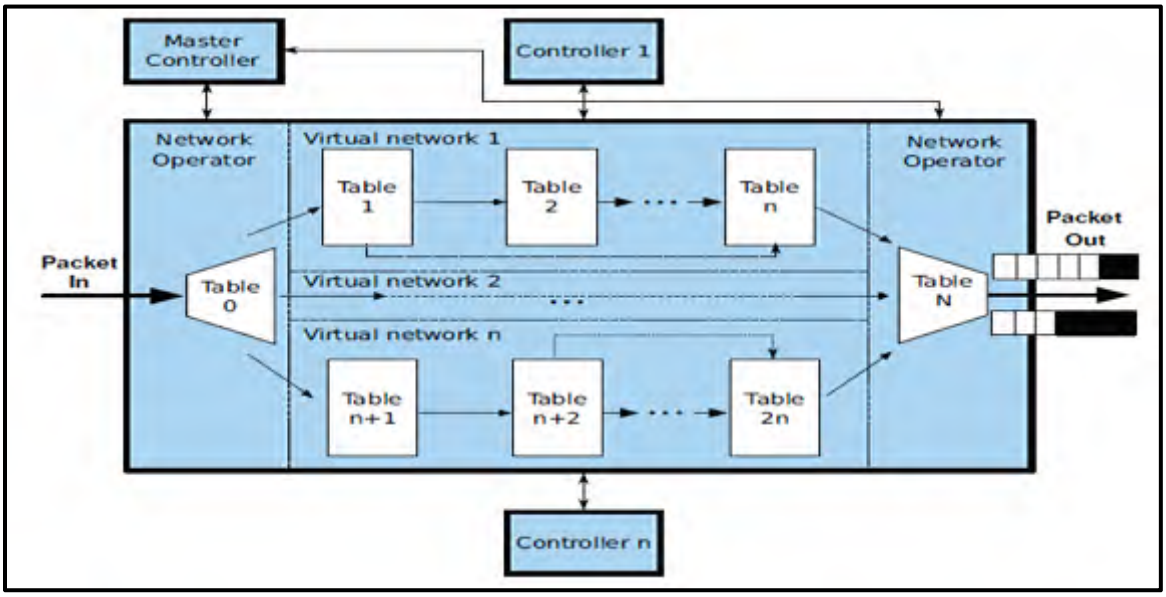


Figure 1.15 Modèle de virtualisation avec contrôleur maître

1.2.1.1.1 Modèle 1 : Flowvisor

Flowvisor (Sherwood, Rob. et al. 2009) consiste à implanter un mandataire entre les contrôleurs et les commutateurs (voir figure 1.14, partie a). La translation pour faire la correspondance entre les éléments virtuels et les éléments physiques est faite au niveau du mandataire Flowvisor lui-même. Cette solution s'assure de séparer les éléments du réseau physique entre les clients. Cette façon de faire assure efficacement le partage des ressources entre les plans de contrôle, car la virtualisation se fait au niveau physique.

1.2.1.1.2 Modèle 2 : Utilisation d'une instance OpenFlow

L'utilisation d'une instance Openflow dans les commutateurs (Sköldström, Pontus. et al. 2012) permet d'acheminer les messages vers les bonnes ressources physiques en fonction des contrôleurs (figure 1.14, partie b). Les translations sont faites au sein même des commutateurs avec l'instance Openflow. Cette solution élimine l'utilisation d'un mandataire en se servant d'Openflow.

1.2.1.1.3 Modèle 3 : Utilisation de plusieurs instances Openflow

L'utilisation de plusieurs instances Openflow (figure 1.14, partie c) permet aux contrôleurs d'avoir un certain niveau de contrôle sur les processus qui reçoivent leurs règles dans le commutateur. Par la suite, une translation est faite en dessous des instances Openflow en vue d'associer les ressources physiques aux différents contrôleurs.

1.2.1.1.4 Modèle 4 : Utilisation de plusieurs instances Openflow avec *Datapath*

Un *Datapath* est un équipement qui a la capacité de transmettre rapidement des paquets. Dans l'architecture SDN, il est représenté par le commutateur qui contient les tables de flux. Ces tables sont utilisées pour transmettre les paquets rapidement. Les prises de décisions sont confiées au contrôleur. Le modèle dont il est question dans cette section permet de créer plusieurs Datapath virtuels en raison d'un par instance Openflow (figure 1.14, partie d).

1.2.1.1.5 Modèle 5 : Virtualisation à niveaux multiples

Cette solution propose une autre approche qui consiste à permettre plusieurs niveaux de virtualisation (figure 1.14, partie e). Cette virtualisation est faite à la fois à l'intérieur et à l'extérieur des commutateurs. Par exemple il y a ADVisor (Salvadori, E. et al. 2011) qui permet la virtualisation au-delà du commutateur et ainsi permettre le partage des ressources pour une même entité donnée à savoir un client. Lorsque le réseau d'une compagnie donnée est virtuel, elle a la possibilité de créer des sous-réseaux virtuels en se servant d'une instance Openflow. Ce modèle est utile dans le contexte où une compagnie veut étendre son réseau virtuel en dehors de celui offert par un centre de données.

1.2.1.1.6 Modèle 6 : Utilisation d'un contrôleur maître

Pontus Sköldström (2012) propose un modèle flexible qu'il juge idéal (figure 1.15). Il consiste à créer plusieurs instances Openflow, à savoir d'une par réseau virtuel. Chaque instance écrit ses règles dans une table dédiée. Le contrôleur maître assure la virtualisation en autorisant les contrôleurs à écrire que dans les tables assignées.
Les modèles sont des techniques qui permettent de créer des réseaux virtuels. Ils s'appliquent sur des architectures.

1.2.1.2 Les architectures

Les architectures regroupent les composants qui permettent d'obtenir des réseaux virtuels. Les architectures présentées se spécialisent dans la flexibilité pour la gestion des réseaux virtuels. La flexibilité est un élément important en virtualisation, car elle permet une manipulation plus facile des réseaux virtuels. Dans cette sous-section cinq types d'architecture sont présentés, ce sont : Flown, architecture à couches de gestion multiples, Flowvisor, architecture de virtualisation au niveau de la couche 2, architecture avec contrôleur maître.

1.2.1.2.1 Architecture 1 : Flown

Drutskoy, D. A. (2012) propose une architecture dans laquelle un seul contrôleur appelé FlowN est utilisé. Il permet de gérer plusieurs applications appartenant à des clients différents. Une base de données est utilisée pour faire la correspondance entre les ressources physiques et ressources virtuelles (figure 1.16). Cette solution offre une gestion saine des correspondances entre les ressources virtuelles et physiques, car elle utilise une base de données qui offre une bonne sauvegarde structurée. Mais l'utilisation d'une base de données peut générer des délais supplémentaires.

1.2.1.2.2 Architecture 2 : Couches de gestion multiples

Sonkoly, B (2012) se sert de OpenNms (OPENNMS GROUP. Repéré le 7 octobre 2016 de www.opennms.org) pour créer des commutateurs virtuels servant à créer des réseaux virtuels. Le *Framework* bâti à l'aide de OpenNms permet de centraliser la gestion des commutateurs virtuels et physiques. Il permet aussi une configuration des contrôleurs utilisant des commutateurs virtuels. Cette solution est pertinente dans la situation où une gestion centralisée des ressources est importante.

1.2.1.2.3 Architecture 3 : Flowvisor

Sherwood(2009) présente Flowvisor. Cette solution présente une architecture dans laquelle une composante est insérée entre les commutateurs et les contrôleurs. La composante se comportant comme un hyperviseur permet efficacement de diviser les ressources physiques entre les contrôleurs.

1.2.1.2.4 Architecture 4 : Virtualisation à la couche 2

Matias, J (2011) utilise des gestionnaires d'adresses MAC pour créer des réseaux virtuels. Cette solution est comparable à Flown. Mais Flown utilise une base de données pour assurer la virtualisation alors que la solution de Matias, J(2011) propose un générateur d'adresse MAC. Chaque réseau virtuel a la possibilité d'étendre son environnement en manipulant une portion de l'adresse MAC avec un gestionnaire interne. Ceci permet d'obtenir des réseaux virtuels. La solution présentée ici opère à un niveau d'abstraction qui est bas (niveau 2).

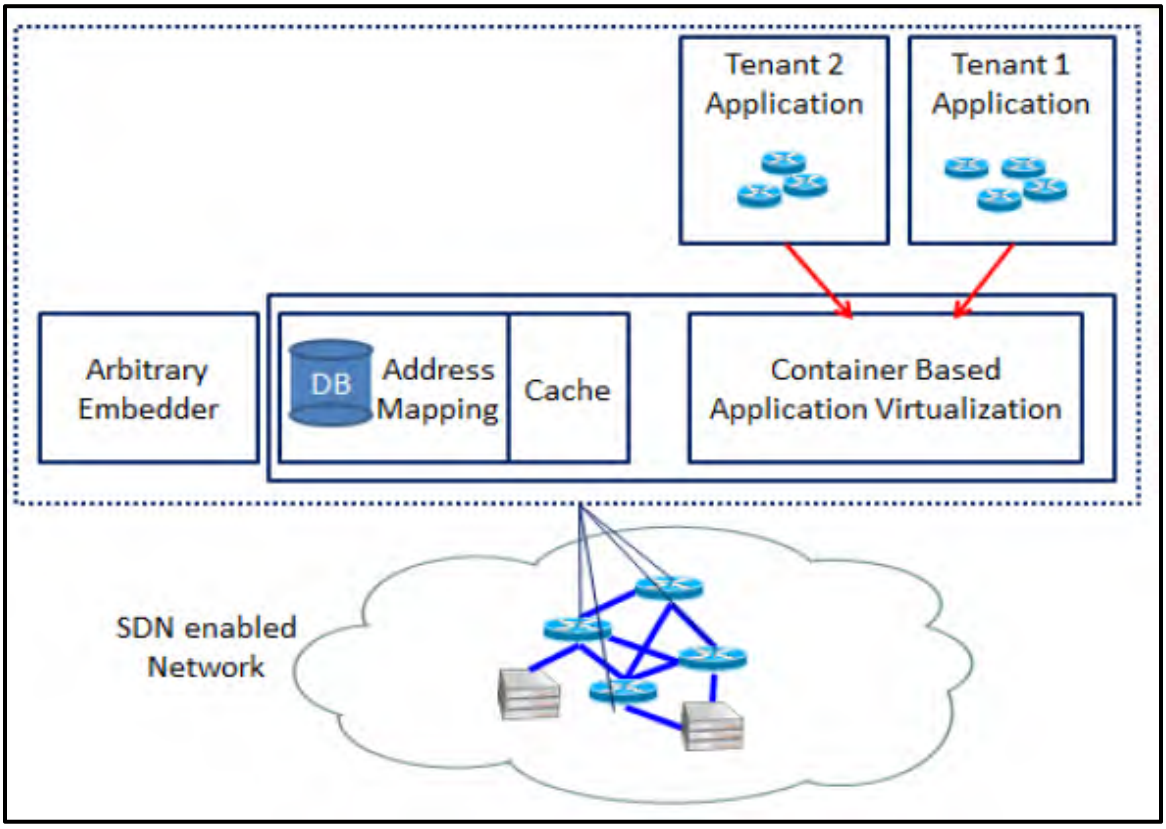


Figure 1.16 architecture Flown. (Drutskoy, D. A. et al., 2012)

1.2.1.2.5 Architecture 5 : Contrôleur maître

L'utilisation d'un contrôleur maître (Pontus Sköldström, 2012) permet à un composant (contrôleur maître) de configurer les instances Openflow en vue de créer des réseaux virtuels. Ce contrôleur maître se charge d'insérer des règles dans la table principale (*master table*) dans le but d'acheminer les paquets vers les tables appartenant à un contrôleur donné qui gère un réseau virtuel.

Les solutions présentées pour la virtualisation offrent la possibilité de bâtir des réseaux virtuels. La virtualisation est un concept qui s'impose dans un environnement multi clients. Dans le cadre des travaux de cette maîtrise, la mise en place de la virtualisation s'inspire du modèle avec une seule instance OpenFlow, de celui de FlowVisor et de l'utilisation d'un contrôleur maître. Une instance Openflow permet une gestion efficace des tables de flux, car cela implique la prise en compte d'un seul processus Openflow. Le modèle de FlowVisor aide à insérer un

mandataire entre les contrôleurs et la couche physique. L'utilisation d'un contrôleur maître permet d'utiliser la flexibilité d'Openflow pour créer des réseaux virtuels à travers les commutateurs qui contiennent les tables de flux.

1.2.2 Isolation

L'isolation permet de protéger la confidentialité des paquets qui circulent dans le plan de données et aussi d'empêcher qu'un paquet se rende au mauvais contrôleur. La suite de la section fait part des recherches qui traitent de la problématique de l'isolation.

1.2.2.1 Flowvisor

Une des premières solutions développées pour l'isolation en SDN est Flowvisor (Sherwood, Rob, et al. 2009). Flowvisor se comporte comme un hyperviseur situé entre le plan de données et le plan de contrôle. Cette solution permet de diviser les ressources du réseau entre différents contrôleurs. Les règles des contrôleurs sont appliquées seulement que sur les ressources qui leur sont allouées. Parmi les ressources divisées, il y a les entrées dans les tables de flux. Il utilise des règles OpenFlow pour créer l'isolation des trafics des contrôleurs. Flowvisor n'inclut pas la gestion au-delà du plan de contrôle. Également une seule table est utilisée pour insérer les règles des contrôleurs.

1.2.2.2 Splendid isolation

Splendid Isolation(Gutz, Stephen. et al., 2012) met en cause la technique de Flowvisor en proposant l'isolation par la programmation. Il propose de se servir des capacités d'Openflow pour diviser le réseau. Il utilise la spécification des *slices* (division des éléments du réseau physique) en Frenetic et les programmes en NetCore pour générer des règles qui vont être installées dans les commutateurs. Ces règles permettent de créer des réseaux virtuels. Mais encore une fois, des faiblesses existent. Il ne donne pas de solutions pour la gestion des messages contrôleur-commutateur ou commutateur-contrôleur. Aucune indication n'est faîte concernant l'utilisation d'une seule ou plusieurs tables par commutateur. Également la solution n'a pas fait l'effet d'évaluation de performance. Les auteurs se limitent à donner une vérification formelle de l'isolation des trafics des réseaux virtuels.

1.2.2.3 OpenVirtex

OpenVirtex(Al-Shabibi, Ali. 2014) se sert des concepts de Flowvisor pour apporter une autre dimension à l'isolation des trafics. Cette solution donne la possibilité à un client de configurer un réseau virtuel tout en utilisant son contrôleur. Chaque contrôleur gère des trafics différents dans des réseaux virtuels. Les informations des réseaux virtuels (IP,port) sont ensuite changées au niveau de l'hyperviseur pour créer des *slices* au niveau du réseau physique. Du réseau physique au réseau virtuel, un démultiplexage est fait.
Mais un problème se dégage de cette solution. OpenVirtex ne supporte pas la gestion des tables multiples. Toutes les règles sont insérées dans une seule table dans les commutateurs. Ceci entraine un nombre important d'entrées dans les tables de flux et un manque de cohésion pour la table hébergeant les entrées de flux.

1.2.2.4 SDNMS

SDNMS (Ahmed, Mohamed Fekih. et al. 2015) propose l'isolation en se servant des tables de flux. Il utilise aussi la hiérarchisation des contrôleurs comme dans le modèle de virtualisation avec contrôleur maître. Il crée des VTS (*virtual tenant slice*) en combinant la flexibilité de SDN et les dimensionnements de réseau offerts par le protocole OVERLAY. Comme avec Flowvisor, une translation des données virtuelles (comme ports) est faite. Pour effectuer une communication d'un VTS à l'autre, cela est régi par des contrôleurs indépendants. Chaque host est lié à un port dédié, une table et/ou groupe de tables. Des tests effectués ont permis de montrer que les trois types de VTS ont les mêmes performances. À l'intérieur des commutateurs plusieurs tables sont utilisées. La table 0 est chargée d'acheminer les paquets vers les sous-tables appartenant à des contrôleurs avec une table par contrôleur. Pour acheminer les paquets vers les tables, SDNMS se sert des informations de la couche 2. Dans l'illustration de la figure 1.17, la table 0 se sert des adresses MAC de destination pour soit acheminer les trames vers la table 1 pour le client 1 et vers la table 2 pour le client 2. La gestion de l'isolation au niveau de la couche 2 entraine un acheminement sans doute vers les bons hosts, mais génère plusieurs règles OpenFlow. Le traitement des flux se fait par adresse MAC et non pas groupe d'adresses IP.

Un autre domaine de recherche de SDN concerne aussi la composition entre les contrôleurs. La composition permet à plusieurs contrôleurs de travailler sur les mêmes trafics appartenant à un même client.

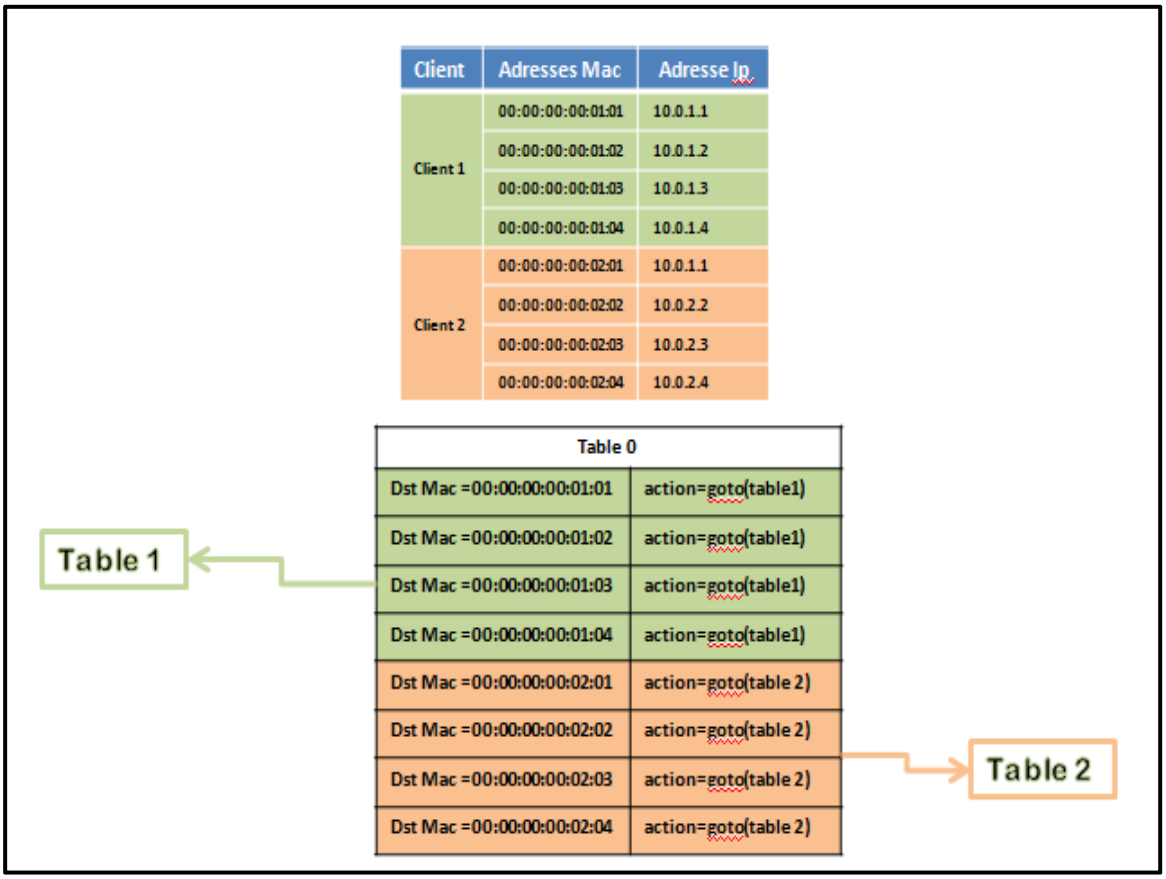


Figure 1.17 Détails du modèle d'isolation de SDNMS

1.2.3 Composition

La composition permet de donner la possibilité à plusieurs environnements de contrôle de travailler sur les mêmes trafics pour un même client. La suite de la section concernant la composition expose les travaux qui proposent des solutions.

1.2.3.1 Flowbricks

Flowbricks (Dixit, A. et al 2014) propose une couche située entre des contrôleurs et le plan de données. À l'aide de spécifications de composition fournies par un opérateur, un pipeline de tables de flux est généré dans les commutateurs en plus de l'utilisation de l'algorithme de qui lie un contrôleur à un ensemble de tables. Chaque contrôleur est lié à plusieurs tables de flux incluant les tables de transition. Les tables de transition permettent de passer d'un pipeline d'un contrôleur au pipeline d'un autre contrôleur. Une composition parallèle ou séquentielle est possible. La composition se fait sur les mêmes flux. Dans cette recherche, les auteurs proposent l'utilisation des tables de transition (Voir figure 1.18). La table 0 dirige les flux vers les pipelines des contrôleurs en fonction de la politique de traitement séquentielle ou parallèle définie. L'existence de la table 0 est inutile, car elle rallonge le traitement de flux dans le pipeline complet. Le pipeline complet est le processus de traitement complet des flux de la table 0 jusqu'à la dernière table qui fait le traitement final. La première table du pipeline du premier contrôleur peut bien prendre en charge le flux dès son entrée dans l'espace du plan de données qui gère la composition. Les tables de transition des contrôleurs acheminent les flux vers le pipeline du prochain contrôleur. Ces tables de transition rallongent aussi le traitement des flux. Une économie des tables à utiliser doit être prise en compte, car le nombre de tables à utiliser en Openflow 1.0 est limité à 256. En lien des tables aux tables de transition, il y a augmentation de couplage entre ces dernières.
Contrairement à Flowbricks, la solution étudiée dans le cadre de cette étude élimine l'utilisation inutile des tables dans les commutateurs.

1.2.3.2 Covisor

Covisor (Jin, Xin. et al. 2015) permet une composition parallèle et séquentielle des contrôleurs. Situé entre les contrôleurs et le plan de données, Covisor reçoit les règles des contrôleurs. Il fait une première transformation des règles venant des contrôleurs pour générer une politique adaptée au réseau virtuel géré par les contrôleurs composés. Par la suite une deuxième phase transforme la politique pour le réseau virtuel en règle pour le réseau physique qui est le plan de données (voir figure 1.19). Toutes les règles venant des contrôleurs en composition sont insérées dans une seule table dans les commutateurs dans le plan de données. Tout comme

OpenVirtex, Covisor ne supporte pas l'utilisation de plusieurs tables. Toutes les règles sont insérées dans une seule table dans les commutateurs. Coretic, la solution développée dans le cadre de cette étude de maîtrise permet d'attribuer une table à chaque contrôleur. La cohésion des tables est ainsi renforcée.

La section 1.2 a permis de lire des études en lien avec les solutions d'*hypervision* pour l'isolation et la composition entre les contrôleurs. Ces solutions sont regroupées dans le tableau 1.4.

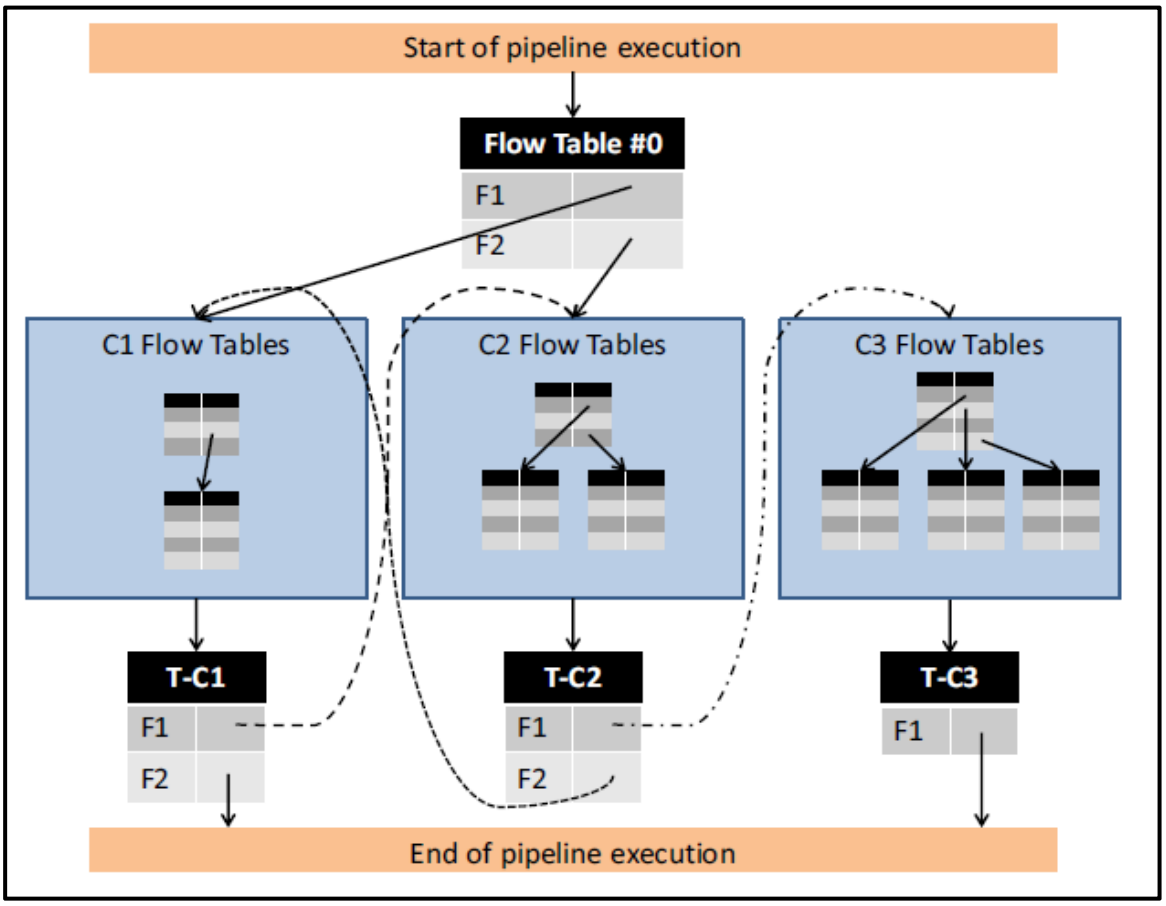


Figure 1.18 Détails du modèle de composition de FlowBricks (Dixit, A. 2014)

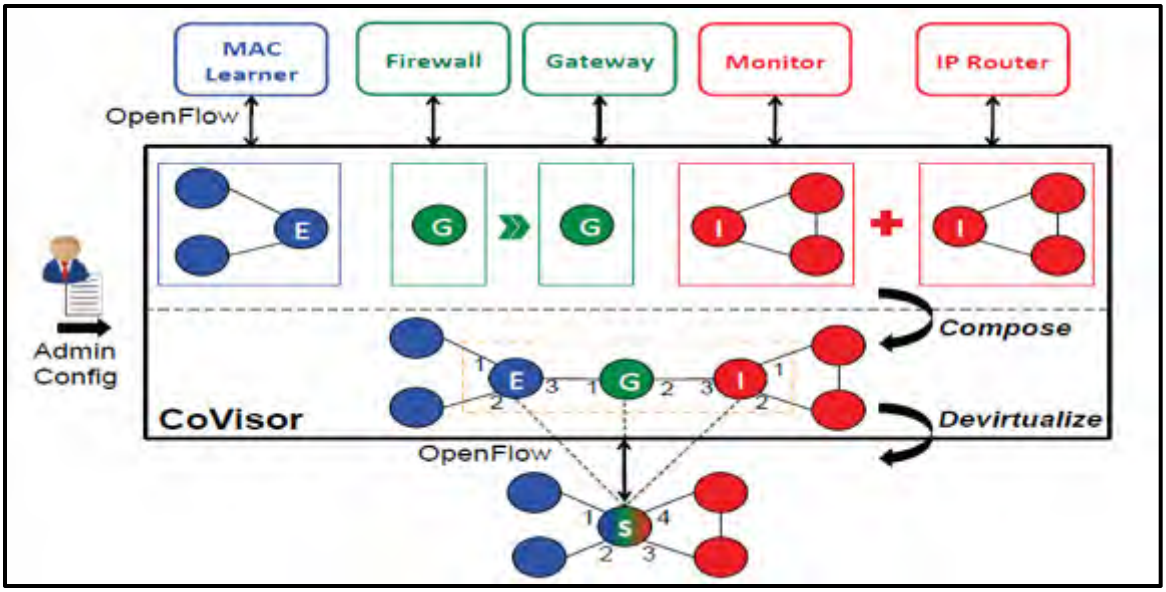


Figure 1.19 Aperçue du fonctionnement de Covisor (Jin, Xin. et al. 2015)

Tableau 1.4 Résumé des solutions d'*hypervision* (isolation et composition)

solutions	Isolation	composition	Multi tables	Modèle(s) virtualisation	Architecture	Informations Niveau 2 utilisées	Informations Niveau 3 utilisées
FlowVisor	Oui	Non	non	Modèle 1	Architecture 1	Oui	Oui
Splendid Isolation	Oui	Non	Non	Modèle 2	Architecture 5	Oui	Oui
OpenVirtex	Oui	Non	Non	Modèle 2 et Modèle 4	Architecture 3	Oui	Oui
SDNMS	Oui	Non	Oui	Modèle 6	Architecture 5	Oui	Non
FlowBricks	Non	Oui	Oui	Modèle 2 et modèle 6	Architecture 5	Oui	Non
Covisor	Non	Oui	Non	Modèle 2 et modèle 4	Architecture 3	Oui	oui

L'architecture SDN, en plus du plan de données et de contrôle, permet d'obtenir une autre couche appelée plan de gestion. Ce dernier contient les plateformes de programmation de haut niveau.

1.3 Plateformes de programmation de haut niveau

Les plateformes de programmation de haut niveau en SDN sont des solutions qui permettent de compiler en Openflow en utilisant des API des contrôleurs des politiques écrites dans un langage plus évolué. Elles permettent de traiter les Packetin à un niveau d'abstraction élevé à la place du plan de contrôle. Les plateformes réduisent les risques d'insertion d'erreurs dans l'écriture des politiques. Celles qui sont étudiées sont spécialisées dans la réalisation de plusieurs politiques en modules. Créer des politiques en utilisant des modules permet de réutiliser une application. La maintenance des modules est ainsi rendue facile. La suite de cette section expose des solutions pour ces plateformes.

1.3.1 Frenetic

La première plateforme ayant eu pour objectif la réalisation d'une plateforme de programmation de haut niveau est Frenetic (Foster, Nate, et al. 2011). Il fait abstraction des détails d'implémentation de bas niveau. Il se sert du langage de requêtes réseau de la librairie "*network policy management*" pour créer des modules. Ces modules peuvent être composés de façon séquentielle. Il s'exécute en mode réactif seulement.

1.3.2 HFT

HFT (Ferguson, Andrew D. et al. 2012) travaillant dans la même veine que Frenetic, propose une plateforme de programmation à haut niveau basée sur la hiérarchisation des politiques en arbre. Avec cet outil, la résolution des conflits entre deux modules se fait au niveau des feuilles des arbres.

1.3.3 Pyretic

Pyretic (Reich, Joshua. et al. 2013) a suivi les objectifs de Frenetic en proposant plusieurs modes d'exécution des applications. Il y a également la composition des modules en parallèle

beaucoup plus élaborée. L'écriture des politiques prend moins de lignes de code. Pour faire abstraction de la couche de contrôle, Pyretic se sert d'un environnement de génération de règle appelé *Runtime*. Ensuite il se sert d'un client Openflow pour envoyer des règles au contrôleur et également recevoir des Packetin. La plateforme se subdivise en quatre grandes parties : *Of_Client, Backend, Runtime*, espace des modules en Python (Figure 1.20)

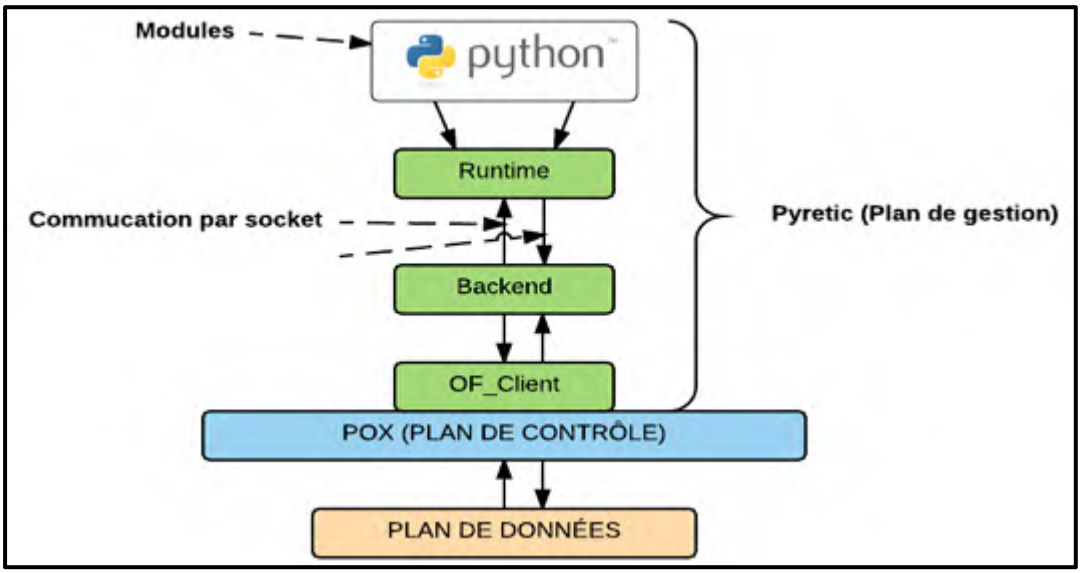


Figure 1.20 représentation des principales couches de Pyretic

La couche *Of_Client* utilise les API du contrôleur pour envoyer des règles au plan de données. Le *Backend* est chargé de faire le lien entre le *Runtime* et *Of_Client*. *Runtime* est la couche qui permet de générer des règles Openflow sous forme de dictionnaire à partir des modules qui se situent au niveau de la couche au-dessus.

La section 1.3 qui vient d'être abordée a permis de prendre connaissance des solutions pour la programmation des politiques à partir du plan de gestion. Ces solutions sont résumées dans le tableau 1.5.

Tableau 1.5 Résumé des solutions de plateformes de programmation de haut niveau

Solution de plateforme	**Permet de faire l'*hypervision***	**Composition de modules**
Frenetic	Non	Oui

HFT	Non	Oui
Pyretic	Non	Oui

Vues les solutions d'isolation, composition et des plateformes de programmation de haut niveau ci-dessus, des problèmes se dégagent. Ces problèmes sont rappelés et détaillés dans la section qui suit.

1.4 Discussion des solutions

Cette section permet de lever les problèmes en lien avec les solutions vues dans les sections présentées précédemment. Elle donne également un récapitulatif de ces solutions à travers un tableau.

1.4.1 Problème en lien avec l'isolation

Pour les solutions d'isolation, plusieurs problèmes ont été identifiés. Ces problèmes sont présentés ci-dessous :

- Utilisation des informations de la couche 2 de réseau

L'utilisation des informations de la couche 2 comme l'adresse MAC génère plusieurs règles, car toutes les machines dans le réseau doivent être prises en compte de façon individuelle. En générant plusieurs règles, des problèmes de performances de traitement de flux peuvent surgir, car il faut passer à travers plusieurs entrées dans les tables de flux pour gérer le trafic. Ce problème est identifié dans SDNMS et OpenVirtex(Al-Shabibi, Ali. 2014).

- Manque de cohésion

Le manque de cohésion est aussi observé dans SDNMS et OpenVirtex. SDNMS laisse les tables des clients supprimer les paquets qui appartiennent à un autre client. Ce qui donne deux rôles aux tables des clients à savoir supprimer des paquets et aussi acheminer des paquets légitimes vers les bons ports de sortie. Le manque de cohésion est aussi noté avec OpenVirtex, car toutes les règles sont une seule table de flux.

1.4.2 Problème en lien avec la composition

Plusieurs problèmes ont été identifiés pour la composition. Ces derniers sont :

- Manque de cohésion

La solution Covisor utilise une seule table pour écrire les règles. Ceci permet de confier plusieurs responsabilités à une seule table de flux. Ce qui augmente très considérablement la cohésion.

- Couplage élevé entre les tables de flux

La solution de composition de contrôleurs avec Flowbricks (Dixit, A. 2014) utilise des tables de transition pour traiter des flux. Ces tables de transition augmentent le couplage des tables de flux, car les tables des contrôleurs sont liées à ces tables de transition. Il faut aussi noter qu'il est important de faire l'économie de tables en OpenFlow 1.0, car le nombre de tables qu'il est possible d'utiliser est 256.

1.4.3 Problème en lien avec les plateformes de programmation de haut niveau

Les plateformes de programmation de haut niveau ne sont pas conçues pour être utilisées en parallèle. Par exemple, Pyretic est mono utilisateur. C'est-à-dire que plusieurs personnes ne peuvent utiliser Pyretic en même temps pour un même contrôleur. Les hyperviseurs conçus pour les contrôleurs ne sont pas adaptés aux plateformes de programmation de haut niveau. Étant donné que le rôle de contrôle des flux est confié à la plateforme de programmation de haut niveau, une solution aidant à créer de l'*hypervision* entre ces plateformes devient pertinente. Ces plateformes de haut niveau dans ce contexte sont appelées des contrôleurs de haut niveau dans ce travail de maîtrise.

1.5 Conclusion

Le chapitre 1 a permis de présenter SDN, le protocole Openflow, les solutions d'*hypervision* et une section de discussion pour justifier les raisons de ce travail de maîtrise. Les différentes couches de SDN ont été présentées. Le plan de données se sert des tables de flux pour gérer les paquets ou les trames. Le plan de contrôle à travers des messages indique au plan de données la façon de gérer les flux. Le plan de gestion permet de créer une abstraction des API du plan de contrôle. Ce plan permet aussi de faire une gestion des requêtes du plan de données à un niveau d'abstraction plus élevé. Le protocole Openflow est le standard utilisé pour permettre au plan de données et le plan de contrôle de communiquer.

À la suite de la présentation de SDN et du protocole Openflow, plusieurs travaux en lien avec l'*hypervision* en SDN ont été présentés. D'abord des modèles et des architectures d'*hypervision* ont été présentés. Ensuite des travaux en lien avec les trois axes de recherches de cette maîtrise ont été présentés. Pour l'isolation des flux des plans de contrôle, il y a utilisation des informations de la couche 2 du modèle OSI et aussi l'utilisation d'une seule table pour écrire toutes les règles. Au niveau de la composition, il y a aussi utilisation d'une seule table avec Covisor et une mauvaise utilisation de plusieurs tables avec Flowbricks. Pour les plateformes de programmation de haut niveau, une *hypervision* est impossible.

La dernière section du chapitre 1 a permis de rappeler les problèmes liés avec les solutions précédentes d'isolation, de composition et des plateformes de programmation de haut niveau. Le chapitre 2 suivant permettra de donner plus de détails sur les moyens utilisés pour offrir une solution pour atteindre les objectifs.

CHAPITRE 2

FONCTIONNEMENT DE LA SOLUTION CORETIC

Le chapitre 1 a permis de montrer plusieurs problèmes rencontrés dans les études antérieures pour l'isolation, la composition et les plateformes de programmation de haut niveau. Pour atteindre les objectifs, des hypothèses furent émises. Le chapitre 2 explique en détails la solution Coretic permettant d'atteindre les objectifs. Elle explique comment les segments d'adresse IP et les tables de flux sont utilisés pour l'isolation et la composition. Dans ce chapitre également il sera expliqué le fonctionnement de l'hyperviseur des contrôleurs de haut niveau. En vue de donner une bonne lecture de la solution préconisée dans le cadre de cette étude, le chapitre 2 a été divisé en quatre parties. La première partie 2.1 donnera la signification des notations qui seront utilisées tout au long de l'explication de la solution. La partie 2.2 traite de la solution d'isolation des réseaux virtuels dirigés par des contrôleurs différents en mode proactif. La solution d'isolation est appelée CoreticIsol. La partie 2.3 du chapitre 2 expliquera aussi la solution proposée pour permettre à plusieurs contrôleurs, en composition de traiter des flux. Pour la composition, la solution est appelée CoreticComp. Le chapitre 2 se terminera par l'explication de la solution proposée pour faire de l'*hypervision* des contrôleurs de haut niveau dans la partie 2.4. Cette troisième solution est appelée CoreticHip. Le chapitre 2 se terminera par une conclusion qui va résumer les solutions des trois axes de travail.

2.1 Notations

Avant de continuer plus loin, cette sous-section décrit la signification de certaines formules.

- $\boldsymbol{P}$: Il s'agit d'une politique donnée pour gérer des trafics dans un commutateur.
- $\boldsymbol{M}$: Il s'agit d'un module qui définit une politique donnée.
- $\boldsymbol{P} = \boldsymbol{M}+$: Une politique est composée d'un ou plusieurs modules.
- $\boldsymbol{R}$: Une règle qui est représentée par une entrée dans la table de flux.
- $\boldsymbol{M} = \boldsymbol{R}+$: Un module est constitué d'une ou plusieurs règles.

- ***S*** : Pour désigner un commutateur.
- ***R = (Rm, Rp, Ra)*** : Une règle est constituée de trois parties qui sont le Match(*Rm*), la priorité (*Rp*) et l'action(*Ra*). Il est à noter que les actions peuvent être plusieurs.
- **PktIn *:*** Il s'agit d'un paquet qui va de commutateurs au contrôleur.
- **PktOut *:*** Il s'agit d'un paquet qui va du contrôleur aux commutateurs.

2.2 Isolation

La solution d'isolation, CoreticIsol, développée dans le cadre de cette maîtrise permet de séparer les trafics dirigés par des contrôleurs différents. Cette solution s'adapte pour les architectures avec les contrôleurs de bas niveau et les contrôleurs à haut niveau. Un contrôleur de bas niveau contrairement au contrôleur de haut niveau se situe dans le plan de contrôle. Avant d'expliquer la solution d'écriture des règles dans les commutateurs, un rappel des architectures SDN applicables sera fait.

2.2.1 Les architectures SDN applicables

La solution CoreticIsol est applicable avec l'architecture SDN avec isolation des trafics des contrôleurs de bas niveau et aussi avec l'architecture avec contrôleurs de haut niveau.

2.2.1.1 Architecture avec contrôleurs de bas niveau

Les solutions d'*hypervision* développées au niveau de CoreticIsol sont des modèles d'écriture de règles insérées dans les commutateurs. Les contrôleurs de façon proactive écrivent leurs règles dans la table de flux assignée. Elle est compatible avec une architecture SDN où il y a une isolation des trafics des contrôleurs de bas niveau (voir figure 2.1). Dans cette architecture, l'hyperviseur qui est en même temps le contrôleur maître configure la table 0 dans les commutateurs afin que les paquets soient acheminés vers les tables appartenant aux différents clients. Quant aux contrôleurs, ils écrivent leurs règles dans les tables fournies en passant par l'hyperviseur.

2.2.1.2 Architecture avec contrôleurs de haut niveau

L'architecture avec contrôleurs de haut niveau implique des contrôleurs situés dans le plan de gestion (voir figure 2.2). Ces contrôleurs sont des modules s'exécutant à l'aide des plateformes de programmation de haut niveau. Chaque contrôleur de haut niveau gère un réseau virtuel avec des ressources physiques assignées. Ces modules à l'aide d'un client OpenFlow envoient leurs règles à l'hyperviseur de haut niveau. L'hyperviseur modifie les messages des contrôleurs et spécifie les numéros des tables d'écriture. Cette architecture sera vue en détail dans la section 2.4.

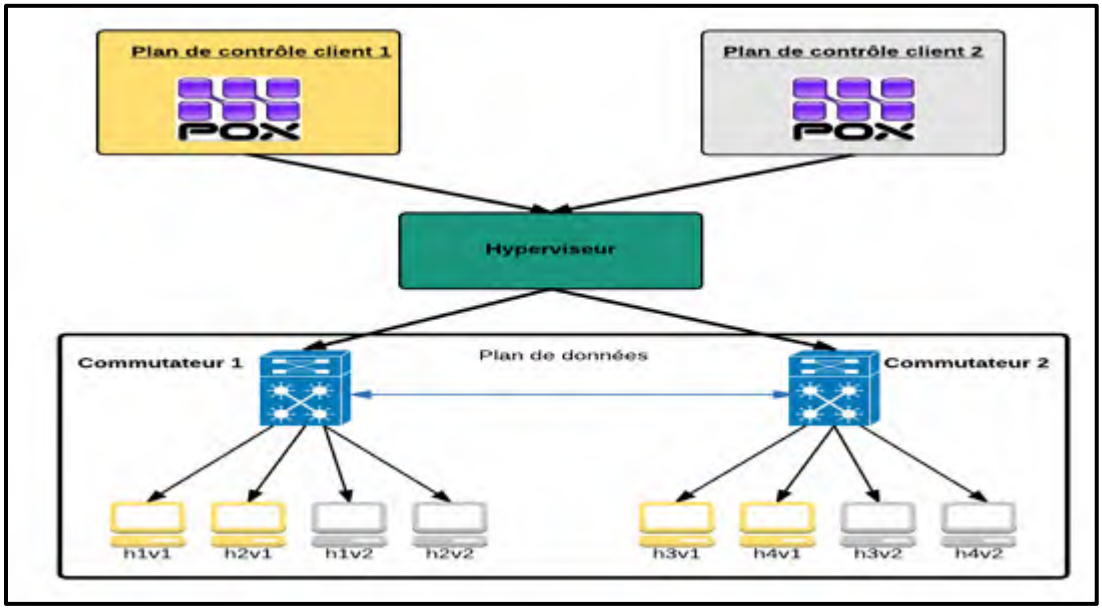


Figure 2.1 Architecture SDN avec *hypervision* des contrôleurs de bas niveau

2.2.2 Solution d'écriture pour l'isolation

La solution d'isolation des trafics des contrôleurs dans le cadre de cette maîtrise utilise plusieurs tables comme proposée par d'autres travaux (Pontus Sköldström (2012) et (Ahmed, Mohamed Fekih. 2015)). L'utilisation de plusieurs tables comme illustrée dans la figure 2.3, permet de faire passer un trafic d'une table maître à d'autres tables gérées par des contrôleurs.

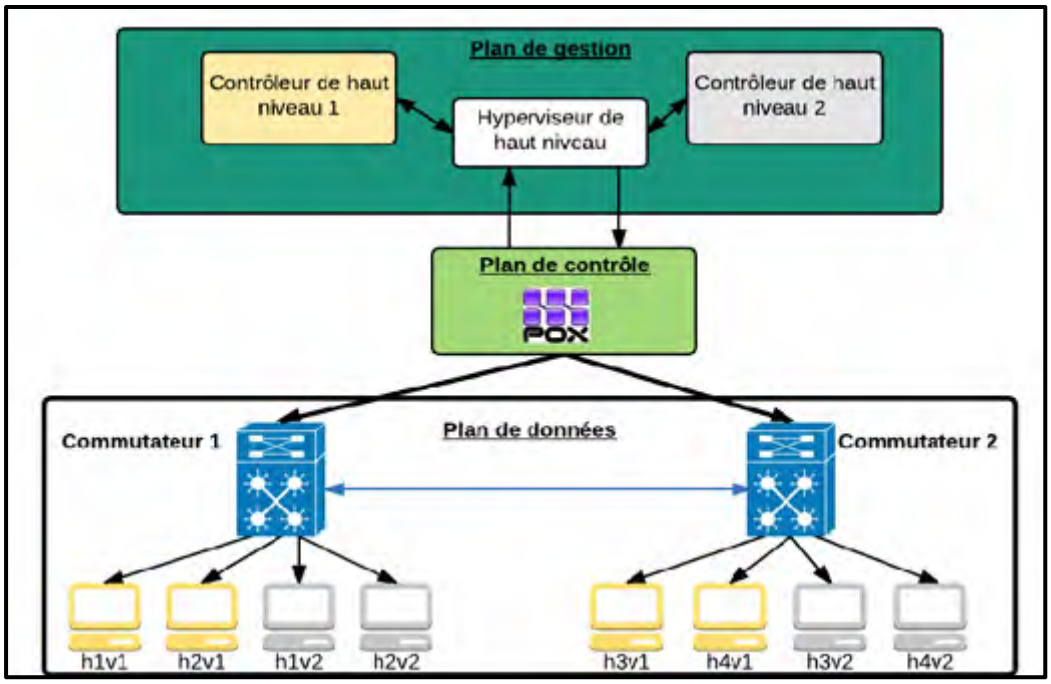


Figure 2.2 Architecture avec hyperviseur de contrôleur de haut niveau

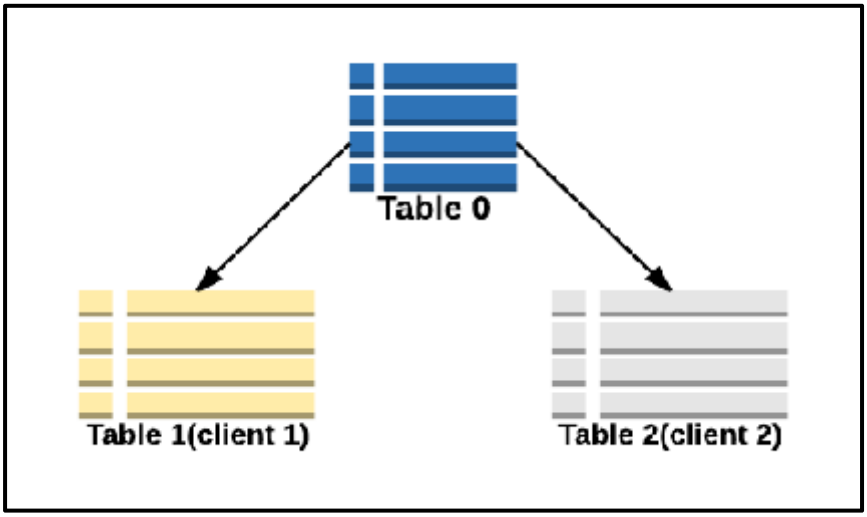


Figure 2.3 Illustration de la hiérarchie des tables
dans les commutateurs

De façon plus précise, la première table appartient au contrôleur ou à l'hyperviseur qui fait la configuration. Cela se rapproche du modèle 6 pour l'*hypervision*. Dans la première table, des règles sont insérées afin de diriger les trafics vers les autres tables qui appartiennent à des contrôleurs différents. Chaque contrôleur écrit sa politique dans sa table qui lui est assignée.

Contrairement aux autres solutions à savoir SDNMS et OpenVirtex, CoreticIsol utilise les informations de la couche 3 du modèle OSI. L'utilisation des informations de la couche 3 du modèle OSI permet de regrouper les adresses IP des machines virtuelles par segment. Pour pouvoir approfondir l'explication de la solution, le tableau 2.1 sera utilisé. Dans ce tableau, deux clients sont illustrés avec leurs adresses IP, MAC, noms de *hosts* et des ports de connexion aux commutateurs.

Tableau 2.1 Tableau de répartition des ressources physiques pour isolation

Client	Nom des machines virtuelles	Les adresses IP(de façon respective)	Les adresses MAC	Ports des hosts aux commutateurs
Client 1	h1v1	10.0.1.1	00 :00 :00 :00 :01 :01	1 (S1)
	h2v1	10.0.1.2	00 :00 :00 :00 :01 :02	2 (S1)
	h3v1	10.0.1.3	00 :00 :00 :00 :01 :03	1 (S2)
	h4v1	10.0.1.4	00 :00 :00 :00 :01 :04	2 (S2)
Client 2	h1v2	10.0.2.1	00 :00 :00 :00 :02 :01	3 (S1)
	h2v2	10.0.2.2	00 :00 :00 :00 :02 :02	4 (S1)
	h3v2	10.0.2.3	00 :00 :00 :00 :02 :03	3 (S2)
	h4v2	10.0.2.4	00 :00 :00 :00 :02 :04	4 (S2)

La table 0 contiendra les règles pour soit acheminer les paquets vers la table 1 ou vers la table 2 en fonction des adresses IP. Les règles ci-dessous seront écrites dans la table 0 :

- Pour client 1 :***R1* = [*m :(srcip=10.0.1.0/24, dstip=10.0.1.0/24); p :1000; a :goto(table1)***]**.
- Pour client 2: ***R2* = [*m :(srcip=10.0.2.0/24, dstip=10.0.2.0/24);p :1000; a :goto(table2)***]**.

Pour la règle *R1*, tous les paquets dont l'entête respecte les conditions selon lesquelles l'adresse source et l'adresse destination sont dans le réseau "10.0.1.0/24", seront acheminés dans la table 1. Pour la règle R2, c'est la même situation avec cette fois-ci l'adresse réseau sera "10.0.2.0/24". Et la table 2 recevra les paquets cette fois-ci. Les autres paquets qui ne respectent les règles R1 et R2 seront simplement supprimés par une règle par défaut. Cela pourrait être par exemple la machine h1v1 du client 1 qui voudrait envoyer un message à la machine h1v2 du client 2. Le processus d'isolation est illustré dans la figure 2.4. Dans cette

figure, le nombre d'entrées réel n'est pas représenté. Les entrées sont représentées à un niveau d'abstraction élevé. Pour un nombre de clients N donné, avec CoreticIsol les besoins en ressource sont les suivants :

- Total de tables utilisées (T_N) = N +1
- Pour la table 0, ΣR_i (somme des règles) = (Nx2) + 1 où i est le nombre de clients ou nombre de réseaux virtuels à isoler.

N a été multiplié par deux, car deux types de messages sont traités dans les tables de flux. Ce sont les trames ARP et les paquets IP.

Le nombre de machines appartenant à un client n'influence pas la quantité de règles dans la table 0. Ce qui n'est pas le cas avec SDNMS dont le nombre de règles dans la table de flux augmente en fonction du nombre de machines d'un client donné. Avec deux réseaux virtuels utilisant 100 machines chacune, le nombre d'adresses MAC ou de port de communication des commutateurs devient difficile à gérer. Cette gestion devient difficile, car SDNMS utilise les informations de la couche 2 du modèle OSI pour faire l'isolation. Avec Openvirtex, le nombre d'entrées dans la table de flux augmente aussi en fonction du nombre de clients.

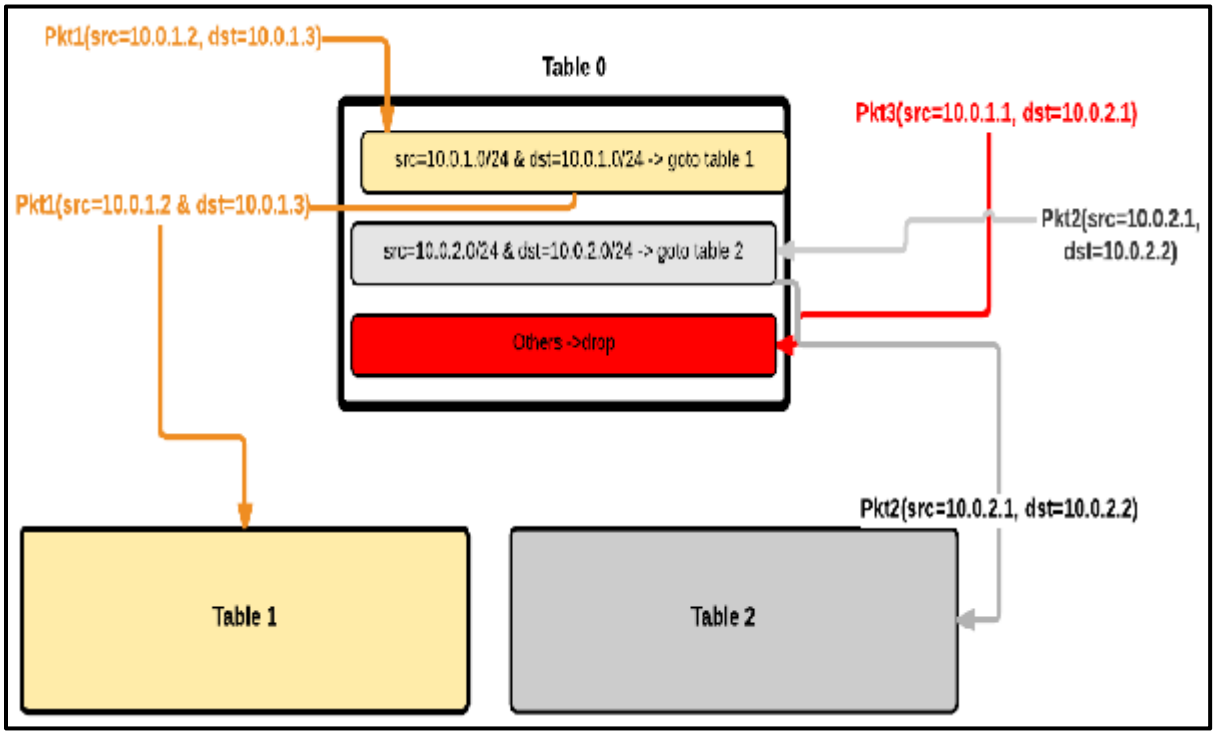


Figure 2.4 Illustration de la solution d'isolation de CoreticIsol

La partie 2.2 précédente a permis d'expliquer comment Coretic à travers sa composante CoreticIsol utilise plusieurs tables et les informations de la couche 3 pour l'isolation. La section 2.3 qui suit aborde le sujet en lien avec la composition.

2.3 Composition

La solution de composition de Coretic, appelé CoreticComp, permet à plusieurs modules provenant de contrôleurs différents de travailler en même temps sur les mêmes paquets. Pour expliquer les détails de la solution, le contexte de composition sera expliqué. Ensuite suivront les explications sur la composition séquentielle et parallèle.

Pour le reste de la lecture sur la composition, veuillez considérer les hypothèses suivantes :

- Soit un client T1 ayant deux contrôleurs C1 et C2
- Les contrôleurs C1 et C2 produisent respectivement les règles R1(Rm1,Rp1,Ra1) et R2(Rm2,Rp2,Ra2).
- Soit le plan de données disponible ayant quatre machines (voir tableau 2.2)

Tableau 2.2 Tableau fournissant les informations des machines pour CoreticComp

Client	Nom des machines virtuelles	Les adresses IP(de façon respective)	Les adresses MAC	Ports des hosts au commutateur
Client 1	h1v1	10.0.1.1	00 :00 :00 :00 :01 :01	1 (S1)
	h2v1	10.0.1.2	00 :00 :00 :00 :01 :02	2 (S1)
	h3v1	10.0.1.3	00 :00 :00 :00 :01 :03	3 (S1)
	h4v1	10.0.1.4	00 :00 :00 :00 :01 :04	4 (S1)

2.3.1 Contexte de composition

Pour la composition des règles provenant de plusieurs contrôleurs, deux cas peuvent arriver. Il y a le cas où les règles d'un contrôleur précédent modifient l'entête du paquet (cas 1) et l'autre cas où les règles du contrôleur précédent ne modifient pas l'entête du paquet (cas 2). Dans le cadre de cette maîtrise, le cas 2 est pris en compte. Par exemple, dans la notation C1 >> C2 se traduisant par l'opération de composition OP=R1 >> R2. Nous avons R1(Rm1, Rp1, Ra1) et R2(Rm2, Rp2, Ra2). À l'arrivée d'un paquet, si le *Match* Rm1 est respecté, Ra1 ne devrait pas

modifier les conditions d'entête de paquet dans Rm1. Les contrôleurs dans ce contexte sont indépendants. C'est-à-dire qu'un contrôleur pose ses actions comme s'il était seul.

2.3.2 Composition séquentielle

La solution de composition proposée ici est basée sur l'utilisation de plusieurs tables. Pour mieux expliquer la solution, un cas d'utilisation simple (voir les tirets ci-dessous) est utilisé.

- R1 fournit des règles de coupe-feu pour supprimer les paquets avec adresse IP source ou destination étant impaire.
- R2 fournit les règles de commutation pour les autres machines dont les adresses IP sont paires.
- La composition séquentielle ici donne C1>>C2 qui équivaut à Σr1>>Σr2. L'ensemble des règles de R1 s'appliquent avant celles de R2.

Dans le tableau 2.3, les règles pour C1 et C2 sont représentées. Ces règles sont basées sur le tableau 2.2 qui contient les informations de quatre machines.

Tableau 2.3 Liste des règles pour C1 et C2 pour la composition séquentielle

Contrôleurs	Règles	M(entêtes de correspondance)	Priorités	actions	Tables
C1	R1c1	srcIp=10.0.1.1 ou dstIp=10.0.1.1	1000	Drop	0
	R2c1	srcIp=10.0.1.3 ou dstIp=10.0.1.3	1000	Drop	0
	R3c1	Any	10	Goto table 1	0
C2	R1c2	dstIp=10.0.1.2	1000	outPort 2	1
	R2c2	dstIp=10.0.1.4	1000	outPort 4	1

La solution proposée dans ce travail de maîtrise dans le cadre de la composition séquentielle écrit les règles de C1 avec des priorités élevées. Ceci va obliger la table concernée de traiter les règles de C1 en premier. La table de numéro inférieur est confiée au contrôleur précédent dans la composition, car les paquets ne peuvent aller d'une table de numéro supérieur à une table de numéro inférieur. Le contrôleur C1 fournit les règles de coupe-feu qui sont écrites dans la table 0. Les paquets ayant des adresses IP impaires sont supprimés. Ces règles de coupe-feu doivent être d'une plus grande priorité, car elles sont utilisées en premier pour évaluer les

paquets. Dans cette évaluation, l'action *Drop* permet de supprimer les paquets non conformes. Puisque l'analyse des règles se fait de façon séquentielle, les paquets dont les adresses IP sont paires arrivent au niveau de la règle R3c1. Dans la règle R3c1, l'action correspondante achemine les paquets conformes à la table suivante avec l'action "Goto table1". Pour encore mieux comprendre le processus de traitement des paquets dans la solution de composition séquentielle, la figure 2.5 est fournie. Dans cette figure, deux tables sont utilisées, car il y a composition séquentielle entre deux contrôleurs. De façon générale, lorsqu'il y a N contrôleurs, le nombre de tables à utiliser sera N. Les règles du contrôleur N_i dans la table Ti doivent être d'une priorité élevée afin que les actions puissent s'exécuter. La règle permettant aux paquets légitimes de se rendre à la table T_{i+1} est d'une priorité faible. Ceci va permettre à la règle de se retrouver au bas de la table T_i.

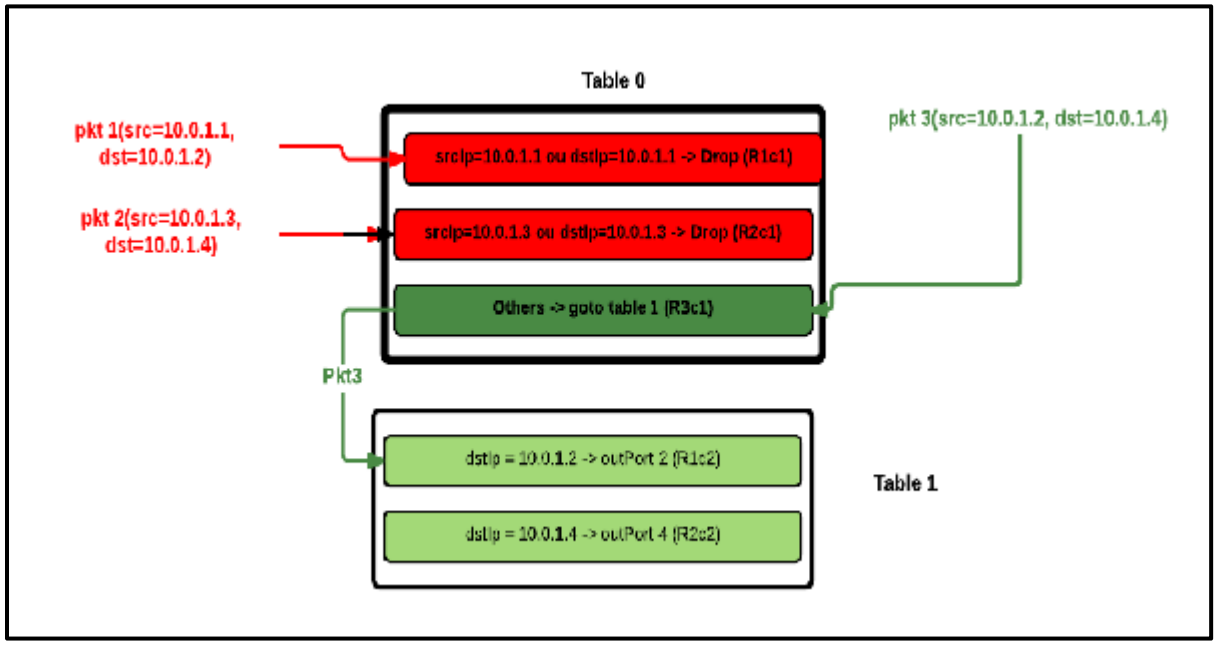


Figure 2.5 Méthode de traitement de paquets pour la composition séquentielle

La section 2.3.2 explique la solution pour la composition séquentielle. La suivante explique la solution proposée pour la composition parallèle.

2.3.3 Composition parallèle

La solution pour la composition parallèle est semblable à celle de la composition séquentielle. Mais la composition parallèle n'empêche pas les actions de C2 de s'exécuter, quelles que soient celles de C1. Avant de continuer, les points ci-dessous expliquent le cas d'utilisation utilisé pour expliquer la solution.

- R1 fournit des règles pour visualiser les paquets avec adresse IP source ou destination étant impaire.
- R2 fournit les règles de commutation pour les autres machines dont les adresses IP sont paires.
- La composition parallèle ici donne **C1** || **C2** qui équivaut à **Σr1** || **Σr2**. L'ensemble des règles de R1 s'appliquent en parallèle avec celles de R2.

Dans le tableau 2.4, les règles pour C1 et C2 sont représentées. Ces règles sont basées sur le tableau 2.2 qui contient les informations de quatre machines.

Tableau 2.4 Liste des règles pour C1 et C2 pour la composition parallèle

Contrôleurs	Règles	M(entêtes de correspondance)	Priorités	actions	Tables
C1	R1c1	srcIp=10.0.1.1 ou dstIp=10.0.1.1	1000	Afficher Pkt, goto table 1	0
	R2c1	srcIp=10.0.1.3 ou dstIp=10.0.1.3	1000	Afficher Pkt, goto table 1	0
	R3c1	Any	10	Goto table 1	0
C2	R1c2	dstIp=10.0.1.2	1000	outPort 2	1
	R2c2	dstIp=10.0.1.4	1000	outPort 4	1
	R3c2	dstIp=10.0.1.1	1000	outPort 1	1
	R4c2	dstIp=10.0.1.3	1000	outPort 3	1

La solution pour la composition parallèle est semblable à celle de la composition séquentielle. Le contrôleur C1 fournit des règles pour afficher les paquets ayant une adresse IP impaire en entête. Les actions pour afficher les paquets concernés pour le contrôleur C1 doivent aussi inclure la possibilité au contrôleur C2 d'appliquer ses règles. C'est pour cela qu'une deuxième action est ajoutée à l'action d'affichage des paquets. Des priorités élevées doivent être appliquées aux règles du contrôleur C1 pour l'affichage des paquets concernés. Les paquets

ayant une adresse IP paire seront traités par la règle R3c1 d'une priorité plus faible. Cette dernière règle ne fait pas d'affichage. Elle ne fait qu'acheminer les paquets à la table 1 du contrôleur C2. Encore une fois, l'ordre des numéros des tables doit être respecté. C'est-à-dire que le numéro de la table du contrôleur C1 doit être inférieur à celui de la table du contrôleur C2. La figure 2.6 montre le processus de traitement des paquets dans les tables. Dans les tables bleues, le mot "Affi." signifie que le paquet va être affiché. Tout comme dans la composition séquentielle, le nombre de tables à utiliser est identique au nombre de contrôleurs en composition. Aussi les règles d'un contrôleur C_i ont une priorité élevée sur la règle qui envoie les paquets non visualisés à la table du contrôleur C_{i+1}.

La section 2.2 et 2.3 ont expliqué respectivement les solutions proposées pour l'isolation des paquets et pour la composition des règles. La dernière section 2.4 qui suit explique la solution proposée pour faire de l'*hypervision* pour les contrôleurs de haut niveau.

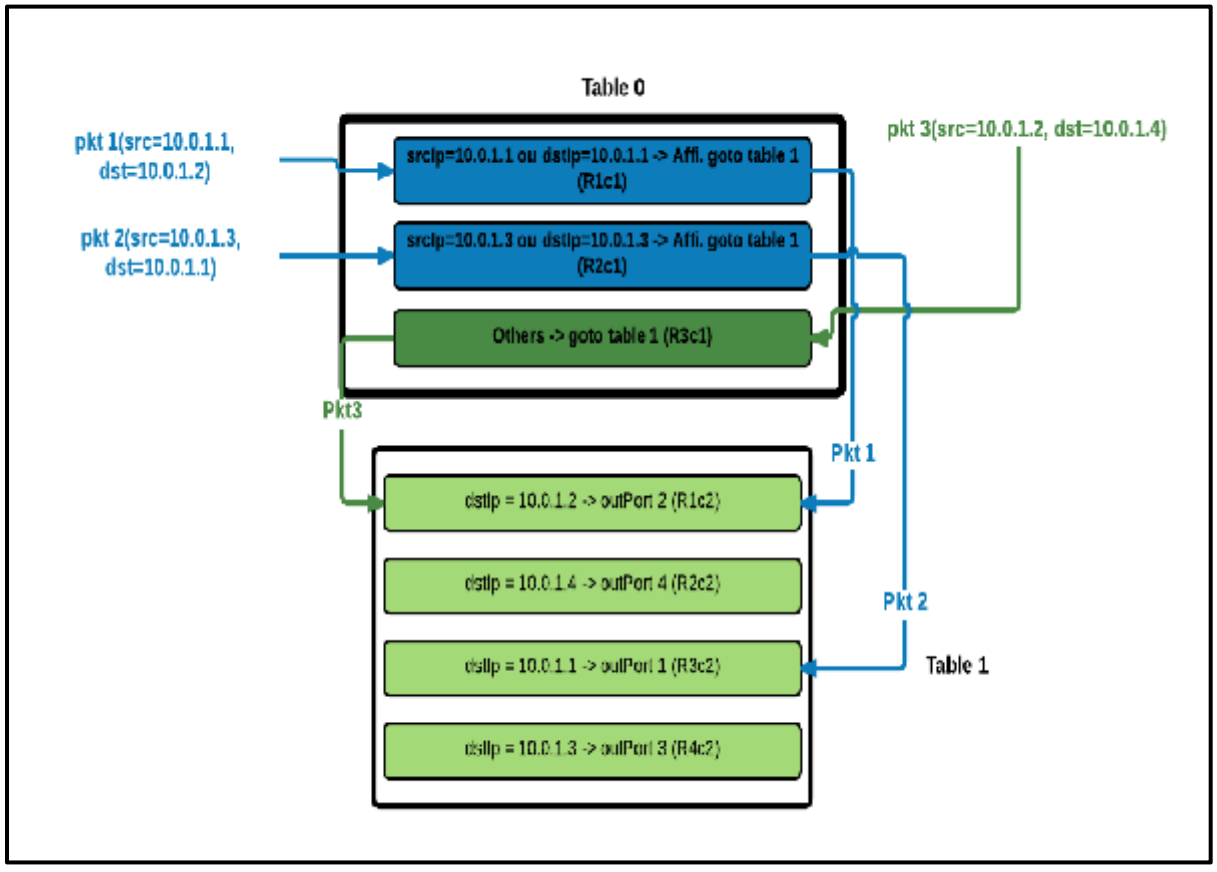


Figure 2.6 Méthode de traitement de paquets pour la composition parallèle

2.4 Hyperviseur de haut niveau

En SDN, le plan de gestion permet de donner l'opportunité de créer des règles à partir d'un langage de haut niveau comme Java, Python. Les solutions comme Frenetic et Pyretic sont utilisées chacune avec un seul contrôleur. Il est impossible d'utiliser plusieurs plateformes de programmation pour un seul contrôleur. Les travaux dans le cadre de cette maîtrise proposent une couche d'*hypervision* appelée CoreticHip qui permet d'utiliser plusieurs plateformes de programmation de haut niveau pour un seul contrôleur. Le reste de cette partie présente l'architecture utilisant l'hyperviseur spécialisé ainsi que le fonctionnement de celui-ci.

2.4.1 Architecture utilisant l'hyperviseur spécialisé

La solution d'*hypervision* est constituée de deux composantes : HCoreticServeur et HCoreticClient (voir figure 2.7). Les deux composants ensemble forment l'hyperviseur des plateformes de programmation de haut niveau. L'hyperviseur permet de gérer les flux partant du contrôleur aux plateformes et vice versa. Il empêche les paquets provenant du plan de données de se rendre à la plateforme d'un client non légitime. C'est-à-dire qu'un paquet destiné à la plateforme 1 ne se rendra pas à la plateforme 2. L'architecture proposée s'inspire de celle de Pyretic. Elle s'inspire aussi du modèle d'architecture 5 expliqué dans la revue de littérature. CoreticHip utilise aussi le fonctionnement de Flowvisor. CoreticHip n'entraîne pas un délai important dans le traitement des demandes reçues du plan de données. La solution d'*hypervision* agit comme un filtre qui empêche les flux de se diriger vers le mauvais plan de gestion ou vers les mauvais commutateurs. La génération des règles se fait au niveau des contrôleurs de haut niveau. La solution d'*hypervision* se situe entre le client OpenFflow et le Backend des plateformes de programmation. En procédant à l'insertion de la solution d'*hypervision* entre le client du contrôleur et les plateformes, il est plus simple de contrôler les communications entre les deux entités. Le rôle ainsi que le fonctionnement des composants seront expliqués à la section 2.4.2.

2.4.2 Fonctionnement

L'architecture proposée utilise celle de Pyretic à laquelle les composants HCoreticClient et HCoreticServeur ont été ajoutés. La suite de la section 2.4.2 expliquera les rôles ainsi que le fonctionnement des composants ajoutés qui sont responsables de créer l'*hypervision* entre les plateformes de programmation de haut niveau.

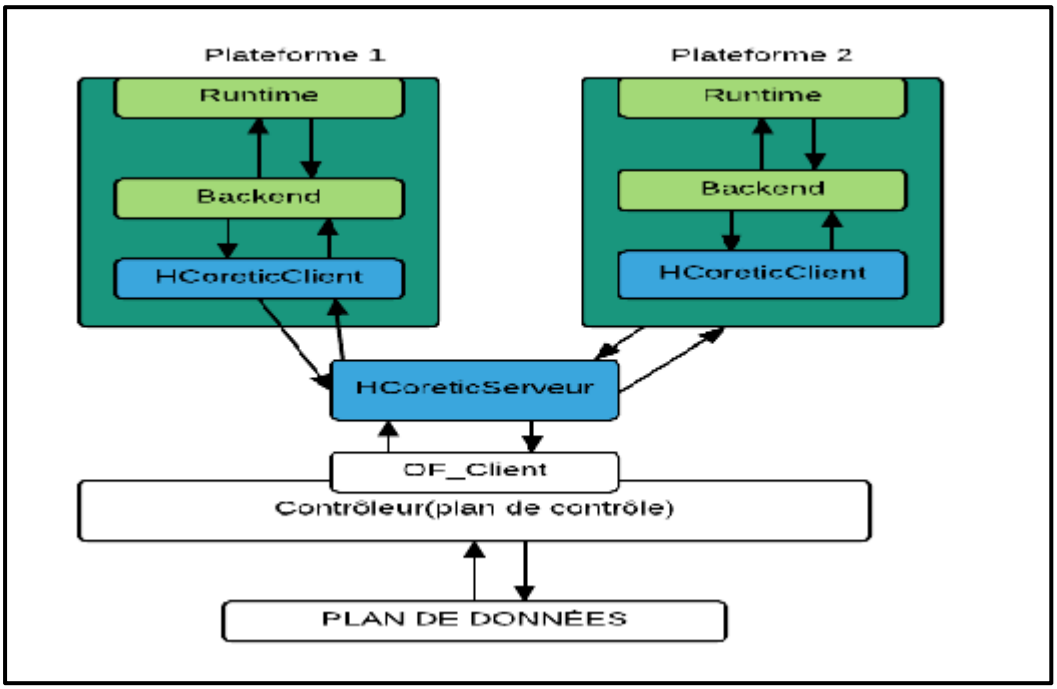


Figure 2.7 Architecture SDN incluant l'hyperviseur de haut niveau

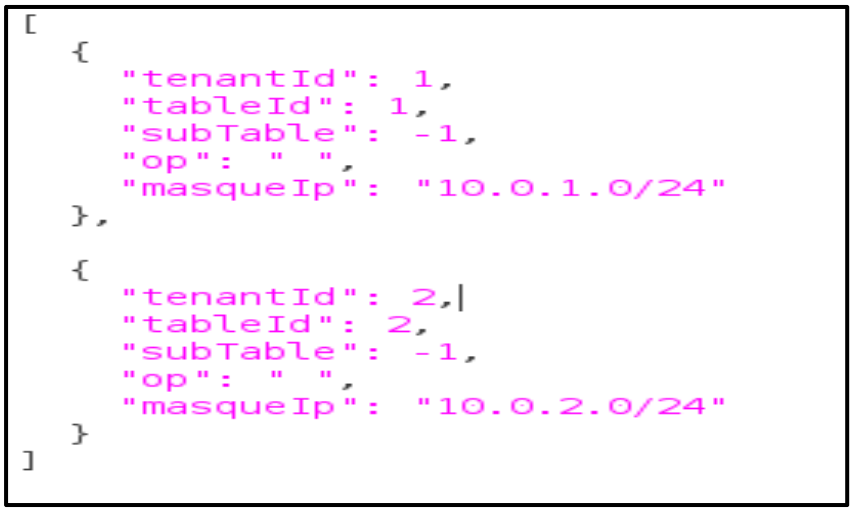
2.4.2.1 HCoreticServeur

HCoreticServeur est le composant situé dans le serveur contenant le contrôleur. Dans ce travail de maîtrise, le contrôleur POX est utilisé à cause de l'existence d'un client pour l'écriture des règles dans les tables de flux. Il a deux rôles : recevoir les informations du réseau dans le plan de données et configurer la table 0 ainsi que les tables destinées à chaque client qui sont les plateformes de programmation de haut niveau.

- **Réception des informations de réseau** : HCoreticServeur reçoit les informations de réseau du plan de données. Il reçoit les informations sur les commutateurs ainsi que les machines connectées. Pour recevoir ces informations, une communication est établie entre le

composant et le client OF_Client. Ce dernier reçoit les informations du contrôleur. HCoreticServeur, après avoir reçu les informations de réseau, il achemine celles-ci aux composants HCoreticClient en utilisant des Sockets.

- **Configuration de la table 0 et initialisation des tables des clients** : HCoreticServeur reçoit un fichier Json en paramètre (voir figure 2.8). Avec ce fichier, il configure la table 0 pour acheminer les flux vers les bonnes tables de flux. La partie 2.2.2 en donne plus de détails, car ce sont les mêmes principes qui sont utilisés pour isoler les trafics avec plusieurs tables. Les tables des clients sont aussi initialisées avec une règle par défaut qui est de supprimer les paquets n'appartenant pas au client d'une table donnée.



```
[
   {
      "tenantId": 1,
      "tableId": 1,
      "subTable": -1,
      "op": " ",
      "masqueIp": "10.0.1.0/24"
   },

   {
      "tenantId": 2,
      "tableId": 2,
      "subTable": -1,
      "op": " ",
      "masqueIp": "10.0.2.0/24"
   }
]
```


Figure 2.8 Fichier Json de configuration pour HCoreticServeur

Une partie de la solution d'hyperviseur est effectuée par la composante HCoreticClient qui est un mandataire installé sur chaque client.

2.4.2.2 HCoreticClient

Sur chaque client, le composant HCoreticClient joue deux rôles : filtrer les paquets provenant de HCoreticServeur et modifier les instructions FlowMod pour que les règles soient exactement écrites aux tables assignées.

- **Filtrage des paquets provenant de HCoreticServeur** : Le composant HCoreticClient se sert d'un fichier de configuration Json pour pouvoir filtrer les paquets qui ont le droit d'appartenir à un client donné. Dans le fichier Json les informations concernant la table, le masque d'adresses IP, les machines assignées dans le plan de données, les commutateurs ainsi que les différents ports de connexion y sont inscrits. Lorsqu'un paquet est reçu de HCoreticServeur, HCoreticClient vérifie si le paquet appartient au client en se servant de l'adresse source et destination. Si le paquet n'appartient pas au client, il est tout simplement ignoré.

- **Modification des instructions d'écriture de règles** : Lorsqu'un client veut envoyer une règle au plan de données, le composant HCoreticClient intercepte ce message afin d'y indiquer le numéro de la table appartenant au client pour un message de type Flowmod. Pour un message Packetout, le paquet est acheminé seulement au commutateur qui est apte à traiter le message. Il est possible que plusieurs commutateurs reçoivent le Packetout si le flux les traverses. HCoreticClient est capable de connaître les commutateurs qui ont le droit de recevoir Packetout grâce au fichier de configuration transmis au démarrage des composants du client.

2.5 Conclusion

Dans ce chapitre 2, les solutions CoreticIsol, CoreticComp et CorticHip ont été expliquées. CoreticIsol se sert des segments d'adresse IP et de plusieurs tables pour rediriger les flux de la table maître vers les tables des clients. CoreticComp offre une solution de composition séquentielle et parallèle en utilisant plusieurs tables. Un processus est établi dans le but de faire passer les flux d'une table à l'autre tout en respectant les règles de composition séquentielle et parallèle. Quant à CoreticHip, ses composants ont été présentés. Ils se situent entre le plan de contrôle et le plan de gestion. Ces composants permettent de gérer la confidentialité des messages entre les contrôleurs de haut niveau.
Le chapitre 2 a permis de donner des détails pour les solutions d'isolation, composition et pour l'hyperviseur de haut niveau. Le chapitre 3 qui suit explique les moyens utilisés pour
Évaluer les solutions.

CHAPITRE 3

ÉVALUATIONS

Coretic se différencie des travaux antérieurs. Elle apporte une nouvelle façon pour faire de l'isolation, la composition des contrôleurs de bas niveau. Elle apporte une l'idée de faire de l'*hypervision* des contrôleurs de haut niveau. Dans le but de s'assurer d'atteinte les objectifs, des évaluations doivent être effectuées. Ce chapitre va donc expliquer les démarches effectuées pour évaluer la solution développée. En premier lieu, les réseaux d'ordinateurs qui ont permis de faire l'évaluation seront expliqués dans la section 3.1. Le plan de données sera expliqué dans la section 3.2 par la suite. Ensuite la section 3.3 expliquera comment les règles sont envoyées à partir du plan de gestion et du plan de contrôle. La section 3.4 expliquera les mesures prises en compte incluant les moyens utilisés. Le chapitre 3 se terminera par la conclusion qui va revenir sur les principaux points abordés.

3.1 Réseaux d'ordinateurs utilisés

Pour exécuter les plans de données, de contrôle et de gestion, Virtualbox (version 5.0.26) a été utilisé. Les machines virtuelles créées s'exécutent dans une machine physique dont les caractéristiques sont dans le tableau 3.1. Deux réseaux d'ordinateurs ont été utilisés avec Virtualbox : un pour l'isolation et la composition et un autre pour l'*hypervision* des contrôleurs de haut niveau.

3.1.1 Pour l'isolation et la composition

Pour faire l'isolation et la composition, deux machines virtuelles ont été utilisées. Les deux machines virtuelles sont mises en réseau en utilisant Virtualbox.

La machine avec système d'exploitation Ubuntu 14.04.4 LTS exécute le plan de données avec Mininet. Mininet est un outil d'émulation des ressources du plan de données. Il est beaucoup utilisé dans le milieu de la recherche (Wang, S. Y. 2014). Le plan de contrôle et le plan de

gestion s'exécutent dans une autre machine appelée Serveur qui utilise aussi Ubuntu 14.04.03 LTS. Les caractéristiques de ces machines virtuelles se retrouvent dans le tableau 3.2.

Tableau 3.1 Caractéristiques de la machine physique

Modèle de machine	Ordinateur portable HP
Mémoire Ram	8Go
Système d'exploitation	Windows 10
Processeur	Hexa-Core 2.0GHz
Type de système	64bits
Disque dur	930Go

Tableau 3.2 Caractéristiques des machines virtuelles Mininet et du Serveur

	Mininet	**Serveur**
Mémoire Ram	2048Mo	1024Mo
Système d'exploitation	Linux (Ubuntu 14.04.4 LTS)	Linux(Ubuntu 14.04.3 LTS)
Processeur	1996MHz	1996MHz
Type de système	64bits	64bits
Disque dur	8589Mo	8589Mo
Adresse IP	192.168.56.102	192.168.56.104

3.1.2 Pour l'hyperviseur des contrôleurs de haut niveau

L'évaluation pour l'hyperviseur des contrôleurs de haut niveau demande deux architectures : sans CoreticHip et avec CoreticHip.

- **Sans CoreticHip** : L'architecture sans CoreticHip permet d'utiliser deux machines virtuelles. L'un contenant Mininet et l'autre contenant la plateforme Pyretic. Les deux machines ont les mêmes caractéristiques comme dans le tableau 3.2. Les deux machines sont connectées à travers un réseau informatique utilisant Virtualbox.

- **Avec CoreticHip** : L'architecture ici utilise 3 machines : une utilisant Mininet, une autre jouant le rôle de serveur, et une autre exécutant la plateforme de programmation Pyretic.

La section 3.1 a permis de connaître les architectures utilisées pour l'évaluation. La suivante donne les détails pour la simulation du plan de données.

3.2 Simulation de plan de données

Pour bâtir le plan de données, l'outil qui a été utilisé est Mininet dans sa version 2.2.0. Dans le cadre de cette maîtrise, une topologie linéaire est utilisée. Cinq commutateurs connectés linéairement sont reliés chacun à 20 hosts (Figure 3.1)**.** Pour pouvoir déployer la topologie linéaire avec cinq commutateurs et cent hosts, un programme écrit en Python utilisant les API de Mininet est utilisé. L'annexe I donne le pseudo-code qui décrit le comportement principal du script. Le script crée cinq commutateurs et associe 20 hosts à chacun d'eux. Si deux clients doivent se partager les équipements du plan de données, sur chaque commutateur, les dix premiers ports de chaque commutateur sont destinés au premier client et les dix autres derniers sont destinés au deuxième client.

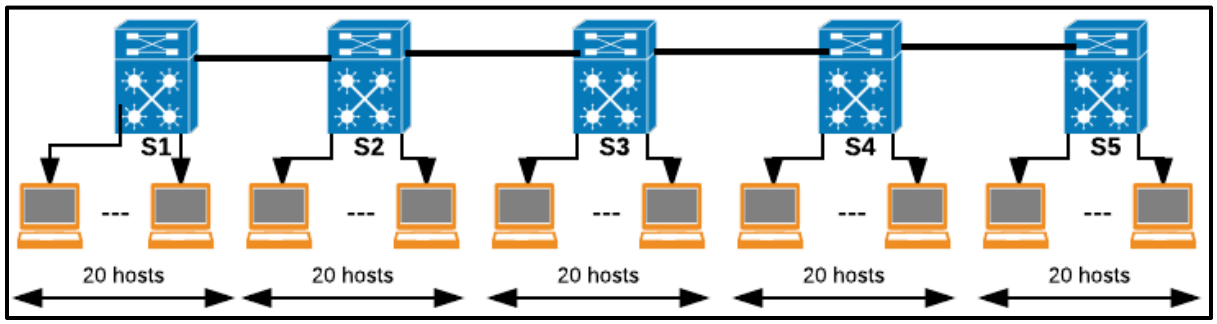


Figure 3.1 Topologie physique dans le plan de données

La section 3.2 explique comment le plan de données a été mis en place. La section 3.3 qui suit explique comment, les règles ont été envoyées aux différents commutateurs pour faire les évaluations.

3.3 Méthode d'envoi des règles au plan de données

Afin d'écrire les règles de traitements des flux, l'utilisation d'un moyen automatique est nécessaire. Il est moins pratique d'écrire manuellement des règles pour 100 machines virtuelles. Le plan de contrôle offre des API pour pouvoir écrire des règles dans les

commutateurs. Dans le cadre de cette maîtrise, le contrôleur POX a été utilisé. Pour encore rendre la tâche moins complexe, la plateforme Pyretic, dans le plan de gestion, a été aussi utilisée. Pyretic, à l'aide d'un script appelé "pyretic.py" (annexe II), écrit en langage Python, permet d'exécuter un module appelé "of_client" ainsi que de démarrer le contrôleur POX. Le module "of_client" qui est en réalité un client au contrôleur POX permet d'effectuer des écritures dans les tables de flux et de gérer les évènements reçus du plan de données. Dans le script présenté, il est possible de spécifier un client désiré pour des besoins bien précis. Pour chaque solution d'écriture de règles de gestion de flux dans les commutateurs, il y a un module "of_client" correspondant. La solution d'écriture de cette maîtrise est comparée avec d'autres solutions. Il faut aussi noter que les règles de gestion des paquets ont été envoyées dans le même ordre pour les évaluations effectuées. Considérons trois règles *R1, R2 et R3*. Quelle que soit la solution d'*hypervision* étudiée, les règles seront envoyées dans cet ordre : *R1-R2-R3*. Il y a deux façons d'envoyer les règles : lorsqu'il y a utilisation de plusieurs tables et lorsqu'il y a utilisation d'une seule table.

3.3.1 Avec plusieurs tables

Pour envoyer les règles avec prise en compte de l'utilisation de plusieurs tables, le module Nicira est utilisé. Nicira permet de manipuler plusieurs tables en OpenFlow 1.0. Pour écrire les règles avec utilisation de plusieurs tables, la commande suivante est utilisée : " ***pyretic.py --nx --enable_multicon --isServer --configTableFile pyretic/pyretic/jsonGenerator/initConfigTableServer.json server***". Comme on peut le voir, l'option **"--nx"** est utilisée pour activer l'utilisation de plusieurs tables. Pour les trois axes de travail qui sont l'isolation, la composition et l'*hypervision* des contrôleurs de haut niveau, il y a utilisation de plusieurs tables.

3.3.1.1 Pour l'isolation

Pour s'assurer de l'obtention de meilleures performances, la solution développée pour l'isolation(CoreticIsol) se compare avec deux autres solutions qui sont SDNMS et OpenVirtex. OpenVirtex sera détaillée dans la partie 3.3.2.1 portant sur l'utilisation d'une seule table de flux pour faire l'isolation. Cette section 3.3.1.1 va expliquer comment est-ce que les règles d'isolation pour CoreticIsol et celles de SDNMS ont été générées et envoyées au plan de données.

3.3.1.1.1 Envoi des règles pour CoreticIsol

Pour faire l'isolation, CoreticIsol se sert des adresses réseaux des clients. Pour spécifier au script Pyretic.py les informations des clients, il faut lui fournir un fichier Json identique à celui vu à la section 2.4.2.1 (figure 2.8). Ce fichier Json sera ensuite utilisé par le client POX destiné à insérer les règles pour CoreticIsol. Le client pour CoreticIsol va écrire des règles dans la table 0 de chaque commutateur afin d'acheminer les paquets vers les bonnes tables des clients. L'algorithme permettant d'envoyer les règles pour CoreticIsol dans les tables 0 est représenté par un pseudo-code en annexe III. L'algorithme achemine les flux vers les tables en fonction du masque d'adresse IP qui est différent d'un client à l'autre.

3.3.1.1.2 Envoi des règles pour SDNMS

Contrairement à CoreticIsol, SDNMS isole les flux en se basant sur les informations de la couche 2 du modèle OSI. Pour pouvoir envoyer les paquets ou les trames aux tables des clients, SDNMS se base sur chaque adresse MAC source lorsque les trames sont envoyées à tous les hosts (cas d'un Broadcast ARP). Si l'adresse de destination MAC est spécifiée, la table 0 se sert de celle-ci pour envoyer les trames vers les tables des clients. L'annexe IV donne un aperçu de l'implémentation des instructions Python envoyant des règles à la table 0 à travers un pseudo-code. Les tables des clients contiennent les règles pour simuler le comportement d'un commutateur.

3.3.1.2 Pour la composition

Pour les évaluations de performance en composition, la topologie dans la figure 3.1 a été utilisée. Pour la composition, un seul client est considéré et les règles sont envoyées pour les 100 machines virtuelles. En plus de la solution développée pour cette maîtrise, les solutions de composition séquentielle et parallèle pour Covisor ont été implémentées. Covisor utilise une seule table. L'envoi des règles pour Covisor sera expliqué dans la section 3.3.2. La section 3.3.1 explique les méthodes d'envoi pour les solutions utilisant plusieurs tables. Le reste de la section 3.3.1.2 va expliquer les méthodes d'envoi de règles pour la solution CoreticComp.

Pour envoyer les règles pour la composition séquentielle, les mêmes techniques ont été utilisées comme dans le cas de l'isolation. La plateforme Pyretic a été utilisée. Un script client

écrit en Python (voir pseudo-code en annexe V) permet d'envoyer des règles à la table 0 pour bloquer les paquets indésirables pour la composition séquentielle. Il faut rappeler dans le scénario mis en place est que la table 0 reçoit les règles de coupe-feu qui suppriment tous les paquets ayant dans son entête une adresse IP impaire. La table 1 reçoit par la suite les règles de commutation pour les paquets n'ayant pas d'adresse IP impaire dans leur entête. Pour la composition parallèle, il y a la même technique d'envoi de règles qui est adoptée. Mais les paquets ayant des adresses IP impaires sont affichés à l'écran avant d'être acheminés vers la table suivante. Les règles pour l'affichage sont insérées dans la table 0. Il faut noter qu'au niveau de la composition, les règles pour les *hosts* utilisés pour faire des évaluations sont écrites en dernier.

3.3.1.3 Pour l'*hypervision* des contrôleurs de haut niveau

Afin d'évaluer l'effet de l'insertion de CoreticHip entre le plan de contrôle et le plan de gestion, les règles ont été envoyées avec et sans CoreticHip. Le mode interprété est utilisé dans le but de simuler le plus de trafic entre le plan de contrôle et le plan de gestion. Le mode interprété donne une assurance de l'envoi des requêtes vers le plan de gestion. Ainsi il est plus adéquat d'évaluer les performances de CoreticHip. Il est à noter que les règles sont écrites en Python à l'aide du langage Python. Un module est écrit pour faire de l'isolation entre deux clients dans le cas où CoreticHip est absent (voir annexe VI). Avec la présence CoreticHip, les mêmes principes d'isolation sans utilisés comme avec CoreticIsol avec l'isolation à partir de la table 0. Le module utilisé dans le plan de gestion et écrit en langage Pyretic est représenté dans l'annexe VII. Les règles sont écrites pour permettre de faire communiquer les machines virtuelles extrêmes : h1v1 et h50v1 qui appartiennent à un client donné.
Le point 3.3.1 vient d'expliquer les moyens utilisés pour générer les règles pour les solutions utilisant plusieurs tables. Le point 3.3.2 qui suit explique les moyens utilisés pour générer les règles des solutions utilisant une seule table.

3.3.2 Avec une seule table

Les solutions d'*hypervision* dans cette catégorie n'utilisent qu'une seule table. Il y a comme conséquence un manque de cohésion et une quantité importante de règles pour une seule table.

Dans cette catégorie, deux solutions seront implémentées et comparées à CoreticIsol et CoreticComp.

3.3.2.1 Pour l'isolation

La solution d'isolation qui sera implémentée et comparée à CoreticIsol est OpenVirtex. Tout comme les autres solutions, des structures de données sont utilisées pour fournir des informations à un algorithme pour écrire des règles dans les commutateurs. Mais ici il y a utilisation d'une seule table. L'annexe VIII montre l'implémentation de l'algorithme principal (en pseudo-code) utilisé pour écrire des règles simulant la solution d'OpenVirtex.

3.3.2.2 Pour la composition

La solution Covisor a été implémentée dans le but de faire une comparaison de performance pour la composition. Le pseudo-code du script Python client développé pour la composition séquentielle se retrouve en annexe IX. Ce script une fois exécuté avec le fichier "pyretic.py " de la plateforme de Pyretic, permet d'écrire des règles seulement dans la table 0. Les règles générées permettent de supprimer les paquets ayant en entête une adresse IP impaire. Pour les autres paquets, les règles de commutation permettent d'acheminer les paquets vers les bons ports des commutateurs. Pour la composition parallèle, le pseudo-code du script Python qui aide à écrire des règles se retrouve en annexe X.
La section 3.3 vient d'expliquer les moyens utilisés pour générer les règles. La section 3.4 explique les mesures effectuées.

3.4 Mesures effectuées

Les solutions développées dans ce travail sont évaluées selon deux modes : mode proactif et mode interprété.

3.4.1 Mesures en mode proactif

Les solutions concernées ici sont SDNMS, OpenVirtex, Covisor, CoreticIsol et CoreticComp. Les mesures effectuées sont : latence, débit de paquets, débit de la bande passante. Ces mesures sont faites en se servant de la machine Mininet qui contient le plan de données.

3.4.1.1 Latence

La latence consiste à mesure le délai d'envoi et de réception de requêtes entre une machine et une autre (figure 3.2). Pour un client donné, le délai est mesuré entre chaque host.

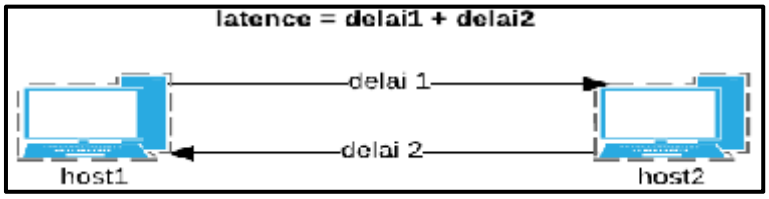


Figure 3.2 Illustration de la mesure de la latence

Afin de s'assurer des résultats obtenus, douze mesures de latence sont effectuées entre des machines qui sont capables de communiquer. Une mesure est la moyenne obtenue à la suite d'une itération de 100 tests de latences sauf la dernière mesure qui est de 125. Chaque test de latence consiste à faire la moyenne de dix tests unitaires de mesure de latence. Pour les mesures en lien avec les solutions d'isolation, les requêtes ICMP sont faites entre toutes les machines d'un client. Pour les solutions de composition, les mesures sont prises que pour des machines situées sur les extrémités. Par exemple pour un ensemble de machines dont les adresses IP vont de 10.0.1.1 à 10.0.1.100 et qui sont capables de communiquer, alors la latence entre ces deux machines sera mesurée. Un seul paquet est injecté à la fois dans le réseau. La figure 3.3 montre le script Python qui automatise la mesure de Latence à l'aide de HPING3. Pour mesurer la latence entre une machine h1 (IP=10.0.2.1) et la machine h100(IP=10.0.1.100), il faut d'abord effectuer un "xterm" sur la machine h1 avec la commande "***xterm h1***". Xterm permet d'obtenir la machine h1 en ligne de commande. Une fois le terminal obtenu, il suffit d'exécuter le script Python. Ce script prend deux paramètres qui sont l'adresse IP de destination et le nombre de fois que la mesure de latence doit être prise. Il est à noter que pour les mesures prises pour les tests d'isolation de trafic, les requêtes ICMP générées impliquent toutes les machines dans le

réseau virtuel d'un client. Les deux machines situées sur les deux extrémités n'ont pas été les seules à être utilisées. Par exemple si un client a un réseau virtuel de 10 machines, les requêtes ICMP les impliqueront tous.

```
import time
import sys
import  subprocess
nombreTest = int(sys.argv[1])
ipDest = str(sys.argv[2])
for x in range(1,nombreTest+1):
                print "test: ",str(x)
                proc = subprocess.Popen("hping3 -q -c 10 -i u100000 %s"%ipDest, shell=True, stdout =subprocess.PIPE)
                proc.wait()
print "finish"
```

Figure 3.3 Script de prise de mesure automatique de la latence en mode proactif

3.4.1.2 Débit de paquets

Le débit de paquets ici permet de mesurer le nombre de paquets perdus en fonction du débit d'injection utilisé. Avec ce test, la résistance des solutions d'*hypervision* sera mesurée. Il sera possible de constater la perte de paquets en fonction de la vitesse à laquelle les requêtes ICMP sont injectées. Pour cette fois, deux hosts se situant dans les deux extrémités de la topologie virtuelle d'un client sont utilisés. Pour un client de cinquante hosts, l'évaluation du débit de paquets va utiliser le host h1 et le host h50. Cent tests sont effectués. À chaque test la vitesse à laquelle les paquets sont injectés dans le réseau augmente de 100Pkt/s. Les tests vont donc de 100Pkts/s à 10000Pkts/s. À chaque test, cinquante mille paquets sont injectés. Le script qui automatise le test pour le débit se trouve dans la figure 3.4.

```
import time
import  subprocess
import sys
nbrePaquetSeconde=0
tour = 1
ipdest = str(sys.argv[1])
for x in range(1,101):
                nbrePaquetSeconde +=100
                delaiPourEnvoi = 1000000/nbrePaquetSeconde
                print "-------------------------------------\n"
                print "pkts/s:",str(nbrePaquetSeconde)
                print "tour: ", str(tour)
                proc = subprocess.Popen("hping3 -q -c 50000 -i u%s %s"%(str(delaiPourEnvoi),ipdest), shell=True, stdout =subprocess.PIPE)
                tour +=1
                proc.wait()

print "finish"
```

Figure 3.4 Script d'automatisation de prise de mesure de débit

3.4.1.3 Débit de la bande passante

Dans un réseau d'ordinateurs, la capacité de la bande passante est un élément important. En SDN, la banque passante est affectée par les entrées dans les tables flux. Étant donné que cette étude propose une solution d'écriture de règles, la mesure de la bande passante devient pertinente. Pour cela, l'outil Iperf3 est utilisé. Comme dans le cas du test avec les débits, les deux extrêmes des machines dans un réseau virtuel sont utilisés. L'un se comportant comme le serveur et l'autre comme le client. La machine se comportant comme le serveur exécute cette commande ci-après "***iperf3 -s -V*** ". Pour la machine se comportant comme le client, un fichier contenant des commandes Bash est exécuté. Le contenu de ce fichier est représenté par la figure 3.5. Une connexion TCP est utilisée par le client pour communiquer avec le serveur avec une fenêtre de 256Ko. 128 Ko est la taille du tampon de lecture et d'écriture. Un test consiste à écrire 256Ko de données 1000 fois. Le test d'écriture de tampon est effectué 100 fois comme on peut le voir dans la boucle. Les deux directions des flux sont prises en compte. Ces deux directions sont lorsque les flux partent du client au serveur et lorsque les flux partent du serveur au client. Le premier est appelé flux d'envoi et le deuxième est appelé flux de réception.

```
import time
import  subprocess
import sys
ipDest = str(sys.argv[1])
tour = 1
for x in range(1,101):

                print "tour: ", str(tour)
                proc = subprocess.Popen["iperf3 -c 10.0.1.2 -f m -V -n 1000 -i 1 -w 256K", shell=True, stdout =subprocess.PIPE)
                proc.wait()
                tour +=1
                print "\n-------------------------------------"

print "finish"
```

Figure 3.5 Script de mesure de la bande passante pour le client

3.4.2 Mesures en mode interprété

Le mode interprété fait en sorte que toutes requêtes sont envoyées au plan de gestion pour traitement. Les mesures prises en mode proactif ci-haut sont les mêmes qu'en mode interprété.

Les prises de mesure en mode interprété prennent en compte une autre architecture où il n'y a pas CoreticHip. Cela permet de mesurer l'effet de l'insertion de l'hyperviseur de haut niveau. Les mêmes principes sont utilisés pour la gestion des flux au niveau du plan de gestion dans le cas où il y a la présence de CoreticHip et dans le cas où il est absent. Pour Pyretic, sans l'intervention de CoreticHip, un module unique permet de s'assurer que les paquets respectent les règles d'isolation avant l'application des règles de commutation. Étant donné que toutes les requêtes sont remontées vers le plan de gestion, il est judicieux de changer des paramètres dans les trois mesures effectuées.

3.4.2.1 Latence

Pour la latence, 5 tests de requêtes ICMP sont effectués entre les machines qui sont situées le plus à l'extrémité gauche du premier commutateur et à l'extrémité droite du dernier commutateur. Chaque test consiste à envoyer cinq requêtes ICMP entre les deux machines. Cela fait un total de 2500 requêtes ICMP. Dans chaque unité de test, on y retrouve la moyenne de latence de 500 requêtes ICMP. Il y a un délai de 800 ms entre chaque requête (voir figure 3.6). Le premier paramètre représente le nombre de tests. Le deuxième paramètre consiste à fournir l'adresse IP de destination.

```
import time
import sys
from threading import Thread
import  subprocess
nombreTest = int(sys.argv[1])
ipDest = str(sys.argv[2])
for x in range(1,nombreTest+1):
                #print "test: ",str(x+400)
                proc = subprocess.Popen("hping3 -q -c 5 -i u800000 %s"%ipDest, shell=True, stdout =subprocess.PIPE)
                proc.wait()
                time.sleep(4)
print "finish"
```

Figure 3.6 Script de mesure de la latence en mode interprété

3.4.2.2 Débit

Le test de débit consiste à envoyer 20 paquets pour des débits variant de 2 paquets par seconde à 20 paquets par seconde. Le script effectuant cela se trouve dans la figure 3.7.

```
import time
import  subprocess
import sys
nbrePaquetSeconde=1
tour = 1
ipdest = str(sys.argv[1])
for x in range(1,20):
                nbrePaquetSeconde +=1
                delaiPourEnvoi = 1000000/nbrePaquetSeconde
                print "-------------------------------------\n"
                print "pkts/s:",str(nbrePaquetSeconde)
                print "tour: ", str(tour)
                proc = subprocess.Popen("hping3 -q -c 20 -i u%s %s"%(str(delaiPourEnvoi),ipdest), shell=True, stdout =subprocess.PIPE)
                tour +=1
                time.sleep(30)
                proc.wait()

print "finish"
```

Figure 3.7 Script de mesure de débit en mode interprété

3.4.2.3 Bande passante

Le test de bande passante consiste à écrire 500 fois 256 Ko de données sur la bande passante. Le test est répété 100 fois (voir le script du client de test à la figure 3.8).

```
#!/bin/bash
#vmax=32
host1=10.0.1.1

for i in {1..100}
do
  iperf3 -c $host1 -f m -V -n 500 -i 1 -w 256K
  sleep 5
done
```

Figure 3.8 Script bande passante en mode interprété

3.5 Conclusion

Le chapitre 3 explique les moyens utilisés pour évaluer les performances des solutions. Il explique l'utilisation de Mininet pour mettre en place le plan de données avec cinq commutateurs et 100 *hosts*. Les moyens utilisés pour envoyer les règles dans le plan de données sont de deux manières : en mode proactif et en mode interprété. Le mode proactif concerne les solutions d'*hypervision* des contrôleurs de bas niveau en se servant des clients OpenFlow du contrôleur Pox. Le mode interprété fait intervenir les plans de gestion où les décisions sont prises en lieu et place du plan de contrôle. Les mesures prises sont la latence, le débit des paquets et le débit de la bande passante.

Le chapitre 3 explique les mesures et les moyens utilisés pour évaluer les solutions développées dans ce travail. Le chapitre 4 qui suit présente les résultats de tests.

CHAPITRE 4

RÉSULTATS ET ANALYSE DES TESTS

Un travail de recherche propose des solutions différentes des travaux antérieurs dans un domaine bien précis. Dans le but d'être crédible, un travail de recherche doit effectuer des tests et présenter des résultats. Ces résultats doivent être discutés dans le but de justifier les objectifs. C'est dans cette optique que s'inscrit ce chapitre 4. Les résultats de test de performance de Coretic seront présentés tout en les comparant avec les résultats de test d'anciens travaux. La vérification des objectifs sera aussi faite dans ce chapitre. De façon plus spécifique, la partie 4.1 parlera en détail des résultats de test pour l'isolation. Il s'en suivra la partie 4.2 qui parlera des résultats obtenus pour la composition. Quant à la partie 4.3, elle parlera des résultats de test de performance de l'hyperviseur des contrôleurs de haut niveau. La partie 4.4 ouvrira une discussion afin de s'assurer que les objectifs ont été atteints. Ce chapitre se terminera par une conclusion qui reviendra sur les points clés abordés.

4.1 Résultats de test pour l'isolation

Les résultats pour l'isolation seront présentés en se basant sur l'existence de deux clients. Les deux clients simulés ont chacun cinquante hosts. Sur chaque commutateur, le premier client se voit octroyer les dix premiers hosts et le deuxième possède les autres dix hosts. Les tableaux 4.1, 4.2, 4.3, 4.4 et 4.5 indiquent la répartition des ressources physiques.

Tableau 4.1 Répartition des ressources physiques du commutateur 1(S1)

Clients	Adresses IPs	Adresses MACs	Ports de connexion au commutateur
Client 1	De **10.0.1.1** à **10.0.1.10**	De **00:00 :00 :00 :01:01** à **00:00 :00 :00 :01:10**	**1** à **10**
Client 2	De **10.0.2.1** à **10.0.2.10**	De **00:00 :00 :00 :02:01** à **00:00 :00 :00 :02:10**	**11** à **20**

Tableau 4.2 Répartition des ressources physiques du commutateur 2(S2)

Clients	Adresses IPs	Adresses MACs	Ports de connexion au commutateur
Client 1	De **10.0.1.11** à **10.0.1.20**	De **00:00 :00 :00 :01:11** à **00:00 :00 :00 :01:20**	**1** à **10**
Client 2	De **10.0.2.11** à **10.0.2.20**	De **00:00 :00 :00 :02:11** à **00:00 :00 :00 :02:20**	**11** à **20**

Tableau 4.3 Répartition des ressources physiques du commutateur 3(S3)

Clients	Adresses IPs	Adresses MACs	Ports de connexion au commutateur
Client 1	De **10.0.1.21** à **10.0.1.30**	De **00:00 :00 :00 :01:21** à **00:00 :00 :00 :01:30**	**1** à **10**
Client 2	De **10.0.2.21** à **10.0.2.30**	De **00:00 :00 :00 :02:21** à **00:00 :00 :00 :02:30**	**11** à **20**

Tableau 4.4 Répartition des ressources physiques du commutateur 4(S4)

Clients	Adresses IPs	Adresses MACs	Ports de connexion au commutateur
Client 1	De **10.0.1.31** à **10.0.1.40**	De **00:00 :00 :00 :01:31** à **00:00 :00 :00 :01:40**	**1** à **10**
Client 2	De **10.0.2.31** à **10.0.2.40**	De **00:00 :00 :00 :02:31** à **00:00 :00 :00 :02:40**	**11** à **20**

Tableau 4.5 Répartition des ressources physiques du commutateur 5(S5)

Clients	Adresses IPs	Adresses MACs	Ports de connexion au commutateur
Client 1	De **10.0.1.41** à **10.0.1.50**	De **00:00 :00 :00 :01:41** à **00:00 :00 :00 :01:50**	**1** à **10**
Client 2	De **10.0.2.41** à **10.0.2.50**	De **00:00 :00 :00 :02:41** à **00:00 :00 :00 :02:50**	**11** à **20**

4.1.1 Latence

La latence mesurée a été faite avec le client 2. Des requêtes ICMP ont été faites entre tous les machines du réseau virtuel du client 2. Ce qui fait un total de 1225 dizaines de requêtes unitaires ICMP. Cela fait un total de 12 250 requêtes ICMP. Une itération consiste à faire 100 fois une dizaine de requêtes unitaires ICMP. La moyenne obtenue constitue un résultat d'un test. En faisant ces itérations, nous obtenons 12 tests. La douzième itération contient 125 dizaines de requêtes ICMP. La figure 4.1 montre une comparaison de mesure de latence entre CoreticIsol, SDNMS et OpenVirtex. Sur l'axe des abscisses, il y a les tests de latences

effectués. Sur l'axe des ordonnées, il y a les délais de latence en milliseconde. Sur chaque figure, la latence moyenne des solutions d'isolation est notée. En observant les moyennes, nous pouvons remarquer que la moyenne de latence pour CoreticIsol est la plus basse. Le tableau 4.6 montre de façon claire la moyenne de latence obtenue pour les trois solutions. Comme le montre ce tableau, la solution développée dans le cadre de cette maîtrise offre des mesures de latences inférieures à celles obtenues avec SDNMS et OpenVirtex. En analysant les délais moyens dans le tableau 4.6, Coretic à travers sa solution CoreticIsol offre une latence moyenne 4,73ms ce qui est meilleur à celle obtenue avec OpenVirtex qui est de 5,40ms et SDNMS qui est de 5,33ms. Avec Coretix, le délai moyen a baissé de 12,40% en comparaison avec OpenVirtex et une autre baisse de 11,25% en comparaison avec SDNMS.

4.1.2 Débit de paquets

Le débit mesuré consiste à évaluer le nombre de paquets perdus en fonction de la vitesse à laquelle les requêtes ICMP sont envoyées. La figure 4.2 montre une comparaison de paquets perdus en fonction du débit d'injection de paquets pour CoreticIsol, SDNMS et OpenVirtex. La vitesse d'injection des paquets qui sont des requêtes ICMP qui vont de 100 paquets par seconde à 10 000 paquets par seconde. Les courbes de couleurs bleu, vertes et rouge représentent respectivement les tendances de paquets perdus pour CoreticIsol, OpenVirtex(OVX) et SDNMS. En analysant la figure 4.2, la ligne bleue représentant la tendance de paquets perdus pour CoreticIsol est en dessous des autres tendances à l'exception de la vitesse allant de 9100 paquets par seconde à 9500 paquets par seconde. En moyenne, la figure 4.2 montre que CoreticIsol est plus résistante à perdre les paquets avec une moyenne de 177 paquets perdus. Quant à OVX et SDNMS la moyenne de paquets perdus est respectivement de 216 paquets et 187 paquets (voir tableau 4.7). L'amélioration de performance à perdre moins de paquets est de 18% en comparaison avec OVX et de 5,65% en comparaison avec SDNMS.

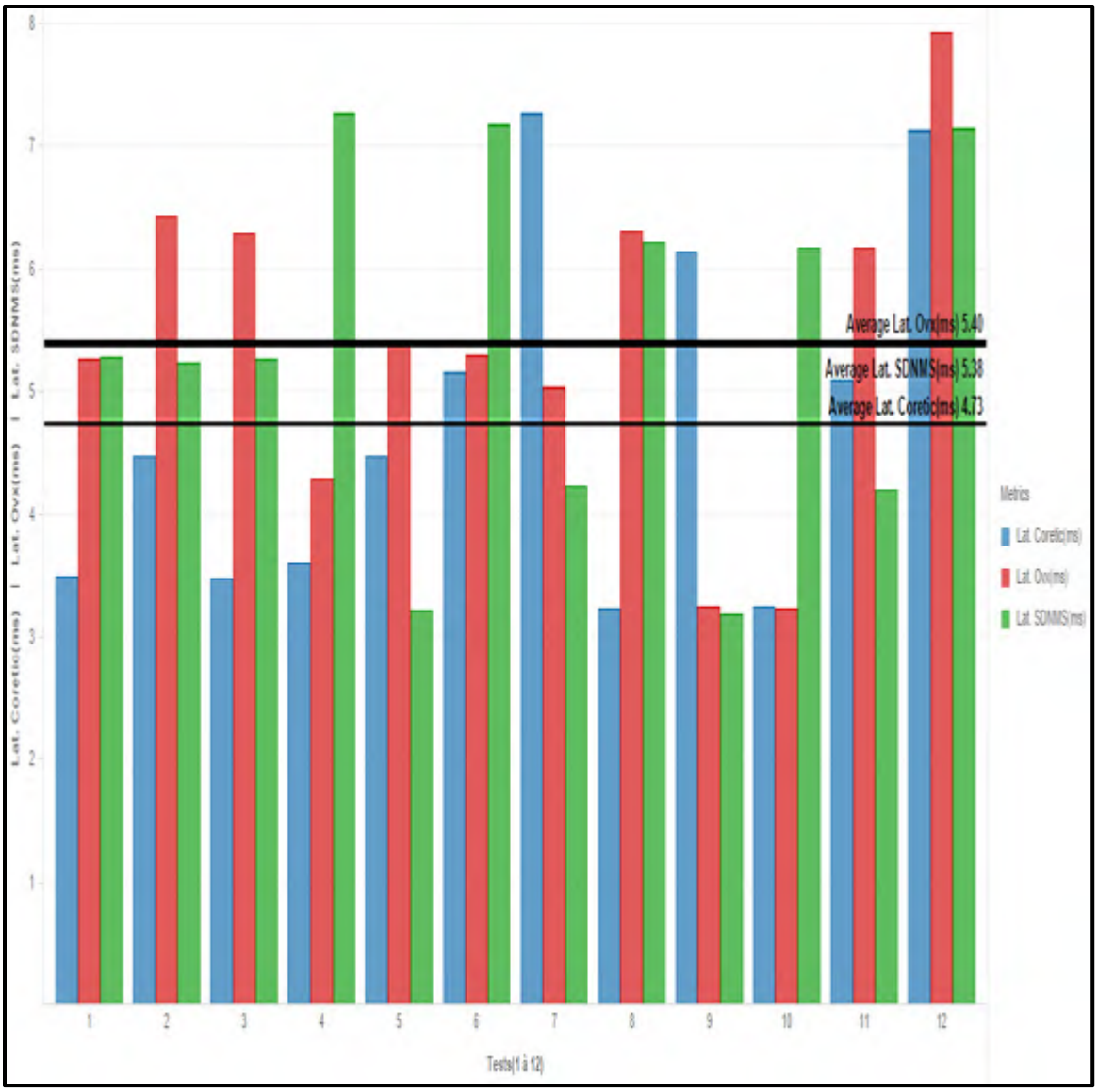


Figure 4.1 Résultat de mesures de latence pour CoreticIsol. SDNMS et OpenVirtex

Tableau 4.6 Tableau de comparaison de latence entre Coretic, OVX et SDNMS

Solutions	Moyenne (ms)
CoreticIsol	4,73
OpenVirtex	5,40
SDNMS	5,33

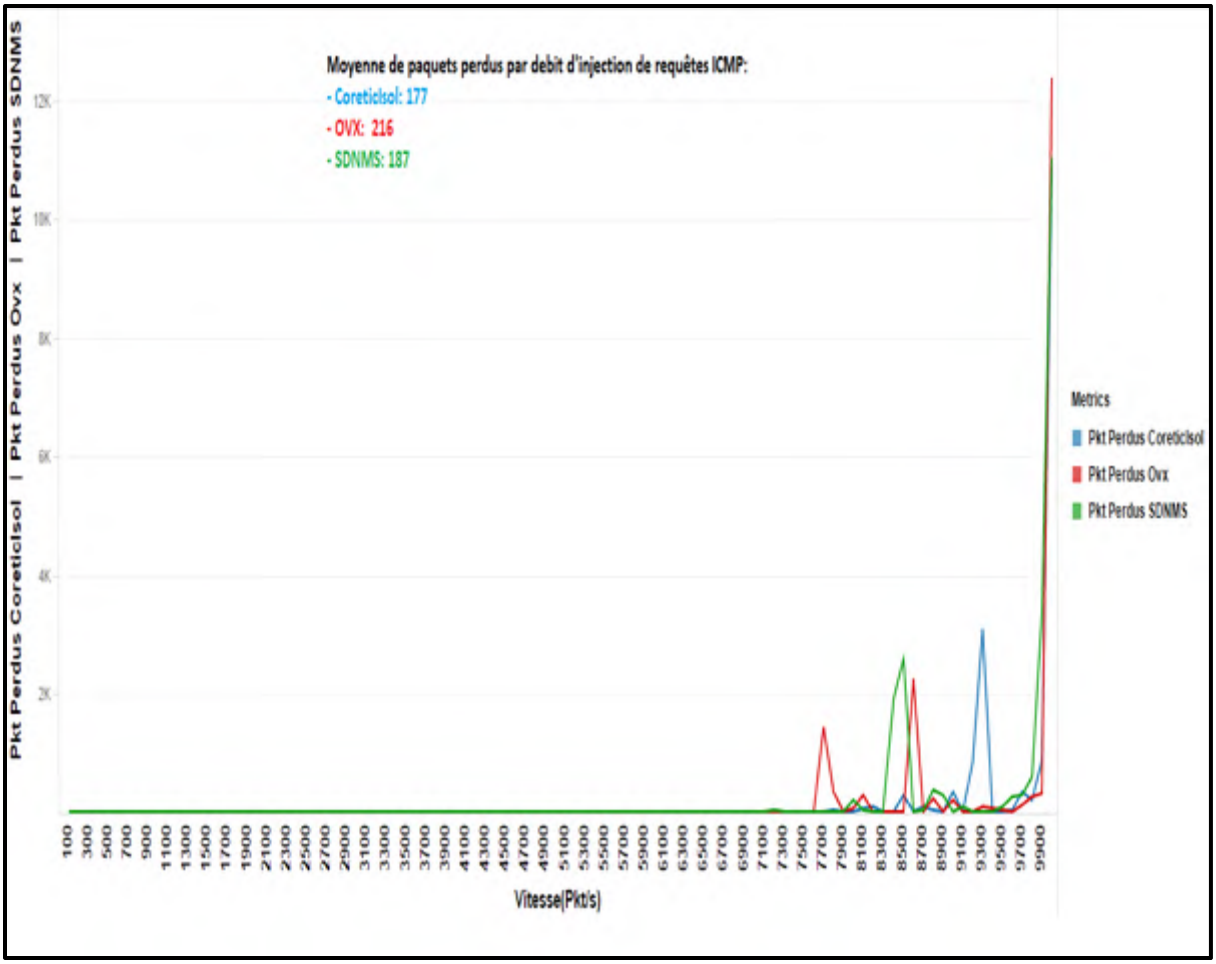


Figure 4.2 Résultat de mesure de paquets perdus en fonction du débit pour les solutions d'isolation

Tableau 4.7 Moyenne des paquets perdus par débit d'injection de paquets

Solutions	Moyenne paquets perdus par débit d'injection de paquets(nombre de paquets)
CoreticIsol	177
OpenVirtex	187
SDNMS	216

4.1.3 Bande passante

La figure de 4.3 donne les résultats obtenus pour les mesures prises sur les bandes passantes pour les trois solutions. Cette figure montre une comparaison des capacités des bandes

passantes des trois solutions. En abscisse il y a les tests effectués et en ordonnée il y a les capacités des bandes passantes en Mégabits par seconde(Mbits/s). Chaque test (1 à 5) représente la moyenne de 20 mesures de la bande passante. En lisant la figure 4.3, CoreticIsol offre une meilleure capacité de bande passante. Les mesures ne prennent en compte que les flux en envoi, c'est-à-dire les paquets allant du client au serveur. Une seule valeur est retenue, car les mesures obtenues en envoi et en réception sont les mêmes. En moyenne la bande passante est de 3 062,28Mbits/s pour CoreticIsol; 2 717,14Mbits/s pour SDNMS et 2 689,47 pour OpenVirtex (voir tableau 4.8). La capacité de la bande passante avec CoreticIsol est supérieure de 12,7% en comparaison avec SDNMS et de 13,86% en comparaison avec OpenVirtex.

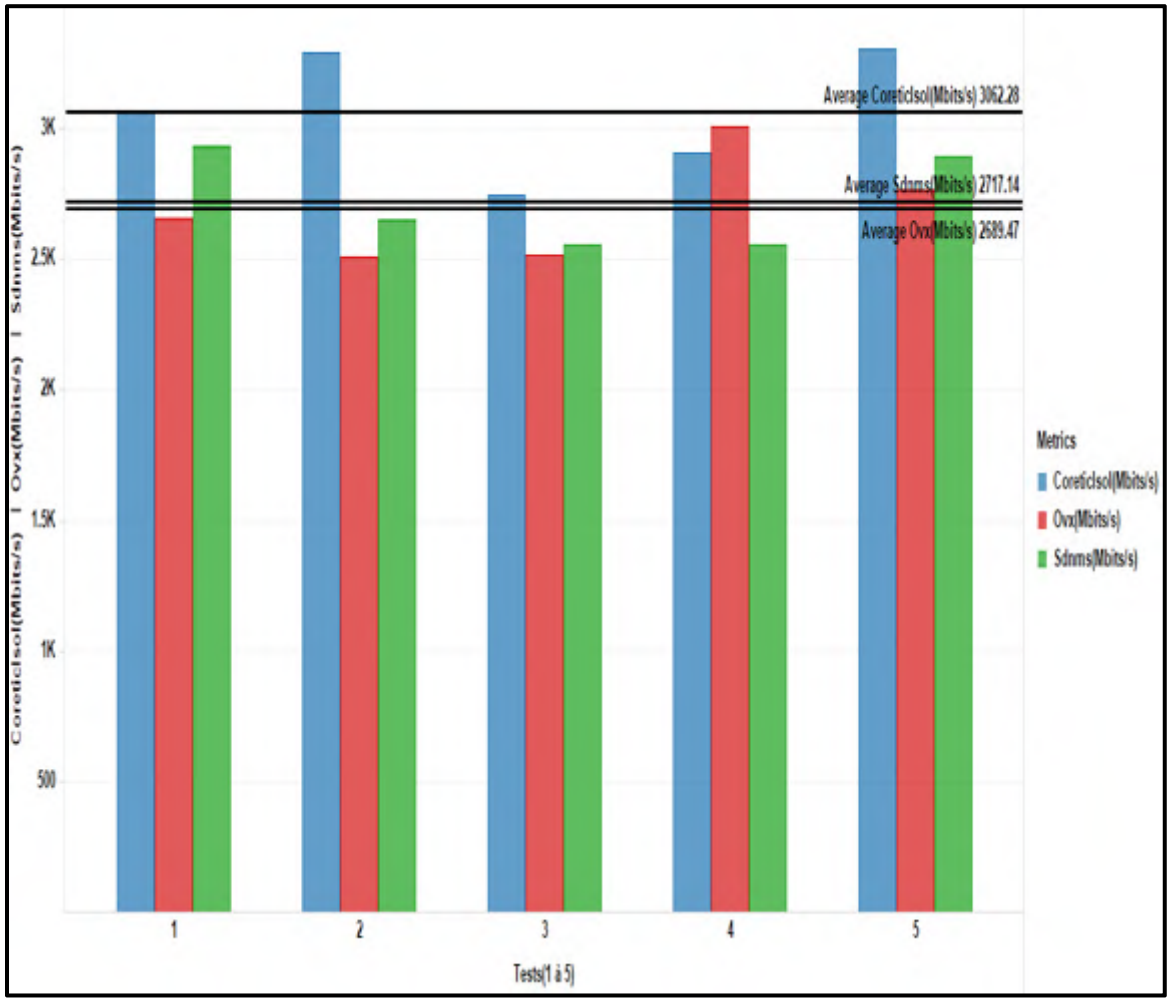


Figure 4.3 Résultats de mesure de bande passante pour l'isolation

Tableau 4.8 Moyenne de capacité de bande passante pour l'isolation

Solutions	Capacité des bandes passantes(Mbits/s)
CoreticIsol	3 062,28
OpenVirtex	2 689,47
SDNMS	2 717,14

4.2 Pour la composition

Les mesures de performance ont été prises en utilisant les solutions de composition de CoreticComp et Covisor. Pour chaque mesure de performance, les solutions de composition de CoreticComp et Covisor seront comparées.

4.2.1 Latence

La figure 4.4 indique les mesures de latence pour CoreticComp et Covisor pour la composition séquentielle. Quant à la composition parallèle, la figure 4.5 indique les mesures prises. Comme mentionné dans les tests d'isolation, sur l'axe des abscisses, il y a les tests effectués et sur l'axe des ordonnés, il y a les mesures prises. Les mesures ont été prises en utilisant les deux machines d'extrémité qui sont les machines 10.0.1.2 et 10.0.1.100. Les requêtes entre ces deux machines sont suffisantes, car les règles qui les concernent sont parmi les dernières. En moyenne la latence en composition séquentielle est de 6,28ms pour Covisor et 5,49ms pour CoreticComp (voir tableau 4.9). Ce qui fait une baisse en latence de 12,58%. Pour la composition parallèle, la moyenne de latence est de 9,86ms pour Covisor et de 7,25ms pour CoreticComp (voir tableau 4.10). Ce qui fait une autre baisse de latence qui est de 26,47%.

4.2.2 Débit de paquets

Le débit permet de mesurer la capacité des solutions à perdre moins de paquets en fonction de la vitesse d'injection des requêtes ICMP dans les tables de flux. Les paquets pour les requêtes ICMP sont envoyés avec un débit allant de 100 pkts/s à 10 000pkts/s avec 50 000 paquets injectés à chaque débit. Le nombre de paquets perdus est mesuré par la suite. Les résultats pour les compositions séquentielles et parallèles sont reportés dans la figure 4.6 et la figure 4.7. Ces

résultats montrent une comparaison des paquets perdus pour CoreticComp et Covisor. Pour la composition séquentielle, la moyenne de paquets perdus est de 3 268 paquets pour CoreticComp et de 3 726 paquets pour Covisor (voir tableau 4.11). Cela fait une capacité à perdre moins de paquets de 12,29%. Pour la composition parallèle, la moyenne est de 27 317 paquets perdus pour CoreticComp et de 36 096 paquets pour Covisor. Cela fait une augmentation de capacité de 24,32%.

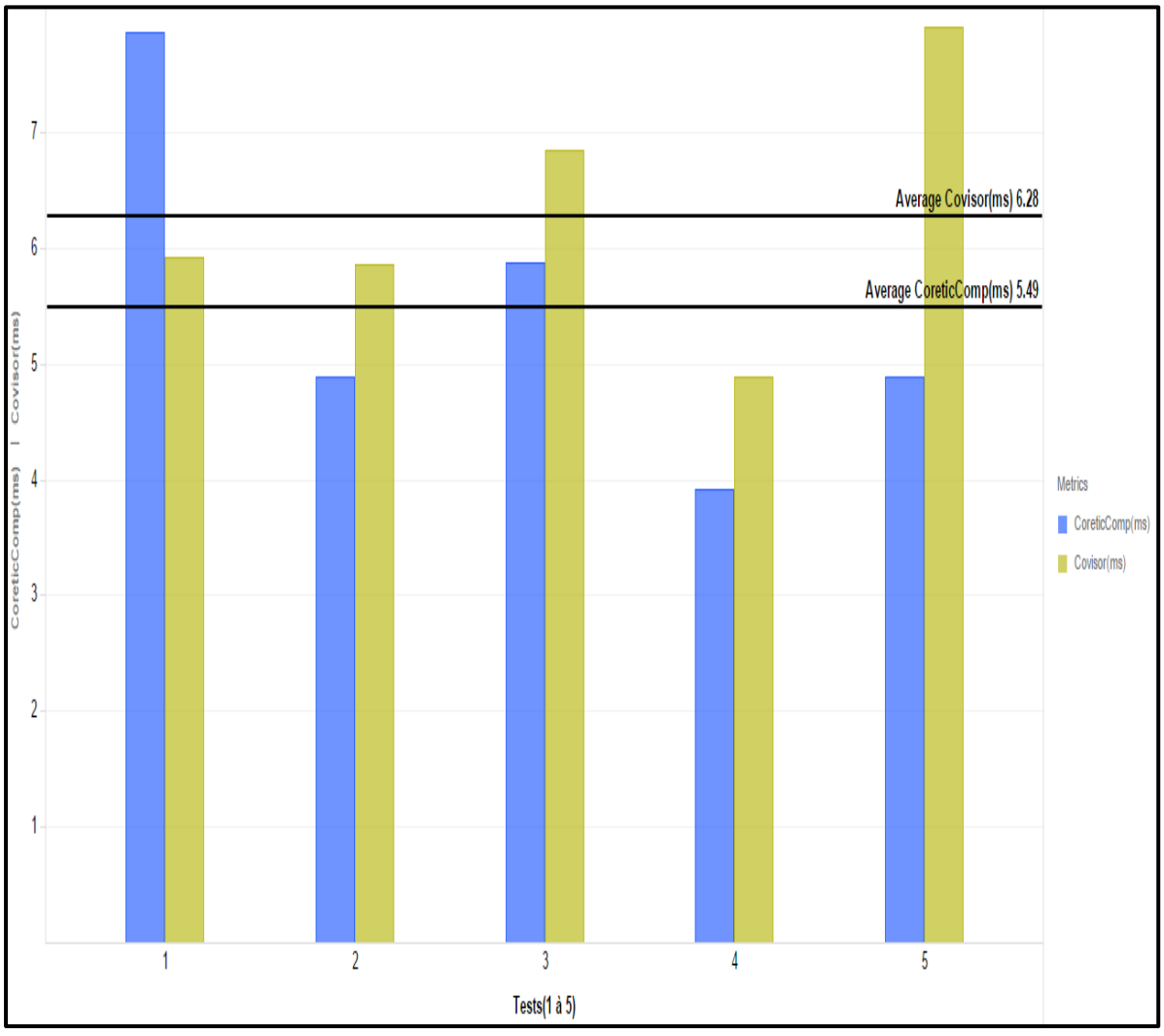


Figure 4.4 Résultat des mesures de latence en composition séquentielle pour CoreticComp et Covisor

Tableau 4.9 Tableau de comparaison de latence entre CoreticComp, Covisor en composition séquentielle

Solutions	Latences(ms)
CoreticComp	5,49
Covisor	6,28

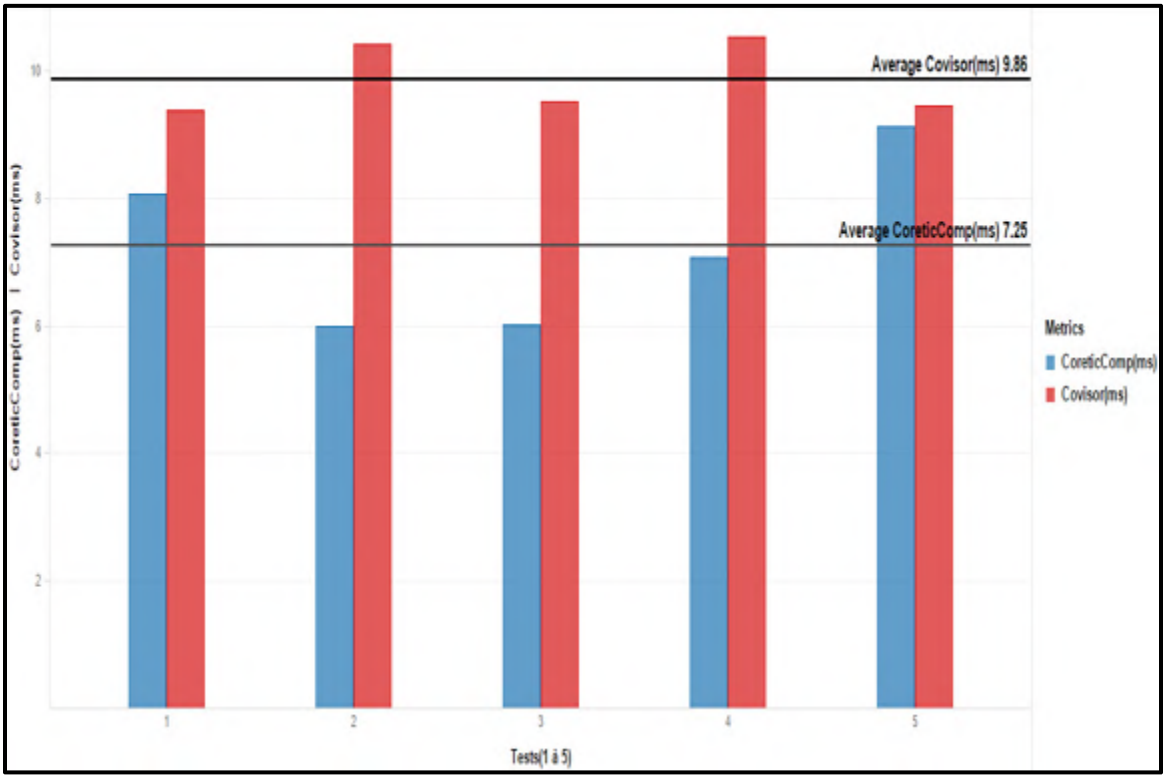


Figure 4.5 Résultat des mesures de latence en composition parallèle pour CoreticComp et Covisor

Tableau 4.10 Tableau de comparaison de latence entre CoreticComp, Covisor en composition parallèle

Solutions	Latences(ms)
CoreticComp	7,25
Covisor	9,86

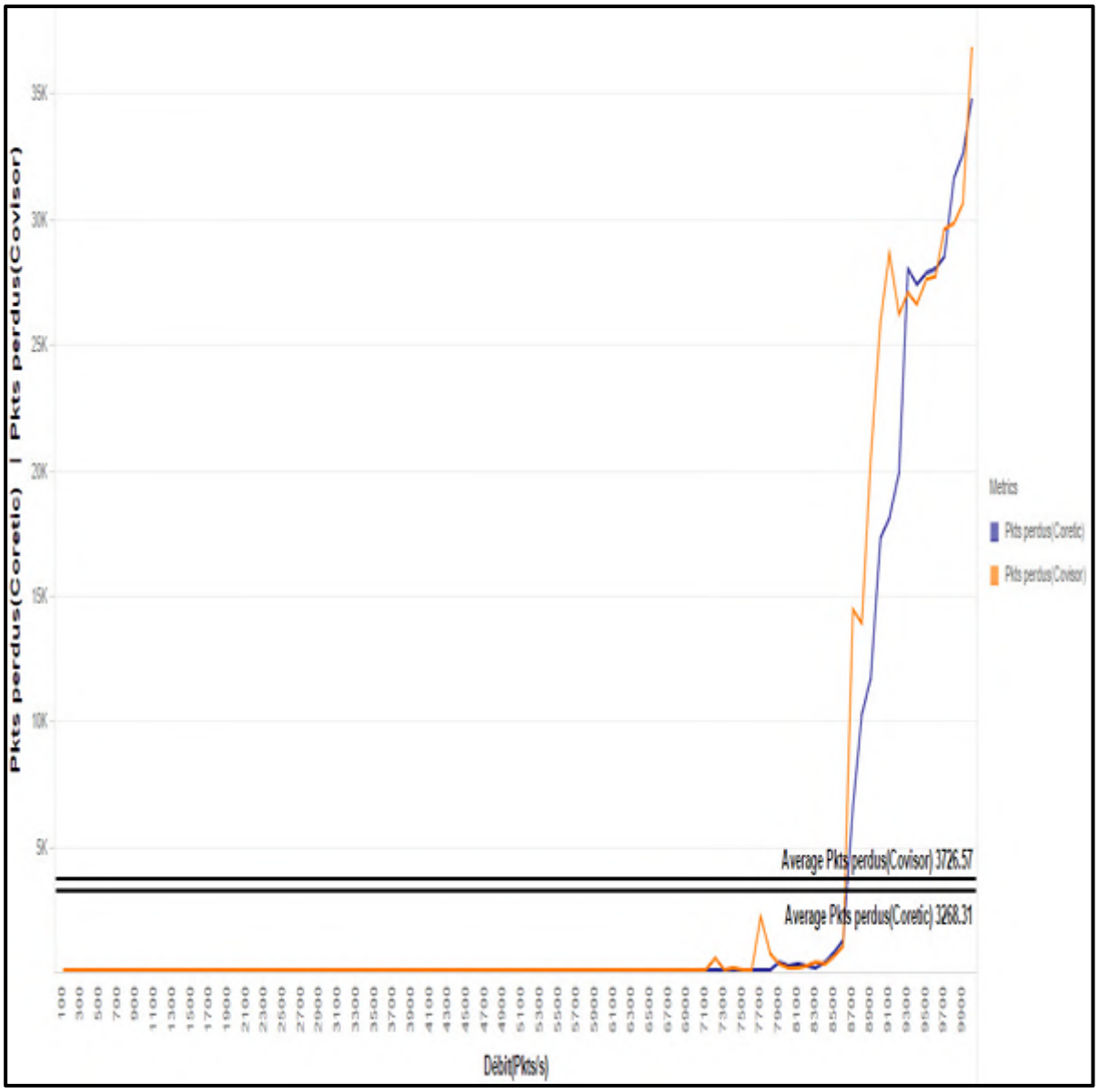


Figure 4.6 Mesure des paquets perdus en fonction du débit pour la composition séquentielle

Tableau 4.11 Moyenne des paquets perdus par débit d'injection de paquets pour la composition séquentielle

Solutions	Moyenne paquets perdu par débit d'injection de paquets (nombre de paquets)
CoreticComp	3 268
Covisor	3 726

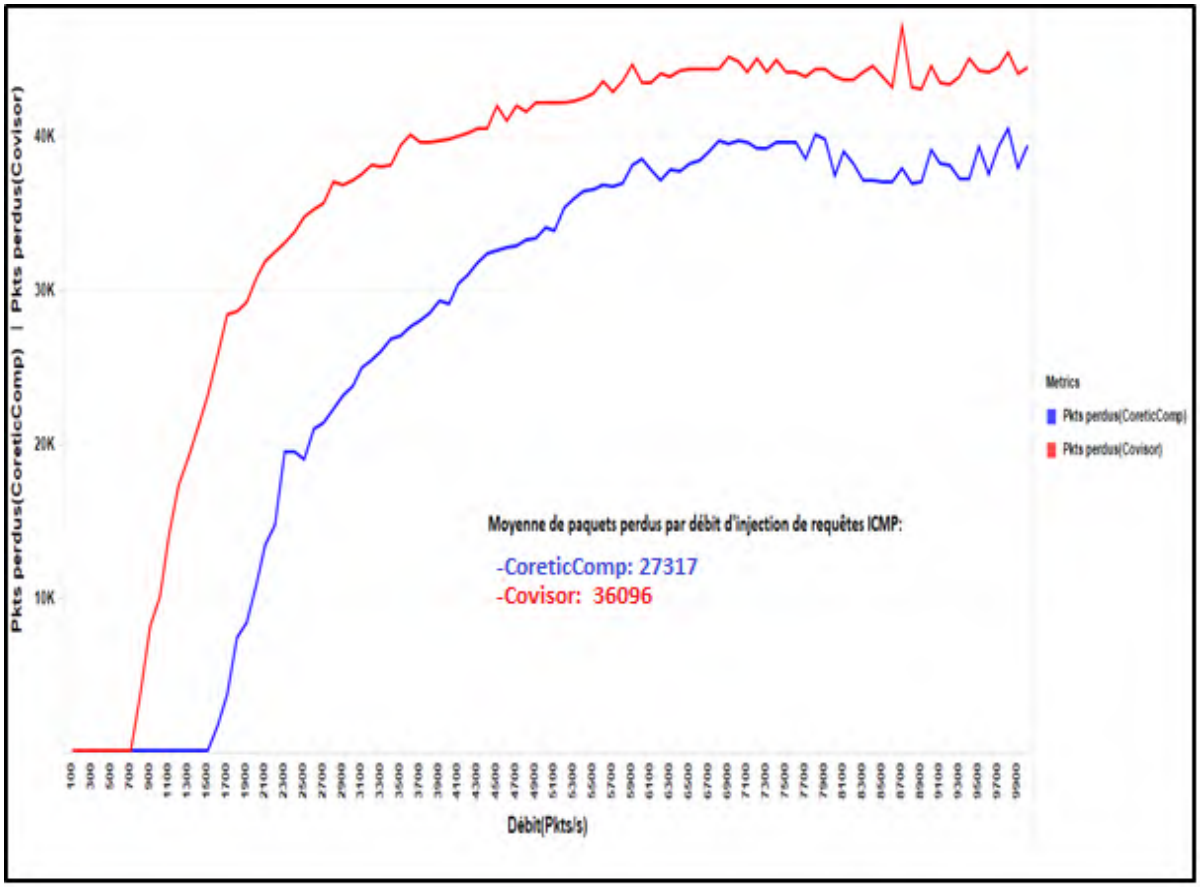


Figure 4.7 Mesure des paquets perdus en fonction du débit pour la composition parallèle

Tableau 4.12 Moyenne des paquets perdus par débit d'injection de paquets pour la composition parallèle

Solutions	Moyenne paquets perdus par débit d'injection de paquets(nombre de paquets)
CoreticComp	27 317
Covisor	36 096

4.2.3 Bande passante

Les bandes passantes ont été mesurées en envoi et en réception de données. Les deux voies de communication ont été évaluées en composition séquentielle et parallèle. Les figures 4.8, 4.9 ci-dessous montrent les comparaisons des capacités de bande passante pour les solutions CoreticComp et Covisor en composition séquentielle et parallèle. En abscisse, il y a les tests

effectués qui vont de 1 à 5. Chaque test consiste à mesurer 100 fois la bande passante. En ordonnée, il y a les capacités des bandes passantes en Mbits/s qui sont les moyennes des résultats en envoi et en réception de paquets. Le mode envoi concerne le trafic entre le client et le serveur. Quant au mode réception, concerne le trafic entre le serveur et le client. Encore une fois, il y a une augmentation de la capacité de la bande passante. Pour la composition séquentielle, la moyenne de la capacité de la bande passante est de 854,64Mbits/s pour CoreticComp et de 563,54Mbits/s pour Covisor (voir tableau 4.13). Cela fait un gain de 34,06%. Pour la composition parallèle, la moyenne de la bande passante est de 943,15 Mbits/s pour CoreticComp et de 743,05 Mbits/s (voir tableau 4.14). Cela fait une augmentation de performance de 21,21%.

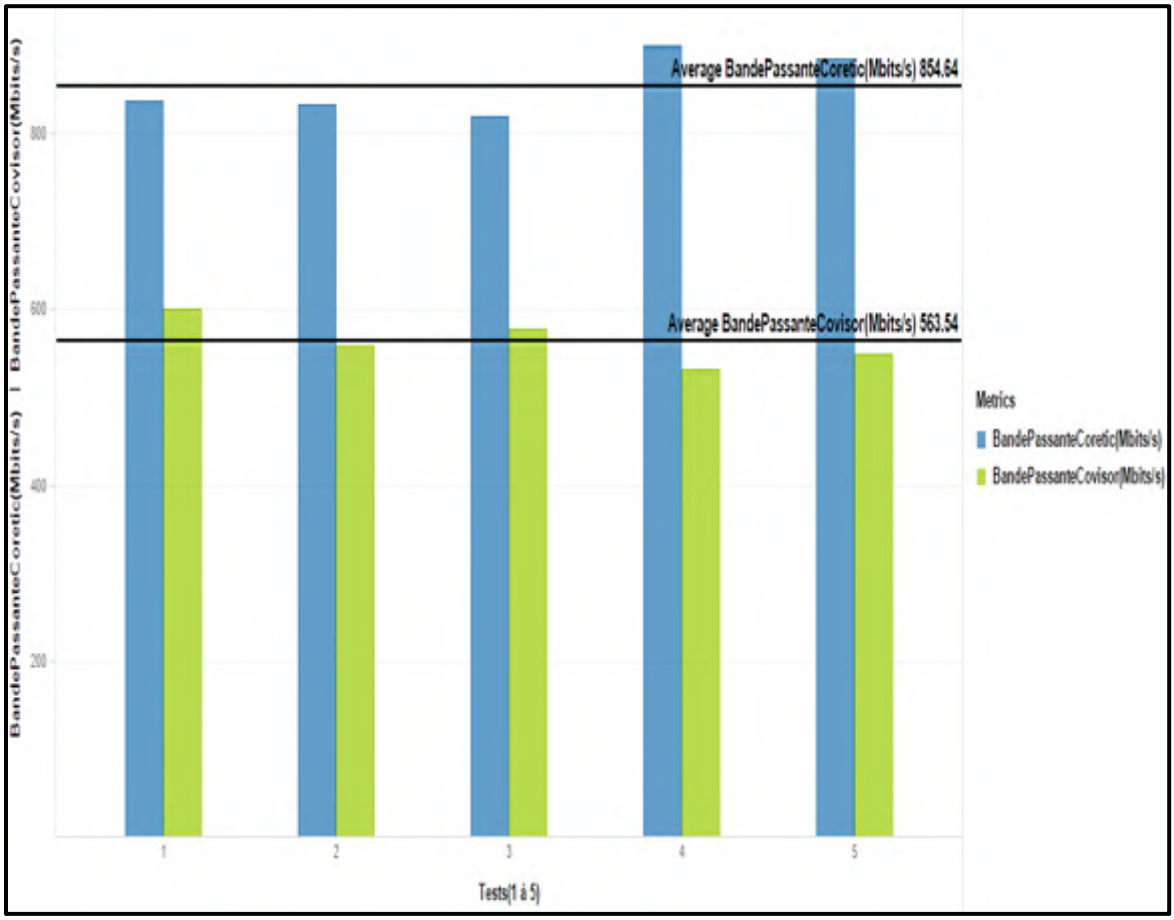


Figure 4.8 Comparaison capacité bande passante CoreticComp et Covisor en composition séquentielle

Tableau 4.13 Moyenne des capacités des bandes passantes en composition séquentielle

Solutions	Moyenne bandes passantes(Mbits/s)		Moyenne générale (Mbits/s)
	Envoi	Réception	
CoreticComp	889,69	819,60	854,64
Covisor	618,77	508,32	563,54

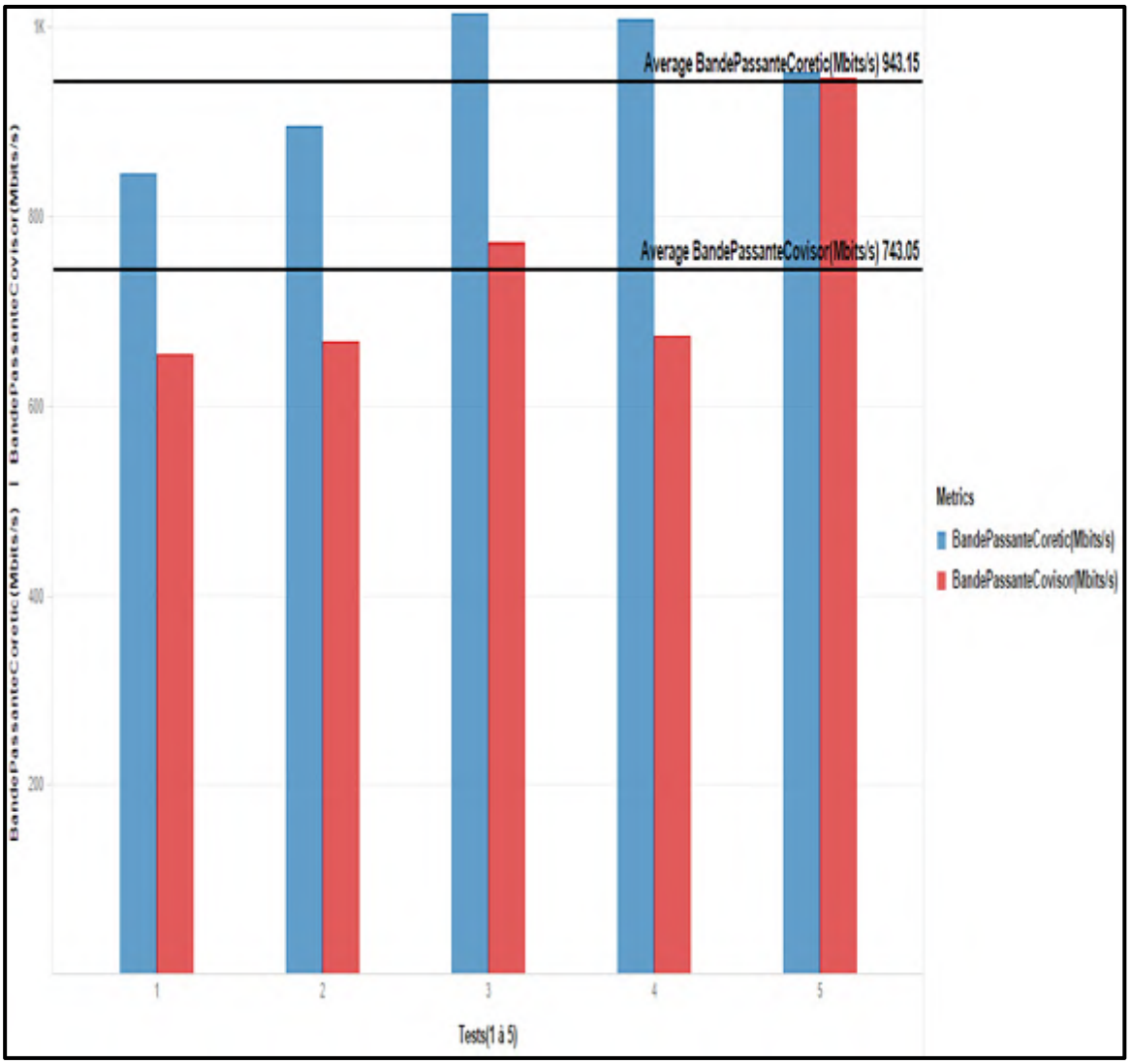


Figure 4.9 Comparaison capacité bande passante CoreticComp et Covisor en composition parallèle

Tableau 4.14 Moyenne des capacités des bandes passantes en composition parallèle

Solutions	Moyenne bandes passantes(Mbits/s)		Moyenne générale(Mbits/s)
	Envoi	Réception	
CoreticComp	950,81	935,51	943,15
Covisor	769,26	716,84	743,05

La section 4.2 ci-dessus a présenté les résultats pour les tests en composition séquentielle et parallèle. La section 4.3 qui suit présente les résultats pour l'hyperviseur de haut niveau.

4.3 Pour l'*hypervision* de contrôleurs de haut niveau

Pour pouvoir évaluer l'effet de la création de l'hyperviseur de haut niveau, les mesures furent prises avec et sans prise en compte de CoreticHip. C'est-à-dire que des évaluations ont faîtes en utilisant seulement Pyretic et en utilisant CoreticHip. Les résultats pour la latence, le débit et la bande passante seront présentés.

4.3.1 Latences

Les mesures de latence ont été prises avec et sans la présence de l'hyperviseur CoreticHip. La figure 4.10 donne les résultats de façon respective des tests avec CoreticHip et sans CoreticHip. En abscisse, il y a les tests et en ordonnée il y a les délais en milliseconde. Le nombre de tests est de 5. Comme pour les autres tests effectués, ici un test consiste à effectuer 100 dizaines de requêtes ICMP. En moyenne, il y a une latence de 537ms pour Pyretic et 532,21ms pour CoreticHip (voir tableau 4.15). Avec l'utilisation de CoreticHip, il y a une augmentation de latence qui est de 0,89%.

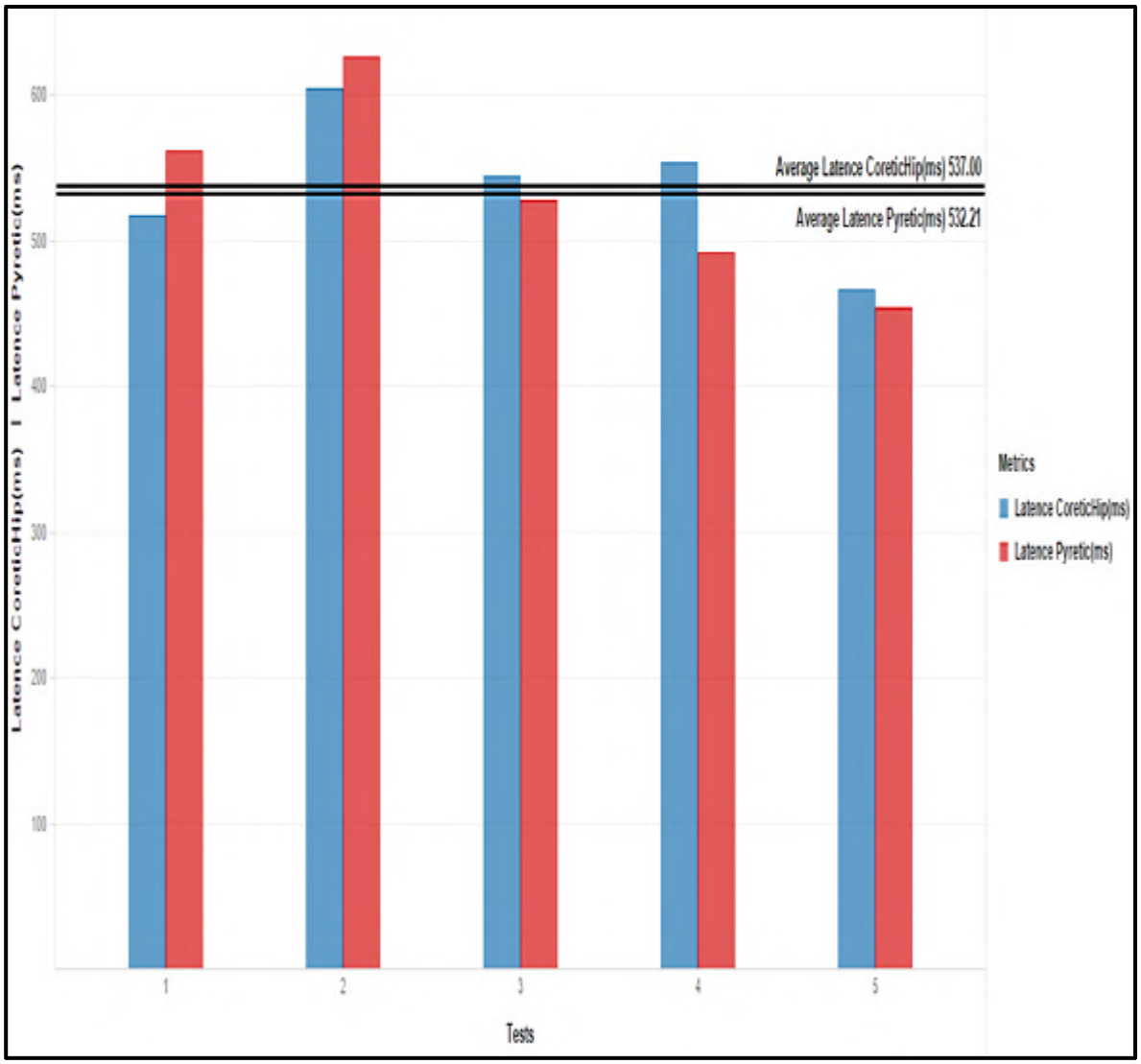


Figure 4.10 Mesure de latence avec et sans la présence de l'hyperviseur CoreticHip

Tableau 4.15 Résumé des mesures de latence avec et sans CoreticHip

Solutions	Minimale(ms)	Moyenne(ms)	Maximale(ms)
Avec CoreticHip	388,9	537	1 182
Avec Pyretic(absence de CoreticHip)	401,8	532,21	3 142

4.3.2 Débit de paquet

La figure 4.11 ci-dessous permet d'observer le nombre de paquets perdus en fonction du débit d'injection de requêtes ICMP qui vont de 2 Pkts/s à 20 Pkts/s. En moyenne, il y a environ 16 paquets perdus pour Coretic (CoreticHip) et environ 14 paquets perdus en utilisant seulement Pyretic (absence de CoreticHip). Cela génère une baisse de performance de 2%.

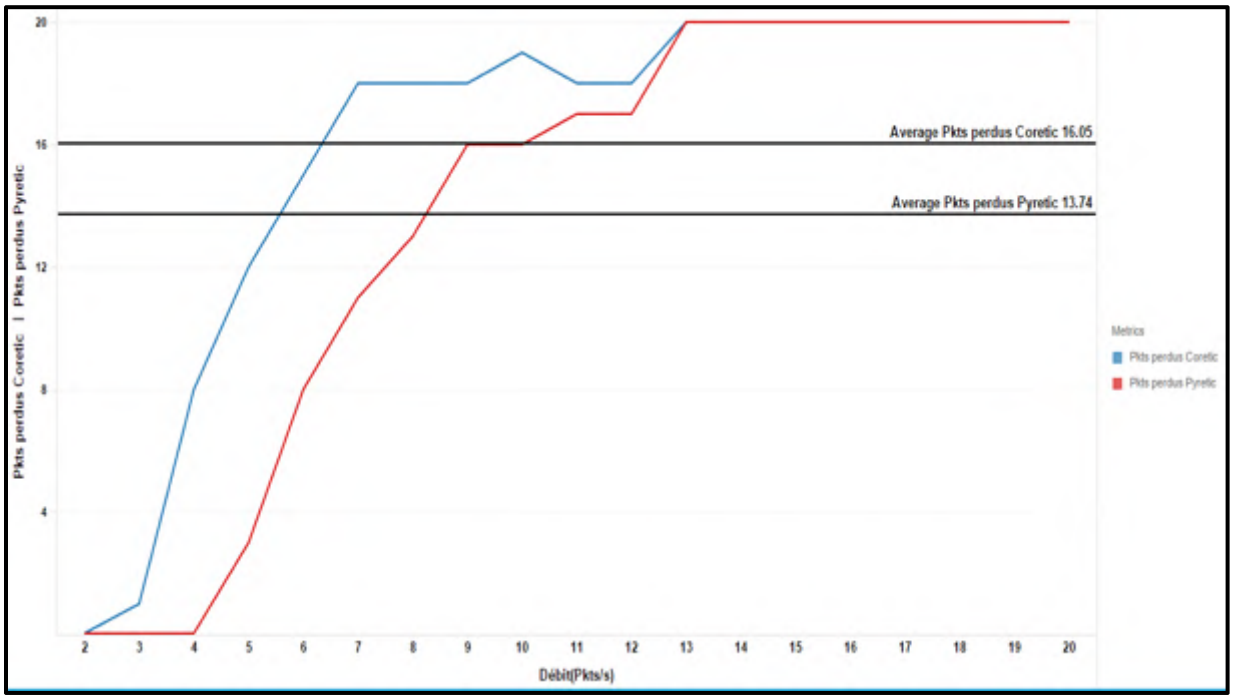


Figure 4.11 Mesure des paquets perdus en fonction du débit avec et sans CoreticHip

4.3.3 Bande passante

La figure 4.12 donne une comparaison de capacité lorsqu'il y présence de CoreticHip et lorsqu'il n'y a pas présence de l'hyperviseur. En ordonnée, il y a la capacité des bandes passante et en abscisse, il y a les tests effectués. Les mesures en ordonnée sont la moyenne entre la capacité de la bande passante en envoi et en réception. La moyenne de la capacité de la bande passante est de 3,21Mbits/s avec Pyretic seulement et de 3Mbits/s avec l'utilisation de CoreticHip (voir tableau 4.16). La baisse de performance est de 6,54%.

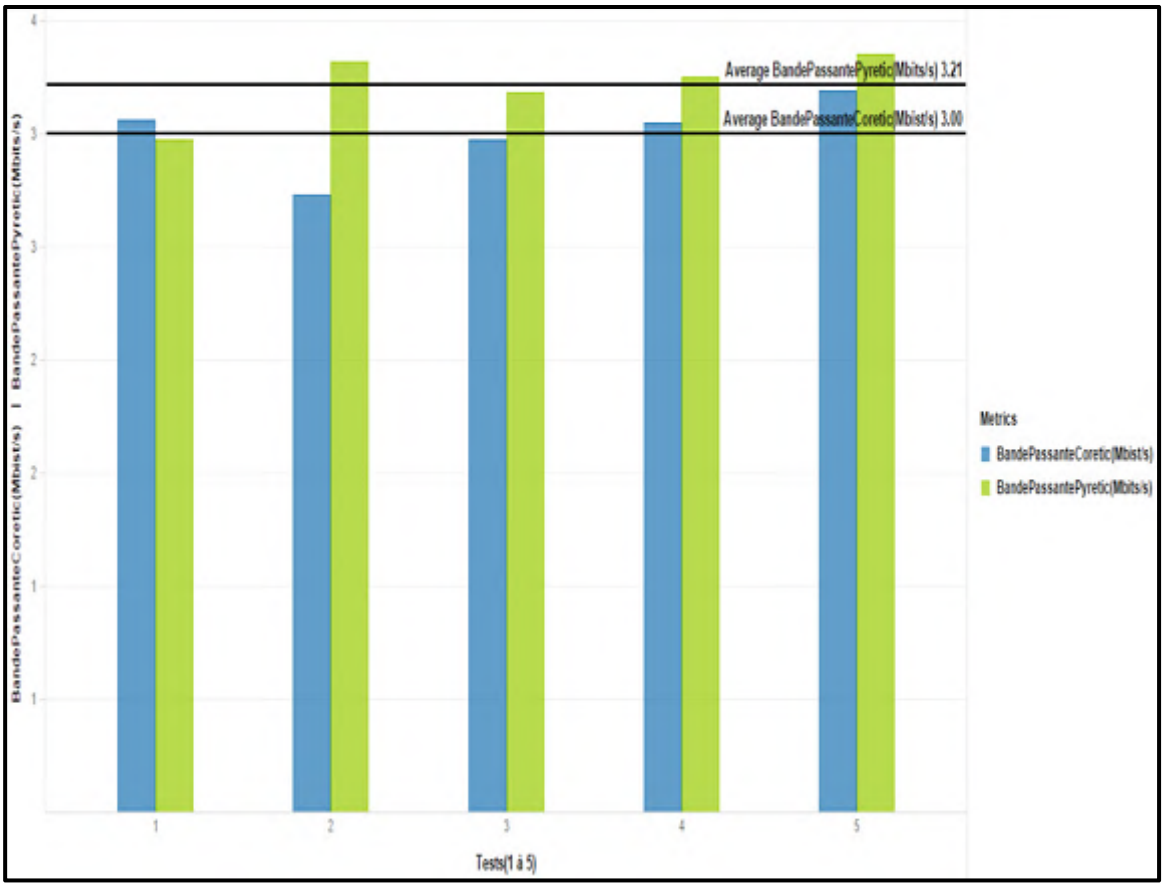


Figure 4.12 Mesure de la bande passante avec et sans CoreticHip

Tableau 4.16 Moyenne des capacités des bandes passantes avec et sans CoreticHip

Solutions	**Moyenne bandes passantes(Mbits/s)**		**Moyenne générale(Mbits/s)**
	Envoi	Réception	
Pyretic	6,21	0,22	3,11
CoreticHip	5,9	0,11	3,005

Les tests effectués avec CoreticHip montrent une baisse de performance face au scénario où il n'y a pas CoreticHip. Les raisons qui expliquent cette baisse seront vues en détail dans le la partie 4.4 qui suit. Cette partie reviendra également sur les résultats obtenus pour l'isolation et la composition.

4.4 Vérification des objectifs

Cette section fait un rappel des objectifs et au vu des résultats, montre l'atteinte des objectifs fixés.

4.4.1 Amélioration des performances

Les tests ont permis de montrer que la solution d'écriture des règles développée pour l'isolation des trafics offre de meilleures performances. Le tableau 4.6 montre un délai moyen pour la latence plus bas pour l'isolation. La latence chute de 12,40% en comparaison avec OpenVirtex et de 11,25% en comparaison avec SDNMS. Toujours au niveau de l'isolation, il y a une baisse moyenne de paquets perdus en fonction de la vitesse d'injection des requêtes ICMP (voir tableau 4.7). La capacité de la bande passante est aussi supérieure (voir tableau 4.8). Cette amélioration de performance trouve se justifie par les points suivants :

- Génération de moins d'entrées dans les tables de flux

L'isolation au niveau de CoreticIsol se fait avec les segments d'adresse IP. En faisant l'isolation avec les segments d'adresse IP, il y a une réduction considérable du nombre d'entrées dans les tables de flux. Ce qui permet au commutateur de traiter les flux beaucoup plus rapidement.

- Utilisation de plusieurs tables en raison d'une table par client et de la table principale d'isolation

L'utilisation de plusieurs tables permet de façon efficace à traiter les flux. Comme cela a déjà été expliqué, OpenVirtex utilise une seule table de flux pour faire de l'isolation. Avec plusieurs clients et avec l'utilisation des informations de la couche 2 et 3 du modèle OSI, le nombre d'entrées dans la table de flux de chacun des commutateurs est énorme. C'est pour cela que ça prend plus de temps au commutateur de trouver la règle correspondante pour traiter un certain flux. Les mêmes constats d'amélioration de performance sont observés avec les résultats observés pour la composition, car Covisor utilise également une seule table.

4.4.2 Augmentation de cohésion de tables

Pour l'isolation, la solution SDNMS permettait aux tables des clients de non seulement gérer les flux, mais aussi de supprimer les paquets, dont les adresses sources, ou de destination

appartenait à un autre client. Ce qui augmente la cohésion de la table du client. Pour améliorer ce fait, la responsabilité de suppression des paquets ne respectant pas les règles d'appartenance à un client ou à un autre est faite dans la table 0 pour CoreticIsol. C'est cette table qui est responsable de distribuer les paquets vers les tables de niveau inférieur. La cohésion de CoreticIsol est aussi supérieure à celle de OpenVirtex. OpenVirtex insère toutes les règles dans une seule table. Ceci affaiblit la cohésion. Pour la composition une amélioration de la cohésion est aussi observée. Contrairement à Covisor qui insère toutes les règles de composition dans une seule table, CoreticComp répartit les tâches des contrôleurs dans les différentes tables.

4.4.3 Réduction de couplage entre les tables

La solution de FlowBricks dans le cadre de la composition introduit la notion de table de transition. Ces tables de transition permettent de faire passer le traitement d'un paquet d'une table à l'autre. Soit une table T1 qui contient des règles pour faire du coupe-feu en supprimant les paquets ayant des adresses IP impaires. La table T1 en rencontrant un paquet ayant une adresse IP impaire va inscrire comme action la suppression du paquet en traitement et acheminer le paquet vers une table de transition T'. Cette table va supprimer le paquet identifié dans la table T1 comme un paquet ayant une adresse IP impaire. Mais pour les autres types de paquets, les redirections seront faites vers une table T2 pour appliquer les règles de commutation. Comme on le voit, l'insertion d'une table de transition fait en sorte que le couplage augmente tant au niveau de la table T1. La table T1 peut directement supprimer les paquets non souhaités que de les soumettre à la table de transition. C'est ce que fait la solution CoreticComp. Elle évite l'utilisation d'une table de transition. Il est à noter également l'économie des tables utilisées. En Openflow 1.0, en se servant de Nicira, le nombre de tables à utiliser est limité à 256. Avec CoreticHip, en évitant l'utilisation d'une table de transition, le nombre de tables à utiliser est réduit.

4.4.4 *Hypervision* des contrôleurs de haut niveau

L'objectif à atteindre était de permettre l'*hypervision* entre plusieurs plans de gestion. Il fallait aussi s'attendre aussi du fait que les performances n'en soient trop atteintes. Le composant CoreticHip est en mesure de bien gérer les flux afin d'assurer une isolation des trafics appartenant à des plateformes de programmation différentes. Une petite baisse de performance

de 0,89% est observée pour la latence. Cette baisse de performance est aussi observée pour le test de résistance à perdre les paquets. À ce niveau la baisse de performance est de 2%. Pour le test de bande passante la baisse de performance est de 6,54%. Les baisses de performance sont raisonnables dans la mesure où dans les tests n'utilisant pas CoreticHip, il n'y a que deux machines virtuelles qui sont utilisées. Mais dans le test utilisant CoreticHip, il y a non seulement la couche de l'hyperviseur qui est prise en compte, mais il y a trois machines virtuelles qui sont utilisées. Le plan de contrôle utilise une machine qui est différente du plan de gestion. En plus de la baisse de performance générée par CoreticHip, il y a aussi le fait que les paquets doivent aller du plan de contrôle au plan de gestion. Or avec le test n'utilisant pas CoreticHip, le plan de contrôle et le plan de gestion sont dans la même machine.

4.5 Conclusion

Le chapitre 4 a permis de prendre connaissance des résultats de test et d'en faire quelques analyses. Ces résultats observés montrent de meilleures performances pour les solutions d'isolation et de composition. Mais au niveau de CoreticHip, il y a une baisse de performance pour la latence, le débit des paquets et le débit de la bande passante. Cette baisse de performance et justifiée à cause de l'insertion d'un hyperviseur se situant entre le contrôleur de bas niveau (dans notre cas POX) et les contrôleurs de haut niveau (dans notre cas ici qui sont les plateformes de programmation de haut niveau). Il y a aussi dans le cas de CoreticHip l'utilisation de trois machines virtuelles. Les analyses des résultats de test ont permis par la suite d'atteindre les objectifs fixés et justifier les hypothèses émises. L'une des hypothèses suggérait de réduire le nombre d'entrée dans les flux en se servant des informations de la couche 3 pour L'isolation tout en utilisant plusieurs tables. Une autre hypothèse suggérait l'utilisation d'un mandataire sans effet sur la génération des règles pour aire de l'*hypervision* entre les contrôleurs de haut niveau. Le chapitre 4 a permis de prendre connaissance des résultats de test et de les analyser afin d'en déduire l'atteinte des objectifs. Ce rapport de maîtrise se terminera par une conclusion dans le prochain grand point qui va revenir sur les éléments clés abordés. La conclusion va aussi orienter le lecteur des autres axes de travaux qui découlent de ce travail.

CONCLUSION

Les performances dans les réseaux informatiques restent depuis des années des éléments importants. Plusieurs règles de gestion de flux s'entrecroisent. Ce qui fait que le travail des équipements comme les commutateurs est complexe. Ceci entraine une défaillance au niveau des performances. SDN, une nouvelle architecture réseau offre une solution permettant de réduire la complexité des tâches des commutateurs. Elle découple le rôle de prise de décision du rôle d'acheminement des flux. Pour acheminer les flux, les commutateurs se servent des tables de flux et des entrées qu'elles contiennent. Les performances du réseau peuvent dépendre de la façon que les tables ainsi que les entrées dans ces tables sont organisées. Les tables ainsi que les entrées sont parcourues pour traiter les flux.

Ce travail de maîtrise a permis de prendre connaissance d'une solution appelée Coretic qui à travers deux de ses trois sous solutions (CoreticIsol, CoreticComp) offre des solutions de gestion de tables de flux et d'entrées. Ces solutions, comme expliqués dans le rapport utilisent plusieurs tables. Pour l'isolation des flux de plusieurs clients, CoreticIsol se sert des informations de la couche 3 du modèle ISO. Ce qui permet de réduire considérablement le nombre d'entrées dans la table principale de gestion de flux. Il s'en suit une amélioration des performances. CoreticComp se sert des notions de cohésion pour organiser les tables de flux de telle sorte que chacune d'elle puisse faire appliquer une politique bien précise. Avec CoreticComp, la baisse de couplage entre les tables et l'économie du nombre de tables sont aussi prises en compte. Les tables de transition ne sont pas prises en compte comme ce qui se fait avec FlowBricks.

Coretic a une troisième sous solution appelée CoreticHip qui permet de faire de l'*hypervision* entre les plateformes de programmation de haut niveau en SDN. CoreticHip avec une baisse de performance justifiée offre la possibilité à plusieurs clients de travailler avec des plateformes de programmation différentes tout en ayant les flux de ces clients isolés les uns des autres.

Coretic à la suite de ce projet de maîtrise apporte les contributions suivantes :

- Solution d'écriture des règles dans les tables de flux pour l'isolation basée sur les informations de réseau de la couche 3 du modèle OSI. Cette solution aide aussi à renforcer la cohésion des tables. Cette solution est réalisée avec CoreticIsol.
- Solution de composition des règles des contrôleurs renforçant la cohésion des tables et réduisant le couplage entre elles. Il est réalisé avec CoreticComp.
- Solution d'*hypervision* des plateformes de programmation de haut niveau qui aide à l'utilisation de plusieurs plans de gestion pour un seul contrôleur. Cette solution est réalisée par CoreticHip.

Coretic apporte certes des contributions pour SDN. Ces contributions auraient été renforcées avec les points ci-dessous accomplis :

- Ajouter à CoreticComp, la composition prenant en compte la modification des paquets par une table précédente dans le cadre de la composition séquentielle.
- Évaluer CoreticIsol en fonction du nombre de réseaux virtuels.
- Avec CoreticHip, permettre l'utilisation des plateformes de programmation de haut niveau hétérogènes comme la prise en compte de Frenetic.
- Faire l'évaluation de CoreticHip avec tous les types de messages allant du contrôleur au plan de données. L'évaluation dans ce travail se limite au message de type Packtout.
- Prendre en compte IPV6.

ANNEXE I

PSEUDO-CODE DE DÉPLOIEMENT DU PLAN DE DONNÉES

```
Lire configurations dans fichier Json
Pour chaque configuration de réseau virtuel faire :
        debutMac = début adresse Mac des hosts du client
        debutIP = début adresse IP des hosts du client
        valPort1, valPort2 =  numéro de port début,  numéro de port de fin
        hostsParCommutateur = nombre de hosts du réseau virtuel par commutateur
        NombreCommutateur = 1
        TotalHosts = 0
        Tant que NombreCommutateur<6 faire :
                TotalHosts = TotalHosts + 1
                nomCommutateur = "s" + NombreCommutateur
                valeurPortActuel = valPort1
                quantiteHosts = 1
                Tant que quantiteHosts <= hostsParCommutateur faire :
                        Créer un host
                        IP de host = debutIP + TotalHosts
                        MAC de host = debutMac + TotalHosts
                        Lier un host à nomCommutateur au port valeurPortActuel
                        valeurPortActuel = valeurPortActuel + 1
                Fin tant que
                NombreCommutateur = NombreCommutateur + 1
        Fin tant que
Fin pour
Lier commutateur 1 à commutateur 2
Lier commutateur 2 à commutateur 3
Lier commutateur 3 à commutateur 4
Lier commutateur 4 à commutateur 5
```

ANNEXE II

SCRIPT POUR DÉMARRER POX ET EXÉCUTER UN CLIENT POUR ÉCRITURE DE RÈGLES

```
300  *** Start pox backend. ***
301  if not options.frontend_only:
302      try:
303          output = subprocess.check_output('echo $PYTHONPATH',shell=True).strip()
304      except:
305          print 'Error: Unable to obtain PYTHONPATH'
306          sys.exit(1)
307      poxpath = None
308      for p in output.split(':'):
309          if re.match('.*pox/?$',p):
310              poxpath = os.path.abspath(p)
311              break
312      if poxpath is None:
313          print 'Error: pox not found in PYTHONPATH'
314          sys.exit(1)
315      pox_exec = os.path.join(poxpath,'pox.py')
316      python=sys.executable
317      # TODO(josh): pipe pox_client stdout to subprocess.PIPE or
318      # other log file descriptor if necessary
319
320      if (options.isServerCont == True):
321              pox_cmd = "python %s %s of_client.pox_clientM_original %s %s %s %s" % (
322                  pox_exec,
323                  'openflow.nicira --convert-packet-in' if options.nx else '',
324                  '--use_nx' if options.nx else '',
325                  "--pipeline=%s" % options.pipeline if options.nx else '', "--initConfigTableServer=%s" % options.configTableFile, "--isServerCont")
326
327              of_client = subprocess.Popen(shlex.split(pox_cmd),
328                                           stdout=sys.stdout,
329                                           stderr=subprocess.STDOUT)
330
331      elif (options.enable_multicon == True):
332              pox_cmd = "python %s %s of_client.pox_clientOVXV2 %s %s" % (
333                  pox_exec,
334                  'openflow.nicira --convert-packet-in' if options.nx else '',
335                  '--use_nx' if options.nx else '',
336                  "--pipeline=%s" % options.pipeline if options.nx else '')
337
338              of_client = subprocess.Popen(shlex.split(pox_cmd),
339                                           stdout=sys.stdout,
340                                           stderr=subprocess.STDOUT)
341
342      else:
343              pox_cmd = "python %s %s of_client.pox_client %s %s" % (
344                  pox_exec,
345                  'openflow.nicira --convert-packet-in' if options.nx else '',
346                  '--use_nx' if options.nx else '',
347                  "--pipeline=%s" % options.pipeline if options.nx else '')
348              of_client = subprocess.Popen(shlex.split(pox_cmd),
349                                           stdout=sys.stdout,
350                                           stderr=subprocess.STDOUT)
```

ANNEXE III

PSEUDO-CODE POUR L'ÉCRITURE DES RÈGLES DANS LA TABLE 0 POUR CORETICISOL

```
R0 = créer règle OpenFlow
Spécifier table visée à 0 pour R0
Spécifier priorité de R0 à 1
Envoyer R0 au plan de données
Lire les paramètres de chaque client dans fichier Json
Pour chaque configuration des réseaux virtuels faire :
        destTable = table de destination du client dans la configuration
        R = créer règle OpenFlow
        Rendre R permanent dans les commutateurs
        Spécifier protocole IP pour R
        Spécifier adresse source IP pour R le masque d'adresse IP dans la configuration
        Spécifier adresse destination IP pour R le masque d'adresse IP dans la configuration
        Spécifier comme action la redirection vers destTable dans R
        Envoyer R au plan de données

        R' = créer règle OpenFlow
        Rendre R' permanent dans les commutateurs
        Spécifier protocole ARP pour R'
        Spécifier adresse source IP pour R' le masque d'adresse IP dans la configuration
        Spécifier adresse destination IP pour R' le masque d'adresse IP dans la configuration
        Spécifier comme action la redirection vers destTable dans R'
        Envoyer R' au plan de données

Fin pour
```

ANNEXE IV

PSEUDO-CODE POUR L'ÉCRITURE DES RÈGLES DANS LA TABLE 0 POUR SDNMS

```
R0 = créer règle OpenFlow
Spécifier table visée à 0 pour R0
Spécifier priorité de R0 à 1
Envoyer R0 au plan de données
Lire les paramètres de chaque client dans fichier Json
Pour chaque configuration de réseau virtuel des clients faire :
        nombreHosts = lire nombre de hosts dans la configuration
        tableDest = lire table de destination dans la configuration
        Pour chaque host :
                destTable = table de destination du client dans la configuration
                R1 = créer règle OpenFlow
                Rendre R1 permanent dans les commutateurs
                Spécifier protocole ARP pour R1
                Adresse MAC source de R1 = MAC du host
                Adresse MAC destination de R1 = "ff :ff :ff :ff :ff :ff"
                "inport" de R1 = port de connexion du commutateur au host
                Spécifier comme action à R1 la redirection vers destTable
                Envoyer R1 au plan de données
        Fin pour
        Pour chaque host :
                destTable = table de destination du client dans la configuration
                R2 = créer règle OpenFlow
                Rendre R2 permanent dans les commutateurs
                Spécifier protocole ARP pour R2
                Adresse destination MAC de R2 = MAC du host
                Spécifier comme action à R2 la redirection vers destTable
                Envoyer R2 au plan de données

                R3 = créer règle OpenFlow
                Rendre R3 permanent dans les commutateurs
                Spécifier protocole IP pour R3
                Adresse destination MAC de R3 = MAC du host
                Spécifier comme action à R3 la redirection vers destTable
                Envoyer R3 au plan de données
        Fin pour
Fin pour
```

ANNEXE V

PSEUDO-CODE POUR LA CONFIGURATION DE LA TABLE 0 POUR CORETICCOMP EN COMPOSITION SÉQUENTIELLE

```
R0 – créer règle OpenFlow
Spécifier table visée à 0 pour R0
Spécifier priorité de R0 à 1
Envoyer R0 au plan de données
Lire les paramètres de pare-feu dans fichier Json

destTable – table de redirection pour module suivant
listesAdresseIP – lire les adresses IP à partir des paramètres
Pour chaque adresse IP Impaire IPimpaire dans listesAdresseIP faire :
        R = créer règle OpenFlow
        Rendre R permanent dans les commutateurs
        Spécifier table visée à 0 pour R
        Spécifier protocole IP pour R
        Source IP de R = IPimpaire
        Spécifier comme action la suppression du paquet
        Envoyer R au plan de données

        R' – créer règle OpenFlow
        Rendre R' permanent dans les commutateurs
        Spécifier table visée à 0 pour R'
        Spécifier protocole IP pour R'
        Destination IP de R' = IPimpaire
        Spécifier comme action la suppression du paquet
        Envoyer R' au plan de données
        R'' – créer règle OpenFlow
        Rendre R'' permanent dans les commutateurs
        Spécifier table visée à 0 pour R''
        Spécifier protocole ARP pour R''
        Source IP de R'' – IPimpaire
        Spécifier comme action la suppression du paquet
        Envoyer R'' au plan de données

        R''' – créer règle OpenFlow
        Rendre R''' permanent dans les commutateurs
        Spécifier table visée à 0 pour R'''
        Spécifier protocole ARP pour R'''
        Destination IP de R''' – IPimpaire
        Spécifier comme action la suppression du paquet
        Envoyer R''' au plan de données
Fin pour
```

RversT1 = créer règle OpenFlow
Spécifier table visée à 0 pour RversT1
Spécifier priorité de RversT1 à 10
Spécifier comme action pour RversT1 la redirection vers la table 1
Envoyer RversT1 au plan de données

ANNEXE VI

MODULE DE GESTION DE FLUX UTILISÉ SANS CORETICHIP

```
from pyretic.lib.corelib import *
from pyretic.lib.std import *
from pyretic.lib.query import *
from pyretic.core import packet
from pyretic.core.network import IPAddr, EthAddr
adresseReseau = '10.0.1.0/24'
mac1 = EthAddr('00:00:00:00:01:01')
mac2 = EthAddr('00:00:00:00:01:32')
mac3 = EthAddr('FF:FF:FF:FF:FF:FF')
ip1 = '10.0.1.1'
ip2 = '10.0.1.50'
politiqueFonwarde = ((match(srcip=ip1, dstip=ip2, switch=1, srcmac=mac1, dstmac=mac3, inport=1, ethtype=packet.ARP) >> fwd(1) + fwd(2) + fwd(3) + fwd(4) + fwd(5) + fwd(6) + fwd(7) + fwd(8) +
fwd(9) + fwd(10) + fwd(21)) + (match(srcip=ip2, dstip=ip1, switch=1, srcmac=mac2, dstmac=mac3, inport=21, ethtype=packet.ARP) >> fwd(1) + fwd(2) + fwd(3) + fwd(4) + fwd(5) + fwd(6) + fwd(7) +
fwd(8) + fwd(9) + fwd(10)) + (match(srcip=ip1, dstip=ip2, switch=2, srcmac=mac1, dstmac=mac3, inport=21, ethtype=packet.ARP) >> fwd(1) + fwd(2) + fwd(3) + fwd(4) + fwd(5) + fwd(6) + fwd(7) +
fwd(8) + fwd(9) + fwd(10) + fwd(22)) + (match(srcip=ip2, dstip=ip1, switch=2, srcmac=mac2, dstmac=mac3, inport=22, ethtype=packet.ARP) >> fwd(1) + fwd(2) + fwd(3) + fwd(4) + fwd(5) + fwd(6) +
fwd(7) + fwd(8) + fwd(9) + fwd(10) + fwd(21)) + (match(srcip=ip1, dstip=ip2, switch=3, srcmac=mac1, dstmac=mac3, inport=21, ethtype=packet.ARP) >> fwd(1) + fwd(2) + fwd(3) + fwd(4) + fwd(5) +
fwd(6) + fwd(7) + fwd(8) + fwd(9) + fwd(10) + fwd(22)) + (match(srcip=ip2, dstip=ip1, switch=3, srcmac=mac2, dstmac=mac3, inport=22, ethtype=packet.ARP) >> fwd(1) + fwd(2) + fwd(3) + fwd(4) +
fwd(5) + fwd(6) + fwd(7) + fwd(8) + fwd(9) + fwd(10) + fwd(21)) + (match(srcip=ip1, dstip=ip2, switch=4, srcmac=mac1, dstmac=mac3, inport=21, ethtype=packet.ARP) >> fwd(1) + fwd(2) + fwd(3) +
fwd(4) + fwd(5) + fwd(6) + fwd(7) + fwd(8) + fwd(9) + fwd(10) + fwd(22)) + (match(srcip=ip2, dstip=ip1, switch=4, srcmac=mac2, dstmac=mac3, inport=22, ethtype=packet.ARP) >> fwd(1) + fwd(2) +
fwd(3) + fwd(4) + fwd(5) + fwd(6) + fwd(7) + fwd(8) + fwd(9) + fwd(10) + fwd(21)) + (match(srcip=ip1, dstip=ip2, switch=5, srcmac=mac1, dstmac=mac3, inport=21, ethtype=packet.ARP) >> fwd(1) +
fwd(2) + fwd(3) + fwd(4) + fwd(5) + fwd(6) + fwd(7) + fwd(8) + fwd(9) + fwd(10)) + (match(srcip=ip2, dstip=ip1, switch=5, srcmac=mac2, dstmac=mac3, inport=10, ethtype=packet.ARP) >> fwd(1) +
fwd(2) + fwd(3) + fwd(4) + fwd(5) + fwd(6) + fwd(7) + fwd(8) + fwd(9) + fwd(10) + fwd(21)) + (match(srcip=ip1, dstip=ip2, switch=1, srcmac=mac1, dstmac=mac2, inport=1, ethtype=packet.ARP) >>
fwd(21)) + (match(srcip=ip2, dstip=ip1, switch=1, srcmac=mac2, dstmac=mac1, inport=21, ethtype=packet.ARP) >> fwd(1)) + (match(srcip=ip1, dstip=ip2, switch=2, srcmac=mac1, dstmac=mac2,
inport=21, ethtype=packet.ARP) >> fwd(22)) + (match(srcip=ip2, dstip=ip1, switch=2, srcmac=mac2, dstmac=mac1, inport=22, ethtype=packet.ARP) >> fwd(21)) + (match(srcip=ip1, dstip=ip2, switch=3,
srcmac=mac1, dstmac=mac2, inport=21, ethtype=packet.ARP) >> fwd(22)) + (match(srcip=ip2, dstip=ip1, switch=3, srcmac=mac2, dstmac=mac1, inport=22, ethtype=packet.ARP) >> fwd(21)) +
                (match(srcip=ip1, dstip=ip2, switch=4, srcmac=mac1, dstmac=mac2, inport=21, ethtype=packet.ARP) >> fwd(22)) + (match(srcip=ip2, dstip=ip1, switch=4, srcmac=mac2,
dstmac=mac1, inport=22, ethtype=packet.ARP) >> fwd(21)) + (match(srcip=ip1, dstip=ip2, switch=5, srcmac=mac1, dstmac=mac2, inport=21, ethtype=packet.ARP) >> fwd(10)) + (match(srcip=ip2,
dstip=ip1, switch=5, srcmac=mac2, dstmac=mac1, inport=10, ethtype=packet.ARP) >> fwd(21)) + (match(srcip=ip1, dstip=ip2, switch=1, srcmac=mac1, dstmac=mac2, inport=1, ethtype=packet.IPV4) >>
fwd(21)) + (match(srcip=ip2, dstip=ip1, switch=1, srcmac=mac2, dstmac=mac1, inport=21, ethtype=packet.IPV4) >> fwd(1)) + (match(srcip=ip1, dstip=ip2, switch=2, srcmac=mac1, dstmac=mac2,
inport=21, ethtype=packet.IPV4) >> fwd(22)) + (match(srcip=ip2, dstip=ip1, switch=2, srcmac=mac2, dstmac=mac1, inport=22, ethtype=packet.IPV4) >> fwd(21)) + (match(srcip=ip1, dstip=ip2,
switch=3, srcmac=mac1, dstmac=mac2, inport=21, ethtype=packet.IPV4) >> fwd(22)) + (match(srcip=ip2, dstip=ip1, switch=3, srcmac=mac2, dstmac=mac1, inport=22, ethtype=packet.IPV4) >> fwd(21)) +
(match(srcip=ip1, dstip=ip2, switch=4, srcmac=mac1, dstmac=mac2, inport=21, ethtype=packet.IPV4) >> fwd(22)) + (match(srcip=ip2, dstip=ip1, switch=4, srcmac=mac2, dstmac=mac1, inport=22,
ethtype=packet.IPV4) >> fwd(21)) + (match(srcip=ip1, dstip=ip2, switch=5, srcmac=mac1, dstmac=mac2, inport=21, ethtype=packet.IPV4) >> fwd(10)) + (match(srcip=ip2, dstip=ip1, switch=5,
srcmac=mac2, dstmac=mac1, inport=10, ethtype=packet.IPV4) >> fwd(21)) )
regle = if_(match(dstip=IPPrefix(adresseReseau),srcip=IPPrefix(adresseReseau)),politiqueFonwarde,drop)
def main():
    return regle
```

ANNEXE VII

MODULE DE GESTION DE FLUX UTILISÉ AVEC CORETICHIP

```
from pyretic.lib.corelib import *
from pyretic.lib.std import *
from pyretic.lib.query import *
from pyretic.core import packet
from pyretic.core.network import IPAddr, EthAddr

adresseReseau = IPPrefix('10.0.1.0/24')
mac1 = EthAddr('00:00:00:00:01:01')
mac2 = EthAddr('00:00:00:00:01:32')
mac3 = EthAddr('ff:ff:ff:ff:ff:ff')
ip1 = IPAddr('10.0.1.1')
ip2 = IPAddr('10.0.1.50')

politiqueForwarde = ( (match(srcip=ip1, dstip=ip2, switch=1, srcmac=mac1, dstmac=mac3, inport=1, ethtype=packet.ARP) >> fwd(1) + fwd(2) + fwd(3) + fwd(4) + fwd(5) + fwd(6) + fwd(7) + fwd(8) +
fwd(9) + fwd(14) + fwd(21)) + (match(srcip=ip2, dstip=ip1, switch=1, srcmac=mac2, dstmac=mac3, inport=21, ethtype=packet.ARP) >> fwd(1) + fwd(2) + fwd(3) + fwd(4) + fwd(5) + fwd(6) + fwd(7) +
fwd(8) + fwd(9) + fwd(18)) + (match(srcip=ip1, dstip=ip2, switch=2, srcmac=mac1, dstmac=mac3, inport=21, ethtype=packet.ARP) >> fwd(1) + fwd(2) + fwd(3) + fwd(4) + fwd(5) + fwd(6) + fwd(7) +
fwd(8) + fwd(9) + fwd(18) + fwd(22)) + (match(srcip=ip2, dstip=ip1, switch=2, srcmac=mac2, dstmac=mac3, inport=22, ethtype=packet.ARP) >> fwd(1) + fwd(2) + fwd(3) + fwd(4) + fwd(5) + fwd(6) +
fwd(7) + fwd(8) + fwd(9) + fwd(18) + fwd(21)) + (match(srcip=ip1, dstip=ip2, switch=3, srcmac=mac1, dstmac=mac3, inport=21, ethtype=packet.ARP) >> fwd(1) + fwd(2) + fwd(3) + fwd(4) + fwd(5) +
fwd(6) + fwd(7) + fwd(8) + fwd(9) + fwd(18) + fwd(22)) + (match(srcip=ip2, dstip=ip1, switch=3, srcmac=mac2, dstmac=mac3, inport=22, ethtype=packet.ARP) >> fwd(1) + fwd(2) + fwd(3) + fwd(4) +
fwd(5) + fwd(6) + fwd(7) + fwd(8) + fwd(9) + fwd(18) + fwd(21)) + (match(srcip=ip1, dstip=ip2, switch=4, srcmac=mac1, dstmac=mac3, inport=21, ethtype=packet.ARP) >> fwd(1) + fwd(2) + fwd(3) +
fwd(4) + fwd(5) + fwd(6) + fwd(7) + fwd(8) + fwd(9) + fwd(18) + fwd(22)) + (match(srcip=ip2, dstip=ip1, switch=4, srcmac=mac2, dstmac=mac3, inport=22, ethtype=packet.ARP) >> fwd(1) + fwd(2) +
fwd(3) + fwd(4) + fwd(5) + fwd(6) + fwd(7) + fwd(8) + fwd(9) + fwd(18) + fwd(21)) + (match(srcip=ip1, dstip=ip2, switch=5, srcmac=mac1, dstmac=mac3, inport=21, ethtype=packet.ARP) >> fwd(1) +
fwd(2) + fwd(3) + fwd(4) + fwd(5) + fwd(6) + fwd(7) + fwd(8) + fwd(9) + fwd(18)) + (match(srcip=ip2, dstip=ip1, switch=5, srcmac=mac2, dstmac=mac3, inport=18, ethtype=packet.ARP) >> fwd(1) +
fwd(2) + fwd(3) + fwd(4) + fwd(5) + fwd(6) + fwd(7) + fwd(8) + fwd(9) + fwd(21)) + (match(srcip=ip1, dstip=ip2, switch=1, srcmac=mac1, dstmac=mac2, inport=1, ethtype=packet.ARP) >>
fwd(21)) + (match(srcip=ip2, dstip=ip1, switch=1, srcmac=mac2, dstmac=mac1, inport=21, ethtype=packet.ARP) >> fwd(1)) + (match(srcip=ip1, dstip=ip2, switch=1, srcmac=mac1, dstmac=mac2,
inport=21, ethtype=packet.ARP) >> fwd(22)) + (match(srcip=ip2, dstip=ip1, switch=1, srcmac=mac2, dstmac=mac1, inport=22, ethtype=packet.ARP) >> fwd(21)) + (match(srcip=ip1, dstip=ip2, switch=1,
srcmac=mac1, dstmac=mac2, inport=21, ethtype=packet.ARP) >> fwd(22)) + (match(srcip=ip2, dstip=ip1, switch=1, srcmac=mac2, dstmac=mac1, inport=22, ethtype=packet.ARP) >> fwd(21)) +
                (match(srcip=ip1, dstip=ip2, switch=4, srcmac=mac1, dstmac=mac2, inport=21, ethtype=packet.ARP) >> fwd(22)) + (match(srcip=ip2, dstip=ip1, switch=4, srcmac=mac2,
dstmac=mac1, inport=22, ethtype=packet.ARP) >> fwd(21)) + (match(srcip=ip1, dstip=ip2, switch=5, srcmac=mac1, dstmac=mac2, inport=21, ethtype=packet.ARP) >> fwd(18)) + (match(srcip=ip2,
dstip=ip1, switch=5, srcmac=mac2, dstmac=mac1, inport=18, ethtype=packet.ARP) >> fwd(21)) + (match(srcip=ip1, dstip=ip2, switch=1, srcmac=mac1, dstmac=mac2, inport=1, ethtype=packet.IPV4) >>
fwd(21)) + (match(srcip=ip2, dstip=ip1, switch=1, srcmac=mac2, dstmac=mac1, inport=21, ethtype=packet.IPV4) >> fwd(1)) + (match(srcip=ip1, dstip=ip2, switch=1, srcmac=mac1, dstmac=mac2,
inport=21, ethtype=packet.IPV4) >> fwd(22)) + (match(srcip=ip2, dstip=ip1, switch=1, srcmac=mac2, dstmac=mac1, inport=22, ethtype=packet.IPV4) >> fwd(21)) + (match(srcip=ip1, dstip=ip2,
switch=3, srcmac=mac1, dstmac=mac2, inport=21, ethtype=packet.IPV4) >> fwd(22)) + (match(srcip=ip2, dstip=ip1, switch=3, srcmac=mac2, dstmac=mac1, inport=22, ethtype=packet.IPV4) >> fwd(21)) +
(match(srcip=ip1, dstip=ip2, switch=4, srcmac=mac1, dstmac=mac2, inport=21, ethtype=packet.IPV4) >> fwd(22)) + (match(srcip=ip2, dstip=ip1, switch=4, srcmac=mac2, dstmac=mac1, inport=22,
ethtype=packet.IPV4) >> fwd(21)) + (match(srcip=ip1, dstip=ip2, switch=5, srcmac=mac1, dstmac=mac2, inport=21, ethtype=packet.IPV4) >> fwd(18)) + (match(srcip=ip2, dstip=ip1, switch=5,
srcmac=mac2, dstmac=mac1, inport=18, ethtype=packet.IPV4) >> fwd(21)) )

def main():
    return politiqueForwarde
```

ANNEXE VIII

PSEUDO-CODE D'ÉCRITURE DES RÈGLES POUR OPENVIRTEX

R0 = créer règle OpenFlow
Spécifier table visée à 0 pour R0
Spécifier priorité de R0 à 1
Envoyer R0 au plan de données
Lire les paramètres de chaque client dans fichier Json
Pour chaque configuration de réseau virtuel faire :
    nombreHosts = lire nombre de hosts dans la configuration du réseau virtuel
    Pour host1 allant $host_i$ à $host_{nombreHosts-1}$ faire :
        Pour host2 allant de $host_{i+1}$ à $host_{nombreHosts}$ faire :
            IP1 = IP de host1
            IP2 = IP de host2
            MAC1 = MAC host1
            MAC2 = MAC host2

            R = créer règle OpenFlow
            Rendre R permanent dans les commutateurs
            Spécifier protocole ARP pour R
            Table de destination de R = 0
            IP source de R = IP1
            IP destination R = IP2
            MAC source de R = MAC1
            MAC destination de R = "ff:ff:ff:ff:ff:ff"
            "inport" de R = port de connexion du commutateur à host1
            Spécifier comme action à R la redirection vers tous les ports
            Envoyer R au plan de données
            R' = créer règle OpenFlow
            Rendre R' permanent dans les commutateurs
            Spécifier protocole ARP pour R'
            Table de destination de R' = 0
            IP source de R' = IP2
            IP destination R' = IP1
            MAC source de R' = MAC2
            MAC destination de R' = "ff:ff:ff:ff:ff:ff"
            "inport" de R' = port de connexion du commutateur au host2
            Spécifier comme action à R' la redirection vers tous les ports
            Envoyer R' au plan de données
            R'' = créer règle OpenFlow
            Rendre R'' permanent dans les commutateurs
            Spécifier protocole ARP pour R''
            Table de destination de R'' = 0
            IP source de R'' = IP1
            IP destination R'' = IP2

MAC source de R'' = MAC1
MAC destination de R'' = MAC2
"inport" de R'' = port de connexion du commutateur à host1
Spécifier comme action à R'' la redirection vers le port de sortie de host2
Envoyer R'' au plan de données

R''' = créer règle OpenFlow
Rendre R''' permanent dans les commutateurs
Spécifier protocole ARP pour R'''
Table de destination de R''' = 0
IP source de R''' = IP2
IP destination R''' = IP1
MAC source de R''' = MAC2
MAC destination de R''' = MAC1
"inport" de R''' = port de connexion du commutateur au host2
Spécifier comme action à R''' la redirection vers le port de sortie de host1
Envoyer R''' au plan de données
R'''' = créer règle OpenFlow
Rendre R'''' permanent dans les commutateurs
Spécifier protocole IP pour R''''
Table de destination de R'''' = 0
IP source de R'''' = IP1
IP destination R'''' = IP2
MAC source de R'''' = MAC1
MAC destination de R'''' = MAC2
"inport" de R'''' = port de connexion du commutateur à host1
Spécifier comme action à R'''' la redirection vers le port de sortie de host2
Envoyer R'''' au plan de données

R''''' = créer règle OpenFlow
Rendre R''''' permanent dans les commutateurs
Spécifier protocole IP pour R'''''
Table de destination de R''''' = 0
IP source de R''''' = IP2
IP destination R''''' = IP1
MAC source de R''''' = MAC2
MAC destination de R''''' = MAC1
"inport" de R''''' = port de connexion du commutateur au host2
Spécifier comme action à R''''' la redirection vers le port de sortie de host1
Envoyer R''''' au plan de données

Fin pour
Fin pour
Fin pour

ANNEXE IX

PSEUDO-CODE D'ÉCRITURE DES RÈGLES POUR COVISOR POUR LA COMPOSITION SÉQUENTIELLE

```
R0 = créer règle OpenFlow
Spécifier table visée à 0 pour R0
Spécifier priorité de R0 à 1
Envoyer R0 au plan de données
Lire les paramètres de chaque client dans fichier Json
listeDesHosts = Lire la liste des hosts dans les paramètres

Pour chaque host1 dans listeDesHosts faire :
        Si IP de host1 est paire alors :
                Pour host2 dans listeDesHosts faire :
                        Si IP de host2 est impaire alors :
                                IP1 = IP de host1
                                IP2 = IP de host2
                                MAC1 = MAC host1
                                MAC2 = MAC host2

                                R = créer règle OpenFlow
                                Rendre R permanent dans les commutateurs
                                Spécifier protocole ARP pour R
                                Table de destination de R = 0
                                IP source de R = IP1
                                IP destination R = IP2
                                MAC source de R = MAC1
                                MAC destination de R = "ff:ff:ff:ff:ff:ff"
                                "inport" de R = port de connexion du commutateur à host1
                                Spécifier comme action à R la suppression du paquet
                                Envoyer R au plan de données
                                R' – créer règle OpenFlow
                                Rendre R' permanent dans les commutateurs
                                Spécifier protocole ARP pour R'
                                Table de destination de R' = 0
                                IP source de R' = IP2
                                IP destination R' = IP1
                                MAC source de R' = MAC2
                                MAC destination de R' = "ff:ff:ff:ff:ff:ff"
                                "inport" de R' = port de connexion du commutateur au host2
                                Spécifier comme action à R' la suppression du paquet
                                Envoyer R' au plan de données
```

R'' = créer règle OpenFlow
Rendre R'' permanent dans les commutateurs
Spécifier protocole ARP pour R''
Table de destination de R'' = 0
IP source de R'' = IP1
IP destination R'' = IP2
MAC source de R'' = MAC1
MAC destination de R'' = MAC2
"inport" de R'' = port de connexion du commutateur à host1
Spécifier comme action à R'' la suppression du paquet
Envoyer R'' au plan de données

R''' = créer règle OpenFlow
Rendre R''' permanent dans les commutateurs
Spécifier protocole ARP pour R'''
Table de destination de R''' = 0
IP source de R''' = IP2
IP destination R''' = IP1
MAC source de R''' = MAC2
MAC destination de R''' = MAC1
"inport" de R''' = port de connexion du commutateur au host2
Spécifier comme action à R''' la suppression du paquet
Envoyer R''' au plan de données

R'''' = créer règle OpenFlow
Rendre R'''' permanent dans les commutateurs
Spécifier protocole IP pour R''''
Table de destination de R'''' = 0
IP source de R'''' = IP1
IP destination R'''' = IP2
MAC source de R'''' = MAC1
MAC destination de R'''' = MAC2
"inport" de R'''' = port de connexion du commutateur à host1
Spécifier comme action à R'''' la suppression du paquet
Envoyer R'''' au plan de données

R''''' = créer règle OpenFlow
Rendre R''''' permanent dans les commutateurs
Spécifier protocole IP pour R'''''
Table de destination de R''''' = 0
IP source de R''''' = IP2
IP destination R''''' = IP1
MAC source de R''''' = MAC2
MAC destination de R''''' = MAC1
"inport" de R''''' = port de connexion du commutateur au host2
Spécifier comme action à R''''' la suppression du paquet
Envoyer R''''' au plan de données

```
                    Fin si
                Fin pour
            Fin si
Fin pour

Pour chaque host1 dans listeDesHosts faire :
        Si IP de host1 est paire alors :
                Pour host2 dans listeDesHosts faire :
                        Si IP de host2 est paire alors :
                                IP1 = IP de host1
                                IP2 = IP de host2
                                MAC1 = MAC host1
                                MAC2 = MAC host2

                                R = créer règle OpenFlow
                                Rendre R permanent dans les commutateurs
                                Spécifier protocole ARP pour R
                                Table de destination de R = 0
                                IP source de R = IP1
                                IP destination R = IP2
                                MAC source de R = MAC1
                                MAC destination de R = "ff:ff:ff:ff:ff:ff"
                                "inport" de R = port de connexion du commutateur à host1
                                Spécifier comme action à R la redirection vers tous les ports
                                Envoyer R au plan de données
                                R' = créer règle OpenFlow
                                Rendre R' permanent dans les commutateurs
                                Spécifier protocole ARP pour R'
                                Table de destination de R' = 0
                                IP source de R' = IP2
                                IP destination R' = IP1
                                MAC source de R' = MAC2
                                MAC destination de R' = "ff:ff:ff:ff:ff:ff"
                                "inport" de R' = port de connexion du commutateur au host2
                                Spécifier comme action à R' la redirection vers tous les ports
                                Envoyer R' au plan de données

                                R'' = créer règle OpenFlow
                                Rendre R'' permanent dans les commutateurs
                                Spécifier protocole ARP pour R''
                                Table de destination de R'' = 0
                                IP source de R'' = IP1
                                IP destination R'' = IP2
                                MAC source de R'' = MAC1
```

MAC destination de R'' = MAC2
"inport" de R'' = port de connexion du commutateur à host1
Spécifier comme action à R'' la redirection vers le port host2
Envoyer R'' au plan de données

R''' = créer règle OpenFlow
Rendre R''' permanent dans les commutateurs
Spécifier protocole ARP pour R'''
Table de destination de R''' = 0
IP source de R''' = IP2
IP destination R''' = IP1
MAC source de R''' = MAC2
MAC destination de R''' = MAC1
"inport" de R''' = port de connexion du commutateur au host2
Spécifier comme action à R''' la redirection vers le port host1
Envoyer R''' au plan de données
R'''' = créer règle OpenFlow
Rendre R'''' permanent dans les commutateurs
Spécifier protocole IP pour R''''
Table de destination de R'''' = 0
IP source de R'''' = IP1
IP destination R'''' = IP2
MAC source de R'''' = MAC1
MAC destination de R'''' = MAC2
"inport" de R'''' = port de connexion du commutateur à host1
Spécifier comme action à R'''' redirection vers le port host2
Envoyer R'''' au plan de données

R''''' = créer règle OpenFlow
Rendre R''''' permanent dans les commutateurs
Spécifier protocole IP pour R'''''
Table de destination de R''''' = 0
IP source de R''''' = IP2
IP destination R''''' = IP1
MAC source de R''''' = MAC2
MAC destination de R''''' = MAC1
"inport" de R''''' = port de connexion du commutateur au host2
Spécifier comme action à R''''' la redirection vers le port host1
Envoyer R''''' au plan de données

Fin si
Fin pour
Fin si
Fin pour

ANNEXE X

PSEUDO-CODE D'ÉCRITURE DES RÈGLES POUR COVISOR POUR LA COMPOSITION PARALLÈLE

```
R0 – créer règle OpenFlow
Spécifier table visée à 0 pour R0
Spécifier priorité de R0 à 1
Envoyer R0 au plan de données
Lire les paramètres de chaque client dans fichier Json
listeDesHosts – Lire la liste des hosts dans les paramètres

Pour chaque host1 dans listeDesHosts faire :
        Si IP de host1 est paire alors :
                Pour host2 dans listeDesHosts faire :
                        Si IP de host2 est impaire alors :
                                IP1 – IP de host1
                                IP2 = IP de host2
                                MAC1 = MAC host1
                                MAC2 = MAC host2

                                R – créer règle OpenFlow
                                Rendre R permanent dans les commutateurs
                                Spécifier protocole ARP pour R
                                Table de destination de R = 0
                                IP source de R – IP1
                                IP destination R – IP2
                                MAC source de R = MAC1
                                MAC destination de R = "ff:ff:ff:ff:ff:ff"
                                "inport" de R = port de connexion du commutateur à host1
                                Spécifier comme action1 à R l'affichage du paquet
                                Spécifier comme action2 à R la redirection vers tous les ports
                                Envoyer R au plan de données

                                R' – créer règle OpenFlow
                                Rendre R' permanent dans les commutateurs
                                Spécifier protocole ARP pour R'
                                Table de destination de R' – 0
                                IP source de R' – IP2
                                IP destination R' – IP1
                                MAC source de R' – MAC2
                                MAC destination de R' – "ff:ff:ff:ff:ff:ff"
                                "inport" de R' – port de connexion du commutateur au host2
                                Spécifier comme action1 à R l'affichage du paquet
                                Spécifier comme action2 à R la redirection vers tous les ports
                                Envoyer R' au plan de données
```

R'' = créer règle OpenFlow
Rendre R'' permanent dans les commutateurs
Spécifier protocole ARP pour R''
Table de destination de R'' = 0
IP source de R'' = IP1
IP destination R'' = IP2
MAC source de R'' = MAC1
MAC destination de R'' = MAC2
"inport" de R'' = port de connexion du commutateur à host1
Spécifier comme action1 à R l'affichage du paquet
Spécifier comme action2 à R la redirection vers le port à host2
Envoyer R'' au plan de données

R''' = créer règle OpenFlow
Rendre R''' permanent dans les commutateurs
Spécifier protocole ARP pour R'''
Table de destination de R''' = 0
IP source de R''' = IP2
IP destination R''' = IP1
MAC source de R''' = MAC2
MAC destination de R''' = MAC1
"inport" de R''' = port de connexion du commutateur au host2
Spécifier comme action1 à R l'affichage du paquet
Spécifier comme action2 à R la redirection vers le port à host1
Envoyer R''' au plan de données

R'''' = créer règle OpenFlow
Rendre R'''' permanent dans les commutateurs
Spécifier protocole IP pour R''''
Table de destination de R'''' = 0
IP source de R'''' = IP1
IP destination R'''' = IP2
MAC source de R'''' = MAC1
MAC destination de R'''' = MAC2
"inport" de R'''' = port de connexion du commutateur à host1
Spécifier comme action1 à R l'affichage du paquet
Spécifier comme action2 à R la redirection vers le port à host2
Envoyer R'''' au plan de données

R''''' = créer règle OpenFlow
Rendre R''''' permanent dans les commutateurs
Spécifier protocole IP pour R'''''
Table de destination de R''''' = 0
IP source de R''''' = IP2

```
                                IP destination R'''' = IP1
                                MAC source de R'''' = MAC2
                                MAC destination de R'''' = MAC1
                                "inport" de R'''' = port de connexion du commutateur au host2
                                 Spécifier comme action1 à R l'affichage du paquet
                                 Spécifier comme action2 à R la redirection vers le port à host1
                                Envoyer R'''' au plan de données

                        Fin si
                Fin pour
        Fin si
Fin pour

Pour chaque host1 dans listeDesHosts faire :
        Si IP de host1 est paire alors :
                Pour host2 dans listeDesHosts faire :
                        Si IP de host2 est paire alors :
                                IP1 = IP de host1
                                IP2 = IP de host2
                                MAC1 = MAC host1
                                MAC2 = MAC host2

                                R = créer règle OpenFlow
                                Rendre R permanent dans les commutateurs
                                Spécifier protocole ARP  pour R
                                Table de destination de R = 0
                                IP source de R = IP1
                                IP destination R = IP2
                                MAC source de R = MAC1
                                MAC destination de R = "ff:ff:ff:ff:ff:ff"
                                "inport" de R = port de connexion du commutateur à host1
                                Spécifier comme action à R la redirection vers tous les ports
                                Envoyer R au plan de données

                                R' = créer règle OpenFlow
                                Rendre R' permanent dans les commutateurs
                                Spécifier protocole ARP  pour R'
                                Table de destination de R' = 0
                                IP source de R' = IP2
                                IP destination R' = IP1
                                MAC source de R' = MAC2
                                MAC destination de R' = "ff:ff:ff:ff:ff:ff"
                                "inport" de R' = port de connexion du commutateur au host2
                                Spécifier comme action à R' la redirection vers tous les ports
                                Envoyer R' au plan de données
```

R'' = créer règle OpenFlow
Rendre R'' permanent dans les commutateurs
Spécifier protocole ARP pour R''
Table de destination de R'' = 0
IP source de R'' = IP1
IP destination R'' = IP2
MAC source de R'' = MAC1
MAC destination de R'' = MAC2
"Inport" de R'' = port de connexion du commutateur à host1
Spécifier comme action à R'' la redirection vers le port host2
Envoyer R'' au plan de données

R''' = créer règle OpenFlow
Rendre R''' permanent dans les commutateurs
Spécifier protocole ARP pour R'''
Table de destination de R''' = 0
IP source de R''' = IP2
IP destination R''' = IP1
MAC source de R''' = MAC2
MAC destination de R''' = MAC1
"inport" de R''' = port de connexion du commutateur au host2
Spécifier comme action à R''' la redirection vers le port host1
Envoyer R''' au plan de données

R'''' = créer règle OpenFlow
Rendre R'''' permanent dans les commutateurs
Spécifier protocole IP pour R''''
Table de destination de R'''' = 0
IP source de R'''' = IP1
IP destination R'''' = IP2
MAC source de R'''' = MAC1
MAC destination de R'''' = MAC2
"inport" de R'''' = port de connexion du commutateur à host1
Spécifier comme action à R'''' redirection vers le port host2
Envoyer R'''' au plan de données

R''''' = créer règle OpenFlow
Rendre R''''' permanent dans les commutateurs
Spécifier protocole IP pour R'''''
Table de destination de R''''' = 0
IP source de R''''' = IP2
IP destination R''''' = IP1
MAC source de R''''' = MAC2
MAC destination de R''''' = MAC1
"inport" de R''''' = port de connexion du commutateur au host2
Spécifier comme action à R''''' la redirection vers le port host1
Envoyer R''''' au plan de données

Fin si
Fin pour
Fin si
Fin pour

LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES

Ahmed, Mohamed Fekih, Chamssedine Talhi et Mohamed Cheriet. 2015. «Towards flexible, scalable and autonomic virtual tenant slices». In 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM). p. 720-726. IEEE.

Al-Shabibi, Ali, Marc De Leenheer, Matteo Gerola, Ayaka Koshibe, Guru Parulkar, Elio Salvadori et Bill Snow. 2014. « OpenVirteX: Make your virtual SDNs programmable ». In 3rd ACM SIGCOMM 2014 Workshop on Hot Topics in Software Defined Networking, HotSDN 2014, August 22, 2014 - August 22, 2014. (Chicago, IL, United states), p. 25-30. Coll. « HotSDN 2014 - Proceedings of the ACM SIGCOMM 2014 Workshop on Hot Topics in Software Defined Networking»: Association for Computing Machinery. <http://dx.doi.org/10.1145/2620728.2620741>.

Casado, Martin, Teemu Koponen, Rajiv Ramanathan et Scott Shenker. 2010. «Virtualizing the network forwarding plane». In Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow. p. 8. ACM.

Congdon, Paul T, Prasant Mohapatra, Matthew Farrens et Venkatesh Akella. 2014. « Simultaneously reducing latency and power consumption in openflow switches». IEEE/ACM Transactions on Networking (TON), vol. 22, no 3, p. 1007-1020.

Costi. 2014. «SDN Lesson #1 – Introduction to Mininet». < http://www.costiser.ro/2014/08/07/sdn-lesson-1-introduction-to-mininet/ >. Consulté le 01 Octobre 2015.

Dixit, A., K. Kogan et P. Eugster. 2014. « Composing heterogeneous SDN controllers with Flowbricks ». In 2014 IEEE 22nd International Conference on Network Protocols (ICNP), 21-24 Oct. 2014. (Los Alamitos, CA, USA), p. 287-92. Coll. « 2014 IEEE 22nd International Conference on Network Protocols (ICNP). Proceedings »: IEEE Computer Society. < http://dx.doi.org/10.1109/ICNP.2014.50 >.

Drutskoy, Dmitry A. 2012. « Software-defined network virtualization with flown ». Master's Thesis, Princeton University.

Facca, Federico M, Elio Salvadori, Holger Karl, Diego R López, Pedro Andrés Aranda Gutiérrez, Dejan Kostic et Roberto Riggio. 2013. « NetIDE: First steps towards an integrated development environment for portable network apps ». In 2013 Second European Workshop on Software Defined Networks. p. 105-110. IEEE.

Ferguson, Andrew D, Arjun Guha, Chen Liang, Rodrigo Fonseca et Shriram Krishnamurthi. 2012. « Hierarchical policies for software defined networks ». In Proceedings of the first workshop on Hot topics in software defined networks. p. 37-42. ACM.

Foster, Nate, Rob Harrison, Michael J Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story et David Walker. 2011. «Frenetic: A network programming language ». In ACM Sigplan Notices. Vol. 46, p. 279-291. ACM.

Giroire, Frédéric, Joanna Moulierac et Truong Khoa Phan. 2014. «Optimizing rule placement in software-defined networks for energy-aware routing». In Global Communications Conference (GLOBECOM), 2014 IEEE. p. 2523-2529. IEEE.

GitHub, Inc. 2015. « frenetic-lang/frenetic ». < https://github.com/frenetic-lang/frenetic >. Consulté le 07 Novembre 2015.

GitHub, Inc. 2015. « PrincetonUniversity/Coursera-SDN ». < https://github.com/PrincetonUniversity/Coursera-SDN/tree/master/assignments/pyretic-firewall >. Consulté le 01 Octobre 2015.

Gutiérrez, PA Aranda, E Rojas, A Schwabe, C Stritzke, R Doriguzzi-Corin, A Leckey, G Petralia, A Marsico, K Phemius et S Tamurejo. 2016. « NetIDE: All-in-one framework for next generation, composed SDN applications ». In 2016 IEEE NetSoft Conference and Workshops (NetSoft). p. 355-356. IEEE.

Gutz, Stephen, Alec Story, Cole Schlesinger et Nate Foster. 2012. «Splendid isolation: A slice abstraction for software-defined networks». In Proceedings of the first workshop on Hot topics in software defined networks. p. 79-84. ACM.

Hu, Fei, Qi Hao et Ke Bao. 2014. «A survey on software-defined network and openflow: from concept to implementation ». IEEE Communications Surveys & Tutorials, vol. 16, no 4, p. 2181-2206.

Jia, Xuya, Yong Jiang, Zehua Guo et Zhenwei Wu. 2016. «Reducing and Balancing Flow Table Entries in Software-Defined Networks ». In Local Computer Networks (LCN), 2016 IEEE 41st Conference on. p. 575-578. IEEE.

Jin, Xin, Jennifer Gossels, Jennifer Rexford et David Walker. 2015. «Covisor: A compositional hypervisor for software-defined networks». In 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15). p. 87-101.

Joshua Reich, Srinivas Narayana, Omid Alipourfard, Cole Schlesinger, Naga Praveen Kumar Katta, Christopher Monsanto, Jen Rexford, Nate Foster, David Walker. n/a. «Python + Frenetic = Pyretic ». < http://frenetic-lang.org/pyretic/ >. Consulté le 23 Septembre 2015.

Kannan, Kalapriya, et Subhasis Banerjee. 2013. « Compact TCAM: Flow entry compaction in TCAM for power aware SDN». In International Conference on Distributed Computing and Networking. p. 439-444. Springer.

Kreutz, D., F. M. V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky et S. Uhlig. 2015. «Software-Defined Networking: A Comprehensive Survey». Proceedings of the IEEE, vol. 103, no 1, p. 14-76.

Lab, Open Networking. inconnue. « POX Wiki ».
< https://openflow.stanford.edu/display/ONL/POX+Wiki >.
Consulté le 24 Octobre 2015.

Luo, Shouxi, Hongfang Yu et Lemin Li. 2015. « Practical flow table aggregation in SDN ». Computer Networks, vol. 92, p. 72-88.

Matias, Jon, Eduardo Jacob, David Sanchez et Yuri Demchenko. 2011. «An OpenFlow based network virtualization framework for the cloud ». In Cloud computing technology and science (CloudCom), 2011 IEEE Third International Conference on. p. 672-678. IEEE.

Mogul, Jeffrey C, Alvin AuYoung, Sujata Banerjee, Lucian Popa, Jeongkeun Lee, Jayaram Mudigonda, Puneet Sharma et Yoshio Turner. 2013. «Corybantic: towards the modular composition of SDN control programs». In Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks. p. 1. ACM.

Mogul, Jeffrey C, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, Andrew R Curtis et Sujata Banerjee. 2010. «Devoflow: Cost-effective flow management for high performance enterprise networks». In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks. p. 1. ACM.

Monsanto, Christopher, Nate Foster, Rob Harrison et David Walker. 2012. «A compiler and run-time system for network programming languages ». In ACM SIGPLAN Notices. Vol. 47, p. 217-230. ACM.

Reich, Joshua, Christopher Monsanto, Nate Foster, Jennifer Rexford et David Walker. 2013. «Modular sdn programming with pyretic». Technical Report of USENIX.

Reich, Joshua, Christopher Monsanto, Nate Foster, Jennifer Rexford, David Walker et Princeton Cornell. 2013. «Composing software defined networks». In Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI).

Salvadori, Elio, Roberto Doriguzzi Corin, Matteo Gerola, Attilio Broglio et Francesco De Pellegrini. 2011. «Demonstrating generalized virtual topologies in an openflow network ». In ACM SIGCOMM Computer communication review. Vol. 41, p. 458-459. ACM.

Sherwood, Rob, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown et Guru Parulkar. 2009. «Flowvisor: A network virtualization layer». OpenFlow Switch Consortium, Tech. Rep, p. 1-13.

Sköldström, Pontus, et Kiran Yedavalli. 2012. «Network virtualization and resource allocation in openflow-based wide area networks». In 2012 IEEE International Conference on Communications (ICC). p. 6622-6626. IEEE.

Sonkoly, Balázs, András Gulyás, Felicián Németh, János Czentye, Krisztián Kurucz, Barnabás Novák et Gábor Vaszkun. 2012. « OpenFlow virtualization framework with advanced capabilities». In 2012 European Workshop on Software Defined Networking. p. 18-23. IEEE.

Team, Mininet. 2015. «Mininet Walkthrough». < http://mininet.org/walkthrough/ >. Consulté le 01 Octobre 2015.

Tuysuz, Mehmet Fatih, Zekiye Kubra Ankarali et Didem Gözüpek. 2016. «A Survey on Energy Efficiency in Software Defined Networks». Computer Networks.

Wang, Shie-Yuan. 2014. «Comparison of SDN OpenFlow network simulator and emulators: EstiNet vs. Mininet». In Computers and Communication (ISCC), 2014 IEEE Symposium on. p. 1-6. IEEE.

Goransson, Paul, et Chuck Black. 2014. Software Defined Networks: A Comprehensive Approach. Elsevier.