Définition d'un langage pour l'unification des réseaux locaux et des réseaux du cloud

Mamadou Tahirou Bah

SORBONNE UNIVERSITÉ

THÈSE DE DOCTORAT DE SORBONNE UNIVERSITÉ

Spécialité

Informatique

École doctorale Informatique, Télécommunications et Électronique de Paris

Présentée par

Mamadou Tahirou BAH

Pour obtenir le grade de

DOCTEUR DE SORBONNE UNIVERSITÉ

Sujet de la thèse :

Définition d'un langage pour l'unification des réseaux locaux et des réseaux du cloud

soutenue publiquement le 06 Décembre 2019

devant le jury composé de :

Yacine GHAMRI-DOUDANE	Rapporteur	Université de la Rochelle
Rami LANGAR	Rapporteur	Univ.Paris-Est Marne-la-Vallée
Prosper CHEMOUIL	Examinateur	Orange Labs
Stefano SECCI	Examinateur	CNAM
Diogo MATTOS	Examinateur	Univ. F. de Fluminense (Brésil)
Guy PUJOLLE	Encadrant	Sorbonne Université
Thi-Mai-Trang NGUYEN	Directrice de thèse	Sorbonne Université

Remerciements

Je souhaite remercier en premier lieu ma directrice de thèse, *Mme. NGUYEN Thi-Mai-Trang*, Maître de conférences, HDR pour avoir accepté de diriger mes travaux dans le cadre de cette thèse. Je vous serai reconnaissant pour toujours du temps conséquent que vous m'avez accordé, de vos qualités pédagogiques et scientifiques, votre franchise et votre sympathie. J'ai beaucoup appris à vos côtés et je vous adresse toute ma gratitude pour tout cela. J'adresse également mes remerciements à mon encadrant de thèse, *M. Guy Pujolle*, professeur des universités, pour avoir accepté de m'accueillir dans son équipe, pour ses conseils avisés, son écoute et sa disponibilité sans faille et également tout le temps consacré à mon encadrement qui ont été prépondérants pour la bonne réussite de cette thèse. Je tiens à remercier tous les membres de l'équipe PHARE pour leur aide, leur soutien et pour leur sympathie durant mes quatre années de thèse.

Les moments que j'ai passé avec les thésards de l'équipe resteront gravés à jamais dans ma mémoire. Enfin, je remercie toute ma famille, à commencer par mes parents qui m'ont toujours soutenu, je remercie également mes frères et soeurs sans exception aucune, mes oncles et particulièrement à **El.Hadj Ibrahima Bagnan**, à mon épouse, mes cousins ainsi que mes amis pour leur encouragement.

« Lorsque tu ne sais pas où tu vas, rappelle toi d'où tu viens »
proverbe peul

A mes parents, mes frères et soeurs, mes amis et à mon épouse

Résumé

Ces dernières années sont marquées dans le monde des réseaux par l'émergence de nouvelles architectures fondées sur le Software Defined Networking (SDN) et le Network Function Virtualization (NFV). Le SDN est caractérisé par un plan de contrôle centralisé et découplé du plan de données tandis que le NFV consiste en une séparation des fonctions réseaux du matériel. Ce sont deux technologies différentes mais leur combinaison effective offrira des perspectives certaines dans l'usage des réseaux. A la différence des réseaux actuels appelés aussi « legacy », l'architecture SDN dispose d'un plan de contrôle centralisé. Cela constitue un des obstacles majeurs à son déploiement effectif. A cela, il faut ajouter également l'investissement initial nécessaire qui comporte le remplacement des équipements actuels, l'acquisition et le déploiement des nouveaux pouvant supporter la technologie SDN.

Nos travaux dans le cadre de cette thèse sont divisés en deux parties. Dans la première partie, nous allons explorer les différentes approches permettant l'introduction incrémentale du SDN en tenant compte des contraintes techniques, des coûts Capital Expenditure (CAPEX) et Operational Expenditure (OPEX). Cette phase est aussi appelée la phase de transition du « legacy » vers le SDN. Elle suppose la coexistence simultanée des deux technologies au sein d'un même réseau ou l'une à côté de l'autre dans des réseaux voisins par exemple. Cela implique également une communication entre les technologies « legacy » et SDN) pour assurer des services aux utilisateurs. Les réseaux SDN hybrides sont donc le fruit de la Coexistance, Communication, Croisement (3C) de ces technologies. Dans la deuxième partie de cette thèse, nous sommes intéressés à l'analyse des performances des contrôleurs SDN les plus avancés en termes de gestion de la topologie, la tolérance aux pannes, le débit des contrôleurs, le trafic de contrôle et le passage à l'échelle. Les résultats obtenus indiquent que le contrôleur Open Network Operating System (ONOS) est le mieux adapté pour cette phase de transition. Cela s'explique notamment par la diversité des fonctionnalités tel que « clustering » qu'il propose mais également les performances du contrôleur ONOS sur la tolérance aux pannes et sur le passage à l'échelle.

Mots clés

SDN, NFV, Cloud, Hybrid networks, ONOS, OpenDaylight, topology management, performance

Abstract

The last few years have been marked in the world of networks by the emergence of new architectures based on Software Defined Networking (SDN) and Network Function Virtualization (NFV). SDN is characterized by a centralized control plan and decoupled from the data plan while NFV consists of a separation of the hardware network functions. These are two different technologies but their effective combination will offer desperate prospects in the use of networks. Unlike current networks, also known as legacy networks, the SDN architecture has a centralized control plan. This is one of the major obstacles to its effective deployment. To this, we must also add the necessary initial investment, which includes the replacement of existing equipments, the acquisition and deployment of new equipments capable of supporting SDN technology. Our work under this thesis is divided into two parts. In the first part, we will explore the different approaches allowing the incremental introduction of the SDN taking into account technical constraints, Capital Expenditure (CAPEX) and Operational Expenditure (OPEX) costs. This phase is also called the transition phase from legacy to SDN. It implies the simultaneous coexistence of the two technologies within the same network or next to each other in neighbouring networks, for example. It also involves the communication between legacy and SDN technologies to provide services to users. Hybrid SDN networks are therefore the result of the Coexistence, Communication, Crossbreeding (3C) of these technologies. In the second part of this thesis, we are interested in analyzing the performance of the most advanced SDN controllers in terms of topology management, fault tolerance, controller throughput, control traffic and scalability. The results obtained indicate that the Open Network Operating System (ONOS) controller is best suited for this transition phase. This is explained in particular by the diversity of functionalities such as "clustering" which can increase the performance of the ONOS controller on fault tolerance and scalability.

Mots clés

SDN, NFV, Cloud, Hybrid networks, ONOS, OpenDaylight, topology management, performance

Table des matières

1 Introduction **23**
- 1.1 Motivation de la thèse . . . 23
- 1.2 Présentation du projet CARP . . . 25
  - 1.2.1 Objectifs de CARP . . . 25
  - 1.2.2 Architecture de CARP . . . 26
- 1.3 Contributions de la thèse . . . 26
- 1.4 Plan de la thèse . . . 28

2 SDN, NFV et Cloud **31**
- 2.1 Introduction . . . 32
- 2.2 SDN . . . 32
  - 2.2.1 Les fonctions de base du SDN . . . 33
  - 2.2.2 La couche application . . . 33
  - 2.2.3 La couche contrôle . . . 33
  - 2.2.4 La couche infrastructure . . . 34
  - 2.2.5 La couche management . . . 34
  - 2.2.6 Interface nord . . . 34
  - 2.2.7 Interface sud . . . 34
    - 2.2.7.1 Le protocole OpenFlow . . . 35
    - 2.2.7.2 Le protocole NETCONF . . . 37
    - 2.2.7.3 Programing Protocol-Independent Packet Processor (P4) . . . 38
    - 2.2.7.4 I2RS (Interface to Routing System) . . . 42
  - 2.2.8 Interface EST/ OUEST . . . 43
  - 2.2.9 Architecture distribués du SDN . . . 44
    - 2.2.9.1 Plan de contrôle logiquement centralisé . . . 44
    - 2.2.9.2 Plan de contrôle logiquement distribué . . . 44
- 2.3 Interopérabilité entre contrôleurs . . . 45
- 2.4 Nombre et emplacement des contrôleurs multiples . . . 45
- 2.5 Cohérence dans les contrôleurs SDN distribués . . . 46

2.5.1 Modèle de Cohérence éventuel (« Eventual Consistency Model ») . . 46
2.5.2 Modèle de cohérence forte (« Strong Consistency Model ») 47
2.6 Network Function Virtualization (NFV) 47
2.6.1 Architecture du NFV 48
2.6.1.1 Element Management (EM) 48
2.6.1.2 VNF 48
2.6.1.3 NFV Infrastructure (NFVI) 49
2.6.1.4 Network Function Virtualization Orchestrator (NFVO) . . . 49
2.6.1.5 VNF Manager (VNFM) 49
2.6.1.6 VIM 49
2.6.2 Orchestration des NFV 49
2.6.3 Projets et plate-formes NFV 49
2.6.3.1 OPNFV 50
2.6.3.2 OpenMANO 50
2.6.3.3 ONAP 50
2.7 Relation entre SDN, NFV et Cloud Computing 51
2.8 Organismes de standardisation 51
2.9 Conclusions 51

3 Le paradigme des réseaux SDN hybrides 55
3.1 Introduction 56
3.2 Réseaux hybrides du SDN 56
3.3 Modèle de réseaux hybrides SDN 57
3.3.1 Modèle hybride basé sur la topologie 57
3.3.2 Sur des noeuds de bordure 58
3.3.3 Modèle hybride basé sur le service 58
3.3.4 Modèle hybride basé sur la classe 59
3.3.5 Modèle hybride basé sur l'intégration 60
3.4 Prototypes et implémentations de solutions hybrides 60
3.4.1 Routing Control Platform (RCP) 60
3.4.2 Panopticon 61
3.4.3 HybNET 61
3.4.4 HybridFlow 61
3.4.5 OpenRouteFlow et ClosedFlow 62
3.4.6 B4 62
3.4.7 Fibbing 62
3.4.8 SDN-IP 62
3.5 Architecture CARP pour les réseaux hybrides 63
3.5.1 Architecture 63
3.5.1.1 Plan de contrôle 63
3.5.1.2 Plan de données 63
3.5.1.3 Agent I2RS 63

3.5.1.4 Contrôleur 63
3.5.2 Fonctionnement 64
3.5.3 Positionnement de notre approche par rapport la littérature 65
3.6 Conclusions 65

4 Évaluation des performances : découverte de la topologie 67
4.1 Introduction 68
4.2 Techniques d'évaluation des performances du SDN 68
4.2.1 Modèle analytique 68
4.2.2 Simulation 69
4.2.3 Approche expérimentale 70
4.3 État de l'art sur l'analyse des performances du SDN 70
4.4 Choix des contrôleurs ONOS et ODL 71
4.5 Gestion de la topologie dans le SDN 72
4.5.1 Découverte des hôtes 72
4.5.2 Découverte des commutateurs 73
4.5.3 Découverte des liens entre commutateurs 73
4.5.3.1 OpenFlow Discovery Protocol (OFDP) 73
4.5.3.2 OpenFlow Discovery Protocol (OFDP) version 2 75
4.6 Plate-formes expérimentales 76
4.6.1 Architecture de la plate-forme expérimentale 76
4.6.2 Méthodologie pour mesurer le temps de découverte de la topologie . 78
4.7 Analyse des résultats 79
4.7.1 Découverte de la topologie : Cas d'un contrôleur unique 80
4.7.2 Découverte de la topologie : Cas de plusieurs contrôleurs (*cluster*) . . 81
4.8 Conclusions 81

5 Évaluation des performances : Tolérance aux pannes 87
5.1 Introduction 87
5.2 Changement de la topologie dans le SDN 88
5.3 Temps d'interruption avec les contrôleurs ONOS et ODL : Cas d'un contrôleur unique 89
5.4 Distribution du temps d'interruption avec les contrôleurs ONOS et ODL . . . 92
5.5 Tolérance aux pannes : Architecture Contrôle et Automatisation unifiés de services Réseaux Programmables (CARP) 93
5.6 Conclusions 95

6 Performances du contrôleur et passage à l'échelle 99
6.1 Introduction 99
6.2 État de l'art 100
6.3 Passage à l'échelle 101
6.4 Trafic de contrôle 102

6.4.1 Approche réactive . . . 102
6.4.2 Approche pro-active . . . 102
6.4.3 Approche hybride . . . 103
6.5 Plate-forme expérimentale . . . 103
6.5.1 Débit du contrôleur . . . 103
6.5.2 Expérimentation pour la passage à l'échelle . . . 103
6.5.2.1 Description des expériences . . . 105
6.5.2.2 Méthodologie . . . 105
6.5.2.3 Scénarios de tests . . . 105
6.5.3 Trafic de contrôle . . . 106
6.5.3.1 Méthode d'évaluation . . . 106
6.5.3.2 Description des expériences . . . 107
6.6 Analyse des résultats . . . 107
6.6.1 Analyse du débit . . . 108
6.6.2 Trafic de contrôle . . . 108
6.6.3 Passage à l'échelle . . . 109
6.7 Conclusion . . . 109

7 Conclusion **113**
7.1 Perspectives . . . 114
7.2 Publications . . . 115
7.2.1 Conférences . . . 115
7.2.2 Rapports techniques . . . 115

A Annexes **117**
A.1 Contribution à l'IETF (Internet Engineering Task Force) . . . 117
A.2 Commandes Mininet . . . 117
A.3 Cluster ONOS . . . 118
A.3.1 Configuration d'un cluster atomix *(atomix.conf)* . . . 118
A.3.2 Ajouter une instance ONOS à un cluster . . . 119
A.4 Cluster OpenDayLight . . . 120
A.4.1 Configuration du cluster . . . 120
A.4.1.1 akka.conf . . . 120
A.4.1.2 modules.conf . . . 121
A.4.1.3 module-shard.conf . . . 122
A.5 Iperf . . . 123
A.5.1 Coté serveur . . . 123
A.5.2 Côté client . . . 123

Bibliographie **132**

Table des figures

1.1 Objectifs de CARP 25
1.2 Architecture de CARP 26

2.1 Architecture du SDN 32
2.2 Éléments d'une table de flux (« flow ») 35
2.3 Les couches du protocole NETCONF 37
2.4 Architecture de P4 40
2.5 Le modèle de transfert abstrait 40
2.6 Architecture du Client et de l'Agent I2RS *(IETF)* 42
2.7 Processus d'élection du *leader* avec le protocole *RAFT* 47
2.8 Architecture de référence du NFV proposée par *ETSI* [1] 48

3.1 Modèle hybride basé sur la topologie 58
3.2 Modèle hybride basé sur le service 59
3.3 Modèle hybride basé sur la classe 59
3.4 Modèle hybride basé sur l'intégration 60
3.5 Nouvelle architecture centralisée pour les réseaux distribués 64

4.1 Scénario basic du LLDP 74
4.2 Structure d'une trame Link Layer Discovery Protocol (LLDP) 75
4.3 Plate-forme expérimentale avec un seul contrôleur 77
4.4 Plate-fome expérimentale avec deux contrôleurs 77
4.5 Cluster avec deux instances 77
4.6 Exemple de topologie en arbre binaire avec une profondeur $d = 4$ 78
4.7 Temps de découverte de topologie des contrôleurs ONOS et ODL en "single mode" 80
4.8 Volume de données de contrôle pour la découverte de topologie avec ONOS et ODL 80
4.9 Mesure du trafic inter-contrôleurs 81

4.10 Temps de découverte de topologie des contrôleurs ONOS et ODL en mode cluster 82

5.1 Mécanisme de changement de la topologie 88
5.2 Topologie du réseau *GEANT* exécutée sur ONOS 90
5.3 Temps d'interruption en changement de topologie avec ONOS en "single mode" 90
5.4 Temps d'interruption en cas de changement de topologie avec ODL en "single mode" 91
5.5 Topologie du réseau du scénario de test 92
5.6 Distribution du temps de réaction (PDF) pour des contrôleurs ONOS et ODL 92
5.7 Topologie d'un réseau basé sur TRILL 93
5.8 Nouveau temps de convergence dans un réseau TRILL avec I2RS 94
5.9 Temps de convergence sans l'intervention du contrôleur 94
5.10 Nouveau temps de convergence avec l'intervention du contrôleur 94

6.1 Plate-forme expérimentale pour d'un cluster ONOS 104
6.2 Plate-forme d'expérimentation pour l'analyse du trafic de contrôle 106
6.3 Plate-forme expérimentale pour Cbench 107
6.4 Test Cbench en « throughput mode » avec 16 commutateurs 107
6.5 Évolution du débit (UDP) en fonction du nombre de noeuds avec ONOS . . . 110
6.6 Comparaison des débits entre trois topologies : *GEANT*, RENATER et 63 commutateurs en mode cluster avec ONOS 110

Liste des tableaux

1.1 Sommaire des contributions 28

2.1 Champs reconnus par le standard OpenFlow 35
2.2 Les opérations de base du protocole NETCONF 38
2.3 Quelques groupes de travail de l'ONF sur le SDN 52
2.4 Quelques groupes de travail de l'IETF/IRTF et de l'ETSI sur le SDN 53
2.5 Quelques groupes de travail de l'IEEE sur le SDN 53
2.6 Quelques groupes de travail de l'UIT sur le SDN 54

4.1 Critères de comparaison des contrôleurs SDN 84
4.2 Méthodes pour la découverte de topologie dans le SDN 85
4.3 Paramètres clés des topologies 85

6.1 Paramètres clés des topologies utilisées 105
6.2 Nombre total de messages de contrôle échangés suite à l'arrivée d'un nouveau flux 108

Glossaire

3C Coexistance, Communication, Croisement.

API Application Programming Interface.
AS Autonomous System.

BGP Border Gateway Protocol.

CAPEX Capital Expenditure.
CARP Contrôle et Automatisation unifiés de services Réseaux Programmables.
CLI Command Line Interface.
CPU Central Processing Unit.

DPI Deep Packet Inspection.
DPID Datapath ID.

ETSI European Telecommunications Standards Institute.

FAI Fournisseur d'Accès à Internet.
FCAPS Fault, Configuration, Accounting, Performance, and Security.
FIB Forwarding Information Base.

GAFAM Google, Amazon, Facebook, Apple et Microsoft.

I2RS Interface to Routing System.
ICIN Conference on Innovation in Clouds, Internet and Networks and Workshops.
IETF Internet Engineering Task Force.
IGP Interior Gateway Protocol.

IP Internet Protocol.
IS-IS Intermediate system to intermediate system.

LIP6 Laboratoire Informatique de Paris 6.
LLDP Link Layer Discovery Protocol.
LLDPDU LLDP Data Unit.

MANO Management Orchestration.
MCDM Multi-Criteria Decision Making.
Media Access Control MAC.
MIB Management Information Base.

NFV Network Function Virtualization.
NFVI NFV Infrastructure.
NFV-O Network Functions Virtualisation Orchestrator.
NIST National Institute of Standards and Technology.
NOS Network Operating System.

ODL OpenDaylight.
OFDP OpenFlow Discovery Protocol.
ONAP Open Network Automation Platform.
ONF Open Networking Foundation.
ONOS Open Network Operating System.
OpenFlow OpenFlow.
OPEX Operational Expenditure.
OPNFV Open Platform for Networks Functions Virtualization.
OSGi Open Services Gateway initiative.
OSPF Open Shortest Path First.
OVS Open vSwitch.

P4 Programing Protocol-Independent Packet Processor.
PCE Path Computation Element.

RAM Random-Access Memory.
RIB Routing Information Base.
RPC Remote procedure call.

SDN Software Defined Networking.

SNMP Simple Network Management Protocol.

SOA Service-Oriented Architecture.

SSH Secure Shell Transport Layer Protocol.

TCP Transmission Control Protocol.

TDG Table Dependecy Graphs.

TLL Time to Live.

TLS Transport Layer Security.

TLV Type-Length-Value.

TRILL TRansparent Interconnection of Lots of Links.

UDP User Datagram Protocol.

VIM Virtualised Infrastructure Manager.

VNF Virtualized Network Function.

Wide Area Network WAN.

XML Extensible Markup Language.

Chapitre 1

Introduction

Sommaire

1.1 Motivation de la thèse 23
1.2 Présentation du projet CARP 25
1.2.1 Objectifs de CARP 25
1.2.2 Architecture de CARP 26
1.3 Contributions de la thèse 26
1.4 Plan de la thèse 28

1.1 Motivation de la thèse

Le SDN (réseaux définis par logiciels) [2] a envahi depuis quelques années le monde des réseaux et plus particulièrement les réseaux du cloud. Le SDN apporte une grande facilité dans l'automatisation et une grande souplesse dans la gestion des réseaux avec un contrôleur central. Son architecture avec un plan de contrôle centralisé apporte une grande flexibilité et une simplicité pour la mise en œuvre des décisions de routage. Avec le SDN, toute l'intelligence du réseau est centralisée sur un contrôleur, le principe est basé sur une séparation du **plan de contrôle** et du **plan de données**. Les équipements (commutateurs ou routeurs) sont consacrés à la seule fonction d'acheminement des données. Cette centralisation permet d'avoir une vue globale sur l'ensemble des équipements du réseau. En outre, elle permet aussi de détecter et de réagir plus facilement suite à tout événement anormal qui interviendrait dans le réseau et facilite également la mise à jour dynamique des règles d'acheminement des données. Le contrôle centralisé offre la possibilité d'analyser en temps réel le trafic et de modifier si besoin les règles d'acheminement des paquets. Ce qui n'est pas facile, voire impossible avec les réseaux traditionnels (« legacy »).

Malgré ses nombreux avantages, le SDN présente tout de même quelques limites liées notamment à son architecture : le passage à l'échelle ou encore la vulnérabilité du contrôleur qui constitue un point unique de défaillance. Cependant, ces points connaissent une évolution très positive depuis la parution de la version 4 (OpenFlow 1.3) du protocole OpenFlow [3] qui permet à un commutateur de s'associer simultanément à plusieurs contrôleurs SDN, mais aussi à l'arrivée des nouveaux contrôleurs avec une architecture distribuée [4, 5] tels *ONOS* et *ODL*.

Le SDN et le NFV sont deux concepts indépendants, mais complémentaires. C'est à dire que le SDN peut être déployé sans le NFV et vice versa, mais leur combinaison offre une valeur ajoutée certaine aux entreprises.

Avec les réseaux traditionnels, les fonctions réseaux sont implémentées sur du matériel dédié. Le développement du logiciel et leur conception sont assurés par leur fabricant. Chaque fournisseur propose de manière exclusive des commutateurs ou routeurs avec un logiciel (« firmware ») propriétaire sur son matériel. Ils sont caractérisés par un fort couplage du matériel et du logiciel. Le fabricant est le seul décideur de la conception, de l'évaluation, de l'évolution ou encore du déploiement d'un nouveau protocole. La possibilité d'innovation est assez limitée pour les utilisateurs finaux.

L'émergence du Cloud et des « data-centers » a modifié considérablement le trafic sur Internet et nécessite une adaptation des protocoles de routage actuels. Outre le trafic traditionnel sur Internet qui peut être parfaitement satisfait par les protocoles traditionnels tels que Open Shortest Path First (OSPF) [6], Intermediate system to intermediate system (IS-IS) [7] ou encore Border Gateway Protocol (BGP) [8], il faut ajouter un autre type de trafic, celui au sein des « data-centers ». Il existe deux types de trafic dans les « data-centers » (DC), le trafic **intra-data-center** et le trafic **inter-data-center** [9]. Les « data-centers » sont constitués de plusieurs milliers de serveurs physiques et plusieurs dizaines de milliers de serveurs virtuels. Ces nouvelles architectures viennent avec de nouvelles exigences en termes de qualité de service, de haute disponibilité, de mobilité, de gestion et de passage à l'échelle.

Les travaux de cette thèse s'inscrivent dans le cadre du projet CARP[1] (Contrôle et Automatisation unifiés de services Réseaux Programmables). Nous ferons une présentation du projet (CARP) dans la section 1.2. Nous présenterons également les objectifs de ce projet dans 1.2.1.

Le paradigme SDN et les technologies NFV offrent de nouvelles perspectives dans le monde des réseaux. Cependant, son déploiement effectif doit faire face à plusieurs défis tant économiques (CAPEX, OPEX) que techniques (sécurité, passage à l'échelle,...).

La combinaisons de ces deux technologies offre des potentialités et nouvelles possibilités aux opérateurs de télécommunications et aux fabricants des équipements réseaux sur l'automatisation, la gestion de la mobilité des services ainsi que la réduction des coûts d'investissement et d'exploitation pour les infrastructures réseaux. Cependant, la non compatibilité du SDN avec un plan de contrôle centralisé et des réseaux traditionnels appelés aussi « legacy » qui quant à eux disposent d'un plan de contrôle distribué constitue un obstacle au déploiement effectif du SDN. A cela, il faut ajouter également le coût d'investissement initial nécessaire au remplacement des équipements actuels par du SDN et la formation du personnel pour assurer leur exploitation.

1. FUI 19 CARP (Contrôle et Automatisation unifiés de services Réseaux Programmables)

1.2 Présentation du projet CARP

CARP s'appuie sur les concepts et technologies SDN et NFV pour réaliser l'automatisation du déploiement et de la mobilité des services réseaux qui reste délicate à accomplir avec les paradigmes traditionnels.

La combinaison du SDN, de NFV et du Cloud ouvre de nouvelles perspectives dans la conception de systèmes agiles et hautement disponibles avec le bénéfice d'une réduction des coûts d'investissement et d'exploitation. Cette conception est toujours difficile à réaliser et reste coûteuse. La figure 1.1 présente l'écosystème de CARP et l'interaction entre les différentes technologies ciblées.

Le consortium CARP regroupe sept partenaires dont cinq industriels (Gandi, Ucopia, CityPassenger, ByO Networks, Kalray) et deux académiques(UPMC, Télécom SudParis) avec des compétences et des positionnements complémentaires et essentiels pour réaliser les objectifs cités.

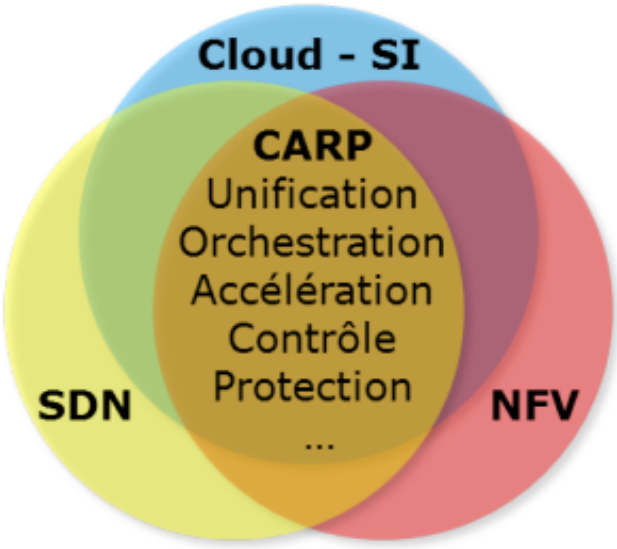


FIGURE 1.1 – Objectifs de CARP

1.2.1 Objectifs de CARP

Les réseaux traditionnels sont encore très majoritairement utilisés dans le monde des entreprises tant au niveau des PME que des grandes entreprises. Ces réseaux présentent quelques inconvénients par rapport aux nouveaux besoins et exigences des entreprises.

CARP s'était fixé pour objectif de faciliter le déploiement et améliorer la configuration dynamique des réseaux pour :

— simplifier et sécuriser la conception des services réseaux,
— réduire les coûts d'intégration des systèmes d'information,
— réduire leur coût d'exploitation et de maintenance,

— réduire les délais de lancement de nouveaux services et fonctionnalités.

L'architecture de CARP devait permettre de profiter des avantages de toutes ses technologies. Elle doit définir la meilleure solution pour l'intégration du SDN et du NFV dans les entreprises.

1.2.2 Architecture de CARP

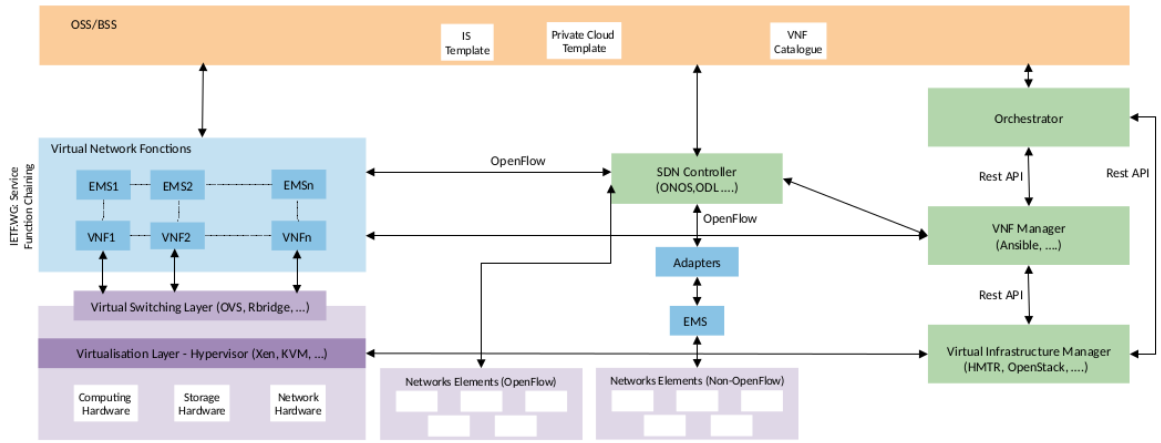


FIGURE 1.2 – Architecture de CARP

La figure 1.2 présente l'architecture, qui a été validé par le consortium, constituée dans le cadre de ce projet. Nous présenterons en détails les composants de cette architecture dans le chapitre 2.

1.3 Contributions de la thèse

Nos travaux dans le cadre de cette thèse sont divisés en deux parties. Dans la première partie (chapitres 2 et 3), nous avons effectué les travaux suivants : un état de l'art sur le SDN, les défis (sécurité, passage à l'échelle,...) liés à son architecture, les obstacles à son déploiement à grande échelle, les approches pour son introduction progressive, la notion de réseaux hybrides (SDN résultat de la coexistence avec les réseaux traditionnels, appelés aussi « Legacy »). [1] Nous avons ensuite analysé les différents modèles de réseaux hybrides introduits dans la littérature en termes de coûts OPEX/CAPEX. Nous avons enfin exploré les principales implémentations des réseaux hybrides tel que le *B4* [10] proposé et déployé par *Google* pour interconnecter et gérer ses « data-centers » sous SDN, la solution *SDN IP* [11] proposée à l'origine par *ON.LAB* (Open Networking Lab) contrôlée par la suite par la fondation *ONF* (Open Networking Foundation) [2] avec laquelle il a fusionné en *2016* fournit un mécanisme de partage des informations de routage entre les domaines gérés sous le contrôle des réseaux traditionnels et des domaines sous contrôle SDN. Les travaux réalisés dans cette partie nous ont permis de découvrir et maîtriser la technologie SDN, son architecture, ses avantages, les protocoles spécifiques tel que OpenFlow (OpenFlow), les différents contrôleurs proposés dans le cadre de plusieurs projets

1. Dans la suite de ce document, nous utiliserons indifféremment les expressions réseaux traditionnels ou « réseaux legacy » pour designer les réseaux distribués actuels

par la communauté « open-sources », ainsi que les principaux outils de simulation tel *Mininet*[12]. Nous avons enfin proposé une architecture hybride SDN dans le cadre du projet CARP qui réalise le meilleur compromis entre l'efficacité des architectures centralisées et la robustesse des architectures distribuées. Cette architecture est basée sur un contrôleur centralisé qui configure de manière éphémère la table de routage avec des chemins alternatifs déjà calculés par les protocoles de routage distribués à état de liens comme OSPF, IS-IS et BGP.

L'évaluation expérimentale dans un réseau avec le protocole TRansparent Interconnection of Lots of Links (TRILL) [13] qui implémente le protocole IS-IS a montré que la restauration du chemin, suite à la défaillance d'une liaison, est presque instantanée. Une partie de ce travail a donné lieu à une publication dans une conférence internationale au (*3rd Smart Cloud Networks & Systems (SCNS)*) [14].

Dans la deuxième partie de cette thèse (chapitres 4, 5 et 6), nos travaux porteront sur l'évaluation des performances des principaux contrôleurs SDN open-source. L'objectif de ce travail est l'identification de la solution SDN (contrôleur), parmi les projets open-source actuellement proposés, la mieux appropriée pour l'introduction progressive du SDN mais aussi offrant une interface de communication avec les technologies cloud et virtualisation de fonctions réseaux tel OpenStack. Ce travail sera réalisé en plusieurs étapes. Dans un premier temps nous analyserons les performances des contrôleurs dans un environnement composé d'un seul contrôleur avant de passer dans la suite à une analyse avec plusieurs contrôleurs. Les résultats dans les chapitres 4 et 5 ont fait l'objet de deux publications respectivement dans une conférence internationale (2019 *22nd* Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)) [15] et dans un rapport technique par l'Open Networking Foundation (ONF) [16].

Nous effectuerons dans un premier temps un état de l'art des études sur les performances des contrôleurs SDN puis nous identifierions les critères tant qualitatifs que quantitatifs ainsi que les outils et techniques utilisés pour l'analyse de leur performance. Nous observerons que la plupart des recherches sur les performances des contrôleurs SDN ont porté sur la célérité et la latence du traitement des requêtes. Ces études répondent principalement aux questions suivantes :

— Quel est temps de réponse d'un contrôleur pour satisfaire une requête ?
— Combien de requêtes un contrôleur peut-il traiter par unité de temps (seconde) ?

Les réponses à ces questions sont certes très pertinentes dans le choix d'un contrôleur, cependant, elles n'apportent pas d'informations sur l'une des fonctions fondamentales de notre point de vue de la technologie SDN, la découverte et la gestion de la topologie. En effet, l'une des tâches principales d'un contrôleur SDN est de garantir à tout instant une bonne connaissance de la topologie et une prise en compte immédiate des changements qui y interviendraient. Ainsi, nous avons proposé l'introduction de deux nouvelles métriques pour l'évaluation des performances des contrôleurs SDN qui sont :

— Temps de découverte de la topologie (TDT)
— Temps de mise à jour de la topologie (TMT)

Nous avons également proposé une méthode pour la mesure de ces métriques dans le cadre d'une configuration avec un seul contrôleur puis avec plusieurs contrôleurs. Nous avons enfin réalisé une étude comparative de deux solutions SDN les plus évoluées du moment dans le but d'évaluer l'impact de leurs choix d'implémentation du protocole de découverte de la topologie OFDP sur leurs performances.

1.4 Plan de la thèse

Le tableau 1.1 présente un sommaire de l'organisation de cette thèse.

Organisation du manuscrit	
Chapitre 2	- Présente les concepts SDN, NFV et cloud - Présente relations entre ces concepts - Les avantages offertes par leur combinaison - Les protocoles l'interface sud, est/ouest de l'interopérabilité des contrôleurs - Présente des aspects avancés du SDN
Chapitre 3	- Introduit la notion des réseaux hybrides SDN - Des problématiques de l'adoption du SDN, de son introduction incrémentale - Des problématiques de la coexistence avec les réseaux « legacy » - Présente une nouvelle architecture des réseaux hybrides - Présente un cas d'utilisation sur la réduction du temps de convergence dans un réseau distribué
Chapitre 4	- État de l'art des études de performance du SDN - Analyse des performances contrôleurs ONOS et OpenDaylight (ODL) - Découverte de la topologie
Chapitre 5	- Analyse des performances sur la tolérance aux pannes
Chapitre 6	- Analyse de performances sur le débit des contrôleurs ONOS et ODL - Analyse de performances sur le trafic de contrôle - Analyse des performances du SDN sur le passage à l'échelle
Chapitre 7	- Une conclusion de la thèse - Perspectives

TABLE 1.1 – Sommaire des contributions

Le manuscrit de la thèse sera organisé comme suit : le chapitre 2 présentera une *background* des principales technologies abordées lors de la thèse (SDN, NFV) avec une présentation détaillée des composants de son architecture SDN ainsi que des principaux protocoles de son interface sud tel que OpenFlow ou encore Programing Protocol-Independent Packet Processor (P4). Ensuite, nous présenterons brièvement le concept et l'architecture NFV ainsi que le rapport entre cette technologie et le SDN.

Le chapitre 3 abordera le concept des réseaux hybrides SDN avec une présentation détaillée des approches et des modèles de cette architecture. Ce chapitre présentera également les principales implémentions et prototypes des solutions SDN hybrides ainsi que la nouvelle architecture hybride proposée dans le projet CARP .

Le chapitre 4 portera sur une évaluation des performances des contrôleurs ONOS et ODL sur la découverte de la topologie dans un environnement avec un contrôleur unique puis dans le cas d'un *cluster* [2] (contrôleurs multiples). Il présentera également une évaluation de l'impact des échanges inter-

2. Dans le reste de ce document nous utiliserons indifféremment les termes *cluster*, multi-contrôleurs ou contrôleurs multiples pour designer une configuration SDN avec plusieurs instances d'un même contrôleur

contrôleurs sur les performances du système.

La chapitre 5 présentera une évaluation des performances des contrôleurs ONOS, ODL et l'architecture hybride CARP sur la tolérance aux pannes. Le chapitre 6 présentera une évaluation de performances portant sur le débit, le trafic de contrôle puis sur le passage à l'échelle dans un environnement de *cluster*. Le chapitre 7 se terminera par une conclusion générale de la thèse.

Chapitre 2

SDN, NFV et Cloud

Sommaire

2.1 Introduction 32
2.2 SDN 32
2.2.1 Les fonctions de base du SDN 33
2.2.2 La couche application 33
2.2.3 La couche contrôle 33
2.2.4 La couche infrastructure 34
2.2.5 La couche management 34
2.2.6 Interface nord 34
2.2.7 Interface sud 34
2.2.8 Interface EST/ OUEST 43
2.2.9 Architecture distribués du SDN 44
2.3 Interopérabilité entre contrôleurs 45
2.4 Nombre et emplacement des contrôleurs multiples 45
2.5 Cohérence dans les contrôleurs SDN distribués 46
2.5.1 Modèle de Cohérence éventuel (« Eventual Consistency Model ») 46
2.5.2 Modèle de cohérence forte (« Strong Consistency Model ») 47
2.6 Network Function Virtualization (NFV) 47
2.6.1 Architecture du NFV 48
2.6.2 Orchestration des NFV 49
2.6.3 Projets et plate-formes NFV 49
2.7 Relation entre SDN, NFV et Cloud Computing 51
2.8 Organismes de standardisation 51
2.9 Conclusions 51

2.1 Introduction

Dans ce chapitre nous présentons le contexte technologique de cette thèse. Il présente en détails les principaux concepts qui serviront de *background* dans le cadre de cette thèse.

Ce chapitre est organisé comme suit : dans la section 2.2 nous présenterons un rappel sur l'architecture du SDN, ses fonctions de base, ses différentes interfaces (nord/sud et est/ouest). Dans la section 2.3, nous aborderons la problématique de l'interopérabilité des contrôleurs SDN hétérogènes. Puis, dans la section 2.4, nous introduirons la problématique du nombre et de l'emplacement des contrôleurs SDN dans le cadre d'un environnement avec de multiples contrôleurs appelé aussi *cluster*. Dans la section 2.5, nous présenterons les différents modèles de cohérence dans une architecture SDN distribuée ainsi que les principaux protocoles actuellement utilisés pour assurer la synchronisation des données dans un *cluster*. La section 2.6 présentera le concept des NFV avec un regard plus approfondi sur les composants de son architecture et les principaux projets open-sources. La section 2.7 présente la relation entre les technologies (SDN, NFV et cloud) présentées dans ce chapitre tandis que la section 2.8 présente les principaux organismes qui participent à l'effort de standardisation. Enfin, la section 2.9 introduit une conclusion à ce chapitre.

2.2 SDN

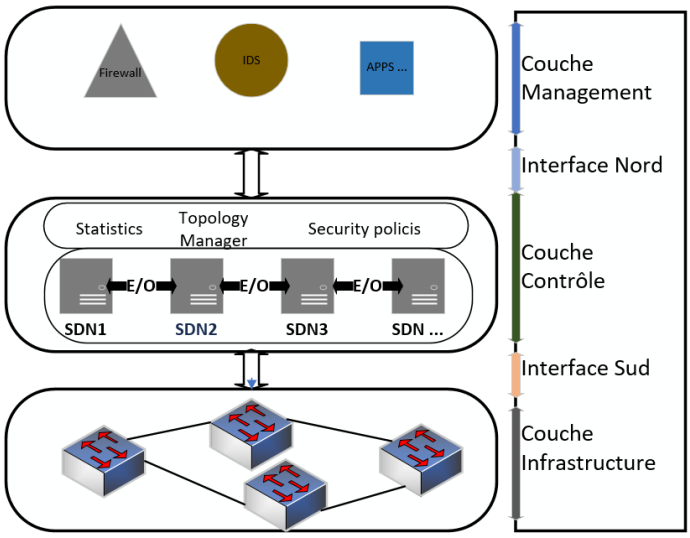


FIGURE 2.1 – Architecture du SDN

Le SDN est une technologie désormais matures, utilisables et « commercialisée » par les opérateurs. Elle permet de centraliser la logique de contrôle dans un contrôleur. Elle permet une séparation du plan de contrôle et du plan de donnée. Selon l'ONF, le SDN est caractérisé par les éléments suivants :

— *Séparation du plan de contrôle et du plan de données* : La fonctionnalité de contrôle est retirée des noeuds du réseau qui deviennent de simples éléments de transfert (de paquets).

— *Les décisions de transfert (de paquets) sont basées sur le flux (et non sur la destination)* :
Un flux est une séquence de paquets entre une source et une destination. Tous les paquets d'un même flux subissent des politiques de service et de traitement identiques au niveau des dispositifs de transfert.
— *La logique de contrôle (l'intelligence) est transférée vers une entité externe, le contrôleur SDN ou Network Operating System (NOS)*. Le NOS est une plate-forme logicielle qui s'appuie sur la technologie des serveurs de base et fournit les ressources et les abstractions nécessaires pour faciliter la programmation des dispositifs de transfert sur la base d'une vue abstraite du réseau et d'une centralisation logique. Son but est donc similaire à celui d'un système d'exploitation traditionnel.
— *Le réseau est programmable* : Le réseau est programmable au moyen d'applications logicielles s'exécutant sur le NOS qui interagit avec les dispositifs du plan de données sous-jacent. Il s'agit là d'une caractéristique fondamentale du SDN, considérés comme sa principale plus-value.

Dans cette section consacrée à la présentation du SDN, nous introduirons son architecture telle qu'elle a été proposée par l'ONF, son principal promoteur, puis nous présenterons en détails les différentes couches de l'architecture, les différentes interfaces (nord, sud, est/ouest) ainsi que les principaux protocoles de communications entre les différentes couches ou entre les différents niveaux d'une même couche (la couche contrôle par exemple).

2.2.1 Les fonctions de base du SDN

Comme nous l'avons mentionné précédemment, le SDN sépare le plan de données et le plan de contrôle. En d'autres termes, l'intelligence du réseau est transférée à un contrôleur, tous les calculs y sont effectués et de nombreuses applications et fonctionnalités peuvent y être ajoutées au besoin comme présenté à la figure 2.1.

Dans [17], les chercheurs ont discuté des modules de base d'un contrôleur SDN. Ils concluent que les modules de découverte de liens, de la gestion de topologie, du stockage, de stratégie, de la gestion des tables de flux et de données de contrôle sont les modules de base du contrôleur SDN. La gestion de la topologie est une des fonctions essentielles et critiques de l'architecture. Elle est assurée par deux modules qui fournissent également le service de routage : le gestionnaire de topologie et le gestionnaire de routage et de découverte des liens. Nous présenterons plus en détails ces fonctionnalités dans le chapitre 4.

2.2.2 La couche application

La couche application se compose principalement des applications métier destinées à l'utilisateur final qui consomme les services SDN et de réseau [18]. Des exemples de ces applications orientées entreprise incluent la visualisation du réseau et les applications de sécurité.

2.2.3 La couche contrôle

Un contrôleur est un élément critique dans une architecture SDN car il est la pièce maîtresse de la logique de contrôle (applications) pour générer la configuration réseau sur la base des politiques définies par l'opérateur réseau. Semblable à un système d'exploitation traditionnel, la plate-forme de

contrôle résume les détails de niveau inférieur de la connexion et de l'utilisation de l'ordinateur. L'interaction avec les dispositifs d'acheminement (c.-à-d. la matérialisation de l'information et les politiques de réseau). Trois interfaces de communication permettent aux contrôleurs d'interagir : les interfaces Sud, Nord et Est/Ouest. Ces interfaces seront présentées en détails dans cette section.

Les fonctionnalités du plan de contrôle comprennent généralement :
— Découverte et maintenance (mise à jour) de la topologie
— Sélection et instanciation de l'itinéraire des paquets
— Mécanismes de basculement de trajectoire

2.2.4 La couche infrastructure

La couche infrastructure se compose principalement d'éléments de transfert incluant des commutateurs et des commutateurs virtuels, tels que Open vSwich [19] ou physiques, tels que *Juniper Junos MX-series* [20]. Ces commutateurs sont accessibles via une interface ouverte pour commuter et transférer des paquets.

2.2.5 La couche management

La couche maagement assure le monitoring, la configuration, la mise en service et la maintenance des équipements du réseau. Elle regroupe l'ensemble des applications déployées dans l'infrastructure du réseau. Elle permet de mettre en place toutes ou partie des règles de transfert en une seule fois, même si une action doit nécessairement s'effectuer avec de nombreuses précautions.

2.2.6 Interface nord

L'interface nord assure la communication entre la couche application et la couche contrôle de l'architecture SDN. Elle est constituée d'un ensemble d'API (Application Programming Interface) qui constituent des éléments critiques de l'architecture. Cette interface doit permettre de supporter une large variété d'applications telles qu'Openstack ou encore CloudStack indispensables aujourd'hui pour le management du Cloud. A ce jour, l'ONF n'a normalisé aucun protocole pour cette interface, cependant le protocole REST (Representational State Transfer) est largement utilisé et est implémenté par la plupart des contrôleurs.

2.2.7 Interface sud

C'est l'interface qui est utilisée par la couche contrôle (CC) pour interagir avec la couche infrastructure (CI). Le protocole OpenFlow [3], proposé par l'ONF, est le principal protocole de cette interface. Cet organisme le met régulièrement à jour en y ajoutant très souvent de nouvelles fonctionnalités, la dernière version, au moment de la rédaction de ces lignes, était la version *1.5.1*. Un dispositif supportant le protocole OpenFlow dispose de tables appelées « tables des flots ». Ces tables contiennent des règles appropriées à chaque flot.

Plusieurs autres protocoles tels que NETCONF [21, 22], BGP [8], Interface to Routing System (I2RS) [23] sont proposés par d'autres organismes comme l'Internet Engineering Task Force (IETF)

Vesion	Date	Champs d'entête
OF 1.0	Déc. 2009	12 champs (Ethernet, TCP/IPv4)
OF 1.1	Fév. 2011	15 champs (MPLS, inter-table-metadonneés)
OF 1.2	Déc. 2011	36 champs (ARP, ICMP, IPv6, etc.)
OF 1.3. OF 1.4	Juin 2012 Oct. 2013	40 champs (Contrôleurs multiples) 41 champs
OF 1.5	Déc. 2014	45 champs

TABLE 2.1 – Champs reconnus par le standard OpenFlow

comme des alternatifs à OpenFlow pour les communications entre l'interface sud et la couche infrastructure. Certains de ces protocoles sont particulièrement pensés pour assurer la phase transitoire entre les réseaux actuels vers un réseau SDN complet.

Nous allons présenter dans cette section les principaux protocoles proposés pour la communication au niveau de l'interface sud de l'architecture SDN en mettant un accent particulier sur ceux qui sont les mieux adaptés aux réseaux hybrides SDN. Ainsi, nous présenterons tout d'abord le protocole Openflow le plus connu d'entre eux, puis le protocole NETCONF, ensuite le protocole P4 [24, 25] présenté comme la version 2.0 du protocole OpenFlow ou tout simplement son successeur et enfin nous présenterons le protcole I2RS. Nous présenterons par ailleurs dans la section 2.8 les principaux organismes participant à la standardisation du SDN.

2.2.7.1 Le protocole OpenFlow

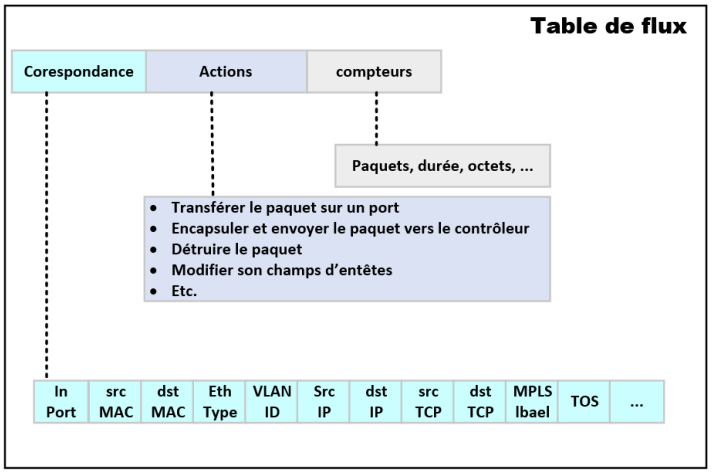


FIGURE 2.2 – Éléments d'une table de flux (« flow »)

OpenFlow est le protocole le plus de connu du SDN. Il est le principal protocole de son interface sud, c'est également la toute première implémentation réelle des concepts de l'architecture du SDN. Proposé à son lancement par l'Université de Stanford, il est désormais sous la responsabilité de l'ONF qui en assure l'évolution et la standardisation. Dans cette section, nous présentons un bref aperçu du

protocole OpenFlow.

Composant du protocole OpenFlow

Le protocole OpenFlow peut être divisé en quatre composants ou niveaux appelés aussi couches (« Layer ») :

- **Couche message :** Cette couche est le cœur de la pile de protocoles. Il définit une structure et une sémantique valable pour tous les messages. Une couche de messages typique permet de construire, copier, comparer, imprimer et manipuler des messages.
- **Couche machine d'état :** La machine d'état définit le comportement de bas niveau du protocole. Typiquement, il est utilisé pour décrire des actions telles que : la négociation, la découverte des capacités, le contrôle du flux, la transmission, etc.
- **Interface système :** Cette interface définit comment le protocole interagit avec le monde extérieur. Il identifie généralement les interfaces nécessaires et optionnelles ainsi que leur utilisation prévue, telles que l'activation du Transport Layer Security (TLS).
- **Couche configuration :** Presque tous les aspects du protocole ont des configurations ou des valeurs initiales. La configuration peut tout couvrir, des tailles de tampon par défaut ou des intervalles de réponse jusqu'aux certificats *X.509*.
  Le protocole OpenFlow définit trois type de communications : La communication d'un commutateur à un autre, la communication d'un commutateur vers le contrôleur (plan de contrôle) et la communication du contrôleur (couche contrôle) vers les commutateurs. Ces différents types de communications permettent la détection des fonctionnalités, la configuration, la programmation et la collecte des informations de configurations des commutateurs et de la notification du contrôleur (plan de contrôle) de l'arrivée d'un nouveau flux pour effectuer le changement de l'état d'un commutateur (*PORT-STATUS*).

Fonctionnement du protocole

A l'entrée d'un nouveau paquet, un commutateur OpenFlow, effectue une vérification dans la tables de flots : l'existence d'une règle de correspondance à partir d'un champ de l'entête du paquet ; c'est le champ « matching ». La figure 2.2 présente les composants d'une table de flux (« Flow ») et le mécanisme appliqué pour le traitement des paquets arrivant au niveau des commutateurs. Ce mécanisme est basé sur le principe *Correspondance-Actions-compteurs*. La correspondance est établie sur la base des règles installées (par défaut ou par l'arrivée d'un nouveau flot) par le contrôleurs. Ces règles concernent par exemple les adresses *MAC/IP* sources et destinations, le port Transmission Control Protocol (TCP), le numéro de *VLAN* ou encore l'étiquette *MPLS* (OpenFlow 1.1.0 et suivants). Le nombre de champs de l'entête (de correspondance) a beaucoup évolué depuis la première version du protocole (une présentation est effectuée dans le tableau 2.1). Pour chaque paquet arrivant au niveau d'un commutateur, la règle appliquée dans le cas où aucune règle ne correspond est d'effectuer les actions/instructions définies par défaut. Les cas de figures les plus standard sont la destruction (*DROP*) et l'acheminement vers un port de sortie spécifique. En l'absence de toute correspondance, le paquet peut être encapsulé et transmis, via un message *Packet-IN*, au contrôleur SDN qui va alors définir les actions à appliquer sur ce paquet ainsi que sur les paquets suivants du même flot. Notons que ce processus de transmission vers le contrôleur est défini comme une règle par défaut par la plupart des contrôleurs. Dans le cas

contraire, le paquet est rejeté. Enfin, le compteur statistique est mis à jour dans la table des flots après l'application des instructions. Ces statistiques sont ensuite transmises vers le contrôleur SDN et vers les applications de monitoring, s'exécutant dans le plan de gestion.

2.2.7.2 Le protocole NETCONF

Network Configuration Protocol (NETCONF) est un protocole pour le management du réseau. Il a été proposé par l'IETF comme un standard dans le *RFC-4741* [21]. NETCONF propose et définit un mécanisme lui permettant d'installer, de manipuler, de supprimer ou mettre à jour les configurations des équipements réseaux. Il utilise le langage de balisage Extensible Markup Language (XML) pour encoder les données de configuration. Le protocole de transport utilisé par NETCONF est le protocole Secure Shell Transport Layer Protocol (SSH) pour assurer l'intégrité et la sécurité des opérations définies dans les messages *XML*.

NETCONF utilise un système d'appel de procédures à distance Remote procedure call (RPC) basé sur des sessions de connexion TCP sécurisées. Le client se connecte et envoi des messages (requêtes) encodés en *XML* au serveur qui exécute les opérations contenues dans la requête et renvoie par la suite une réponse également codée en *XML*. Cela permet d'avoir des échanges cohérents entre les deux parties.

La figure 2.3 présente les différentes couches du protocole NETCONF :

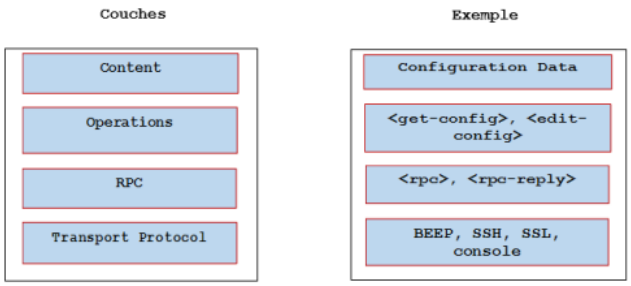


FIGURE 2.3 – Les couches du protocole NETCONF

— **Le protocole de transport :** C'est l'ensemble des protocoles et mécanismes utilisés pour faire communiquer le serveur avec le client, chacun offre ou assure des fonctionnalités particulières.
— **RPC :** Cette couche implémente le mécanisme d'appel de procédure à distance
— **Les opérations** ce sont toute les manipulations qu'il est possible de faire sur les équipements, elle sont invoquées par des méthodes RPC avec des paramètres spécifiques à chacune encodées en *XML*.
— **La couche Content** : Définit la manière dont les données sont représentées au niveau applicatif. Un langage de modélisation de données spécifie la manière dont ce contenu est structuré.

Opérations de base

Une fois que la session est établie entre le client et le serveur avec un échange des fonctionnalités supportées par les deux parties, le client peut alors envoyer des requêtes RPC contenant les opérations à effectuer par le serveur. Ces opérations sont responsable du changement ou de la consultations des configurations présentes sur le dispositif, elles permettent aussi de manipuler les ressources dont dispose cet équipement.

Nous présentons dans le tableau 2.2 les principales opérations utilisées par NETCONF.

Opérations	Description
get	Récupérer les informations de configuration à partir de la *running database* ou des statistiques
get-config	Récupérer les informations de config. à partir de la *running database*
edit-config	Modifier les configurations dans la base de donnés
copy-config	Copier les configurations
delete-config	Supprimer les configurations
commit	Commit du contenu de la config. de la base de données <*candidate/*> vers <*running/*> *database*
lock/ unlock	Bloquer/ débloquer l'écriture sur la base de données par d'autres sessions
validate	Valider tout le contenu de la base de données
close-session	Fermer la session active
kill-session	Fermer d'autres sessions

TABLE 2.2 – Les opérations de base du protocole NETCONF

Configuration des données (Data Configuration

NETCONF manipule au niveau de la couche contenu des données qui représentent les configurations de l'équipement. Ces données sont toutes les ressources dont dispose le dispositif telles que les interfaces réseau, les adresses IP, les noms utilisateur autorisés à accéder à ce dispositif, etc. NETCONF ne spécifie pas la manière dont sont représentées ces données. C'est un autre outils qui permet de définir et de modéliser ces données qui sont manipulées et traitées par NETCONF. Il s'agit du langage de modélisation de données YANG [26, 27] qui a été par l'IETF pour réaliser cette tâche.

2.2.7.3 Programing Protocol-Independent Packet Processor (P4)

En plus de la séparation du plan de contrôle et du plan de données, le SDN est aussi caractérisé par sa programmabilité. En d'autres termes, cela veut dire que les équipements qui constituent l'infrastructure du réseau sont programmables par le biais d'un protocole de communication. OpenFlow s'est imposé comme le principal standard depuis l'émergence du SDN. Cependant, avec l'apparition de nouvelles technologies, notamment le NFV et face à la forte demande de services de plus en plus innovants et exigeants, plusieurs spécifications sont ajoutées régulièrement au protocole OpenFlow par l'ajout de nouveaux champs d'entête (MPLS, ARP, ICMP ...) pour que les données issues de ces techniques puissent être prises en compte par le plan de contrôle. Un problème qui apparaît à chaque intégration d'un nouveau protocole provient de l'ajout de nouveaux champs d'entête à OpenFlow, ce

qui peut s'avérer être une lourde tâche à répétition. Le tableau 2.1 présente les différentes spécifications du protocole OpenFlow ainsi que l'évolution du nombre de champs de l'entête pour chaque version.

Afin de résoudre ce problème, une solution a été proposée de telle façon à permettre aux commutateurs d'avoir des mécanismes flexibles qui puissent automatiser la tâche de reconnaître des paquets issus de différentes technologies et de pouvoir procéder aux opérations qu'il faut pour tel ou tel type de paquet, et cela en utilisant le protocole OpenFlow sans avoir à redéfinir ses spécifications.

P4 vise à répondre à ce défis, on parle aussi de *SDN 2.0*. C'est un langage de programmation de haut niveau utilisé pour configurer des commutateurs et les informer des opérations à appliquer aux paquets transitant dans le réseau. Il est basé sur un modèle de traitement *match+action*. P4 offre un niveau d'abstraction pour le plan de contrôle pour ce qui concerne la configuration des commutateurs, il sert d'interface entre le plan de contrôle et la couche infrastructure. P4 ne vise pas à remplacer OpenFlow, il coexiste avec lui. La figure 2.4 présente son architecture générique avec ses principaux composants. Nous présenterons plus loin dans cette section le rôle de chaque composant. OpenFlow s'occupe d'acheminer et de remplir les tables des flots des commutateurs avec les règles OpenFlow correspondant aux paquets. Donc, P4 offre à OpenFlow des mécanismes pour interagir avec différents commutateurs n'ayant pas le même mode de fonctionnement, pour s'adapter au type de données transitant dans le réseau et également de permettre au plan de contrôle d'opérer avec des équipements sans les contraintes liées aux caractéristiques techniques de ces derniers.

P4 vise à garantir les points suivants :

— **Reconfiguration** : Le contrôleur doit pouvoir redéfinir la manière dont les paquets sont analysés et traités par le plan de données.
— **Indépendance des protocoles** : Le commutateur ne doit pas être contraint par le format ou le type du paquet qu'il traite, l'analyse et le traitement doivent être appliqués aux paquets indépendamment du protocole utilisé pour son acheminement. Tout d'abord le commutateur extrait l'entête du paquet pour l'analyser et un ensemble d'actions seront appliquées selon la correspondance des informations extraites (utilisation de tables type *match+actions*).
— **Indépendance des équipements** : Le programmeur n'a pas besoin de connaître les détails techniques et les mécanismes utilisés par le commutateur pour le traitement des données. C'est le compilateur qui transforme le programme écrit en P4 en un programme tenant compte des caractéristiques de l'équipement afin qu'il puisse être exécuté sur ce dernier.

En résumé, P4 est un langage qui permet de repousser les limites d'OpenFlow en le rendant plus flexible par rapport aux paquets et au type de données qu'il gère. Cela permet d'assurer une programmation du plan de contrôle plus facile et plus dynamique.

Modèle d'acheminement des données

Dans ce modèle, le commutateur utilise un programme *P4* pour analyser les données et les tables *match+action* pour leur traitement. Ce fonctionnement peut être représenté par trois principes généralisés : en premier lieu, ce modèle se base sur un analyseur (*parser*) programmable qui s'adapte selon le type de paquet auquel il a à faire et aussi offre la possibilité de définir de nouveaux en-têtes afin qu'il puisse être capable de les reconnaître. En second, les traitements appliqués aux données par la table

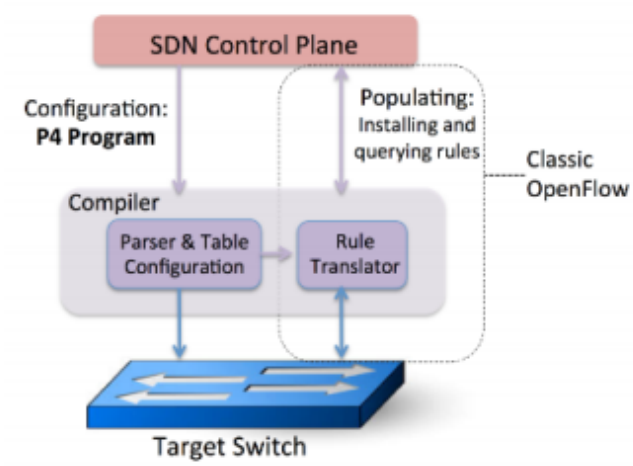


FIGURE 2.4 – Architecture de P4

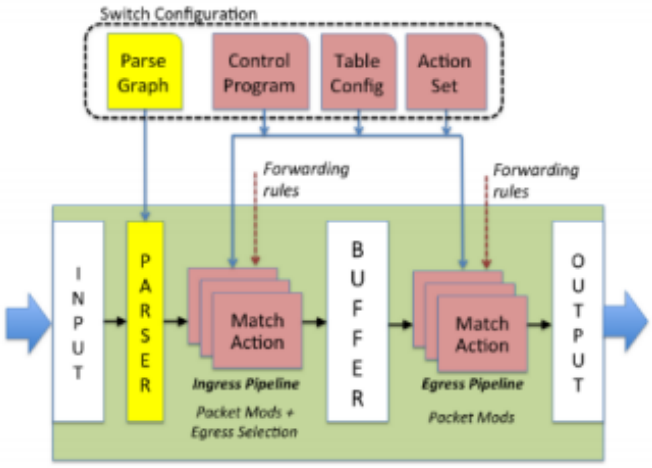


FIGURE 2.5 – Le modèle de transfert abstrait

match+action peuvent être exécutés soit en série soit en parallèle. En dernier, le modèle implémente des primitives indépendantes des protocoles à utiliser et elle sont supportées par les commutateurs pour effectuer les actions nécessaires.

La figure 2.5 montre le modèle utilisé par P4 pour procéder à l'analyse et le traitement des paquets. Ce modèle fait appel à deux opérations majeures : *Configure* (configurer) et *Populate* (peupler).

Les opérations de type *Configure* servent à programmer le *parseur* (analyseur), ordonnancer les tables match+action et spécifier quel champ de l'en-tête *(header)* à prendre en compte à chaque étape du processus, ainsi cela permet de savoir quel protocole est supporté et comment acheminer les paquets par le plan de données. Les opérations de type *populate* quant à elles ajoutent, suppriment ou modifient

les entrées des tables *match+action* et déterminent les politiques à appliquer aux paquets.

Comme le montre la figure 2.5, lorsque des paquets arrivent, ils sont tout d'abord gérés par le *parseur*, ils peuvent êtres séparés en les mettant dans des tampons (*buffers*) pour un traitements ultérieure. Le *parseur* extrait alors les champs d'en-tête et détermine le protocole supporté par le commutateur pour procéder au traitement de ces paquets. Ensuite, ces champs *(fields)* seront envoyés aux tables *match+action* qui sont divisées entre les entrées *ingress* et *egress* du commutateur où ces tables peuvent être modifiées. Au niveau *ingress*, le *parseur* détermine le port de sortie vers *egress* et met les paquets en file d'attente, ainsi les paquets peuvent être directement acheminés, dupliqués (multicast, vers plan de contr^ole) ou simplement rejetés. Au niveau du *egress match+action* les champs d'entête des paquets peuvent être modifiés instantanément selon l'action spécifiée. Le paquets peuvent aussi transporter des informations additionnelles appelées *metadata* qui informent sur le processus d'acheminement du port en entrée (*ingress*) vers le port de sortie (*egress*). Par exemple, il peut s'agir de données échangées entre les tables comme les adresses source ou destination, les ports, des labels rajoutés au paquets, identifiants virtuels, etc.

Le langage de programmation P4

P4 utilise dans son implémentation des déclarations de type *header* permettant ainsi au programmeur de définir autant de types de *header* qu'il souhaite pour être pris en considération par le *parseur*, aussi il lui permet de définir la manière dont ces en-têtes sont traités, par exemple : décrémenter le champ défini, vérifier le *Cheksum*, ajouter des labels, etc. Ces actions à appliquer sont définies par les tables *match+action* et elles sont implémentées en utilisant des primitives que P4 propose. P4 prend aussi en compte les dépendances qui peuvent exister entre les champs d'en-tête qui servent à informer le *parseur* quelles tables il peut exécuter en parallèle et comment optimiser ainsi le traitement, ces dépendances peuvent tre résolues en examinant les tables de graphe de dépendances (Table Dependecy Graphs (TDG)) qui décrivent les entrées de la tables et les flots de contrôle passant d'une entrée de table à une autre.

Traitement des paquets (Packet processing)

P4. Le traitement des données définit les composants suivants pour le traitement des paquets :

— **Headers :** La déclaration des *headers* spécifie le format d'en-tête du paquet et les champs qu'il contient avec la taille de chaque champ *(fields)* en bits.
— **Parsers :** Le *parseur* sert à détecter de quel type de *header* il s'agit et l'analyse en vérifiant les champs qu'il contient.
— **Tables :** Les tables *match+action* servent à traiter les paquets, faire la correspondance des champs auxquels on applique les actions.
— **Actions :** Ce sont des méthodes à définir qui utilise les primitives proposées par P4 afin de manipuler les champs du *header*.
— **Control :** Le programme de contrôle détermine l'ordre des tables *match+action* qui sont appliquées aux paquets.

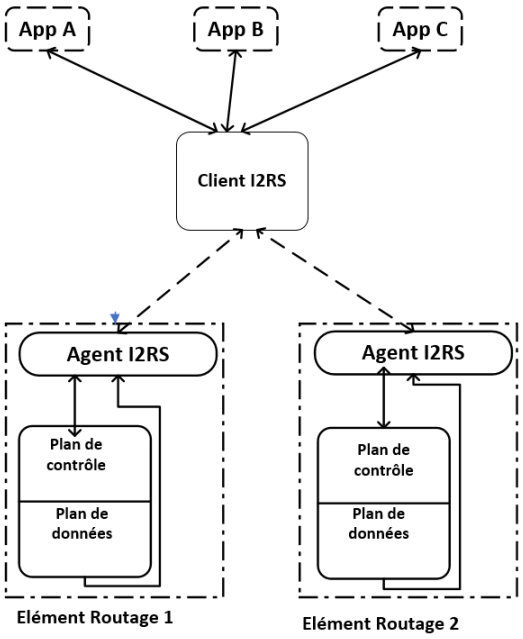


FIGURE 2.6 – Architecture du Client et de l'Agent I2RS *(IETF)*

2.2.7.4 I2RS (Interface to Routing System)

Les routeurs constituent l'infrastructure de routage de l'Internet. Ils disposent de diverses informations dont l'état des réseaux comme par exemple, la Routing Information Base (RIB). Ils implémentent également les différents protocoles de routage tels que OSPF, IS-IS ou encore BGP. Ces informations sont utilisées pour échanger des informations sur la topologie, l'état ainsi que d'autres informations par rapport au réseau.

I2RS est une interface programmatique qui permet le transfert des états des éléments de routage en dehors du système de routage.

I2RS est un protocole de l'interface sud proposé par l'IETF. OpenFlow est le principal protocole normalisé par l'ONF et il n'est pas compatible avec les protocoles distribués des réseaux *legacy*. Ce protocole basé sur une architecture centralisée qui sacrifie de facto les avantages offerts par les protocoles distribués utilisés par les réseaux traditionnels. La figure 2.6 présente son architecture simplifiée. Elle repose sur trois composants : Le Client I2RS ou le contrôleur, l'Agent I2RS et le protocole de communication entre le Client et l'Agent.

Client ou Contrôleur I2RS

Le Client I2RS ou le contrôleur a pour rôle de centraliser toutes les informations définissant l'état du réseau. Il doit interagir de façon sécurisée avec le système de routage à travers les Agents I2RS. A la différence du SDN, le plan de contrôle n'est plus centralisé, il reste au sein du dispositif de routage

(commutateur ou routeur). Il intervient à la suite des événements bien identifiés (la défaillance d'un lien, la charge Central Processing Unit (CPU) d'un routeur, le taux d'occupation d'un lien, la présence d'une attaque DDoS, ...) pour modifier (ajouter ou supprimer une route sur un routeur par exemple, ...). Le contrôleur supplante de façon temporaire les protocoles distribués (IS-IS, OSPF, ...).

Agent I2RS

Le rôle de l'Agent consiste à collecter puis fournir les informations relatives au système de routage dont a besoin le contrôleur (Client I2RS). Il est associé à un commutateur/routeur. Il interagit avec l'élément pour recueillir ses états (charge CPU, statuts des interfaces, etc). Les informations sont ensuite remontées par le biais d'un canal sécurisé au contrôleur qui agit en fonction de la situation.

Protocole de communication

La communication entre le Client et l'Agent I2RS se fait par le biais d'un canal sécurisé pour éviter toute attaque qui pourrait empoisonner la connaissance des routeurs. L'IETF recommande l'utilisation du protocole Netconf comme protocole de transport.

2.2.8 Interface EST/ OUEST

L'architecture SDN classique n'utilise qu'un seul contrôleur pour centraliser le contrôle et la gestion du réseau. Nous appelons cette architecture le *« single mode »* (mono-contrôleur) pour le distinguer du *« cluster mode »* (multi-contrôleurs) dans lequel plusieurs contrôleurs SDN collaborent pour améliorer la disponibilité et l'évolutivité (le passage à l'échelle) du plan de contrôle. Le mode multi-contrôleur (*« cluster mode »*) est aussi appelé architecture distribuée dans le paradigme SDN. Chaque contrôleur dans le « *cluster* » est appelé une *instance*. Toutes les instances du « cluster » ont la même vue du réseau en termes de nœuds actifs et de liens. La cohérence des données partagées entre les instances du« cluster » est garantie par le protocole de gestion de consensus. Le protocole RAFT [28] est le plus utilisé dans le cas du SDN. Cependant, pour la collecte des données ainsi que le rôle de chaque contrôleur vu par le périphérique géré, la collaboration entre les instances peut suivre l'une des deux configurations, *maître-esclave* et *équilibrage de charge*.

Dans la configuration maître-esclave, une instance joue le rôle de *Master* (*Maître*) et les autres jouent le rôle de *Slave* (*Esclave*). Tous les périphériques réseau sont affectés au contrôleur maître et sont contrôlés par cette instance. Les autres instances servent de contrôleurs de secours. Lorsque les contrôleurs esclaves du « cluster » détectent une défaillance du contrôleur maître, le contrôleur principal, désigné par le protocole *RAFT*, prend automatiquement le relais. C'est la configuration par défaut utilisée par ODL lorqu'il est déployé en mode « cluster ».

Dans une configuration d'équilibrage de charge (« load-balancing »), chaque instance contrôle une partie égale du réseau par rapport aux autres instances membres. Les périphériques réseau sont équitablement affectés entre les instances du « cluster » qui joue le rôle de contrôleur maître tandis que les autres jouent le rôle de contrôleurs esclaves. Cette configuration peut être facilement déployée avec la commande `balance-masters` du contrôleur ONOS. Lorsqu'un nouveau commutateur rejoint le réseau, il sera automatiquement affecté à l'instance la moins chargée. Cette instance devient le contrôleur

maître et les autres instances deviennent les contrôleurs esclaves de ce commutateur.

Chaque commutateur du réseau peut être connecté à un ou plusieurs instances. Par conséquent, un commutateur doit avoir un et un seul *Master* qui permet de gérer et de contrôler ses évènements tels que l'état de ses ports, l'envoi d'un paquet, etc. Pour cela ONOS utilise différents algorithmes de gestion de la consistance (cohérence).

L'environnement multi-contrôleurs (*cluster*) promet plusieurs avantages, mais il apporte également un certains nombre de défis tels que le nombre de contrôleurs à déployer et leur emplacement. L'approche architecturale sera présentée dans cette section.

2.2.9 Architecture distribués du SDN

Il existe principalement deux modèles d'architecture distribuée du SDN, dont nous proposons une présentation des modèles.

2.2.9.1 Plan de contrôle logiquement centralisé

Cette approche consiste à déployer un plan de contrôle physiquement distribué mais logiquement centralisé en utilisant plusieurs contrôleurs (instances). Cette solution demande une forte synchronisation entre les instances, chaque contrôleur (instances) doit partager toutes les informations relatives aux commutateurs sous son contrôle avec les autres instances pour assurer une meilleure cohérence du système. Cette approche est particulièrement intéressante dans le cadre des « data-centers ».

De nombreuses solutions basées sur cette approche ont été proposées dans la littérature et par plusieurs projets open-source. Les projets ONOS et ODL, qui se sont imposées comme les principales références parmi les solutions open-source, sont basés sur cette approche. D'autres solutions moins populaires et moins récentes (2010) telles que HyperFlow [29], Onix [30] utilisent également cette approche.

Kandoo [31] à la différence des solutions de la même famille présentées précédemment repose sur une distribution hiérarchique des contrôleurs basée sur deux couches : i) la couche inférieure, un groupe de contrôleurs sans interconnexion et sans connaissance de l'état du réseau et (ii) la couche supérieure, un contrôleur logiquement centralisé qui maintient l'état du réseau.

2.2.9.2 Plan de contrôle logiquement distribué

Cette autre approche consiste à déployer un plan de contrôle logiquement distribué. Cette approche est particulièrement adaptée pour le déploiement incrémental du SDN (réseaux hybrides SDN) ou l'interopérabilité entre contrôleurs hétérogènes. Plusieurs travaux sont proposés dans la littérature pour adresser ce problème. Nous présentons les plus intéressant d'entre eux de notre point de vue.

DISCO [32] est basée sur un plan de contrôle SDN ouvert et extensible, capable de faire face à la nature distribuée et hétérogène des réseaux modernes et des réseaux étendus. Avec DISCO, les contrôleurs gèrent chacun leur propre domaine réseau et communiquent entre eux pour fournir des services réseau de bout en bout. Cette communication est basée sur un canal de contrôle unique, léger et

facile à gérer, utilisé par les agents pour partager de manière auto-adaptative des informations agrégées à l'échelle du réseau.

I2RS (Interface to Routing System) [23, 33] proposé par l'IETF par *WG I2RS* [23, 33] s'inscrit également dans la même vision. Ce groupe de travail de l'*IETF* travaille sur architecture avec un plan de contrôle partiellement centralisé et distribué logiquement basé sur un ou plusieurs contrôleurs et des agents associés aux commutateurs/routeurs.

2.3 Interopérabilité entre contrôleurs

La communication entre contrôleurs est effectuée par l'interface *EST-OUEST*. L'interopérabilité suppose la présence simultanée au sein d'un même domaine (système autonome) ou des domaines voisins des contrôleurs hétérogènes communiquant de manière efficace pour exécuter les services réseau. Il n'existe pas de protocole standardisé au moment ou nous rédigeons ces lignes. Cependant, des travaux tels que ceux abordés dans [34] introduisent ce sujet depuis *2012* sans pourtant parvenir à un début de standard jusqu'au moment où ces lignes sont rédigées.

Dans [35], les auteurs proposent un protocole de découverte de topologie distribuée hiérarchique sur l'utilisation d'éléments de calcul de chemins hiérarchiques « Path Computation Element (PCE) » pour l'orchestration de SDN multi-domaine. Le défi de ce protocole selon les chercheurs est de savoir comment les PCE enfants peuvent maintenir une topologie abstraite inter-domaines tout en préservant les informations confidentielles intra-domaines dans une architecture d'orchestration hiérarchique basée sur PCE.

2.4 Nombre et emplacement des contrôleurs multiples

Dans le SDN, les contrôleurs devraient idéalement être placés de manière à optimiser les performances du système. Le nombre et l'emplacement des contrôleurs doivent par exemple préserver un temps de réponse minimal entre les contrôleurs et les nœuds de transmission (les commutateurs) dans le plan de données. Dans un environnement avec des contrôleurs multiples, leur emplacement peut avoir un impact considérable sur les performances du réseau en termes de réactivité.

La présence de plusieurs contrôleurs dans un réseau SDN vise à répondre à l'un des défis majeurs du paradigme de passage à l'échelle. Le problème de placement des contrôleurs concerne principalement la détermination de leur emplacement optimal ainsi que la manière avec laquelle les commutateurs leur sont affectés. Dans cette section, nous présentons les principales approches introduites par la littérature.

Dans [36], les auteurs ont introduit pour la première fois en *2012* le problème du nombre et de l'emplacement des contrôleurs (environnement avec contrôleurs multiples). Pour une topologie donnée, ils ont essayé de répondre aux questions suivantes : i) Combien de contrôleurs sont-ils nécessaires ? ii) Quel emplacement choisir pour chacun des contrôleurs ? Ils se sont particulièrement intéressés à la *latence* de propagation dans un réseau étendu de type WAN (Wide Area Network). Les chercheurs concluent que le nombre et le choix des emplacements dépendent de la topologie du réseau d'une part,

du choix des métriques d'autres part (délai de propagation entre les commutateurs, entre commutateurs et contrôleurs, temps de réponse des contrôleurs,..). Ils ont démontré dans leur étude que pour réduire la latence de *50%* par exemple, cela demande deux contrôleurs supplémentaires et trois dans le pire cas par rapport à un environnement avec un seul contrôleur.

Dans [37], les auteurs adressent le problème du placement des contrôleurs dans un réseau WAN en partitionnant le domaine en plusieurs sous domaines plus petits et placés chacun sous la responsabilité d'un contrôleur. Les chercheurs proposent en plus de la latence dans [36] la prise de compte d'autres métriques tels l'équilibrage de charge et la fiabilité (« reliability ») qui sont également des éléments essentiels dans l'opérationalité du SDN.

Dans [38] les auteurs traitent également du problème du placement et du nombre des contrôleurs dans le SDN, de manière à maximiser la fiabilité des réseaux de contrôle. Les chercheurs ont proposé une métrique pour caractériser la fiabilité des réseaux de contrôle SDN. Sur la base de plusieurs algorithmes de placement introduits dans la littérature, ils ont évalué et proposé une quantification approfondie de l'impact du nombre de contrôleurs sur la fiabilité des réseaux SDN en utilisant des topologies réelles. Ils concluent que leur approche peut améliorer significativement la fiabilité des réseaux de contrôle SDN sans introduire de latences inacceptables et que le meilleur ratio pour trouver le nombre optimal de contrôleurs à déployer. Cette valeur est comprise entre [*0,035n - 0,117n*], où n est le nombre de commutateurs/routeurs.

2.5 Cohérence dans les contrôleurs SDN distribués

Dans l'architecture distribuée du SDN, la Cohérence signifie que toutes les instances du contrôleur disposent de la même vision de l'état de l'infrastructure du réseau (état de disponibilité des nœuds/liens dans leur structure de données), c'est à dire que toutes les instances possèdent à tout instant la même copie de la topologie du réseau en local. Ainsi, tout changement au niveau de chaque instance (contrôleur), suite par exemple à une défaillance d'un lien, doit être immédiatement propagé aux autres instances via le protocole de gestion de la cohérence. Une incohérence de vision sur l'état de l'infrastructure entre les instances peut entraîner la non exécution correcte des politiques de sécurité, de routage, ou encore de qualité de service définies pour le réseau.

Dans la théorie des systèmes distribués plusieurs modèles sont proposés pour la gestion de la cohérence. Nous nous intéresserons à deux d'entre eux qui ont une application directe dans le cas du SDN.

2.5.1 Modèle de Cohérence éventuel (« Eventual Consistency Model »)

Ce modèle offre une Cohérence faible. Une modification des données sur l'une des instances sera *éventuellement* propagée à toutes les autres instances. Cela implique que pendant un certain temps que des instances peuvent ne pas disposer des dernières mises à jours sur les données partagées, mais après un certains temps, elles (les instances) finiront toutes par disposer de la dernière mise à jour dans la mesure où elles sont toujours en mesure de communiquer les unes avec les autres.

Le protocole *Anti-Entropy* est une des implémentations de ce modèle, utilisée dans le SDN. Chaque

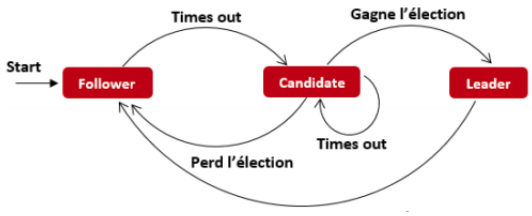


FIGURE 2.7 – Processus d'élection du *leader* avec le protocole *RAFT*

contrôleur (instance) choisit aléatoirement une autre instance à intervalles réguliers (*5 secondes pour ONOS*) à qui il envoie un message contenant la version actuelle (horodatée) de ses données. Le contrôleur choisi compare sa version de données par rapport à celle envoyée par son pair. Dans le cas d'une différence de version, une synchronisation est effectuée sur la base des données horodatées (les données les plus à jour) entre les deux instances. Cela permet aux deux instances de disposer de données mutuellement cohérentes. Cette procédure permet ainsi de garantir que les contrôleurs finiront par disposer de données cohérentes.

2.5.2 Modèle de cohérence forte (« Strong Consistency Model »)

Ce modèle offre une cohérence forte et il garantit que chaque instance dispose toujours des dernières mises à jour sur les données partagées dans un « cluster ».

Le protocole *RAFT* a été pris comme modèle. Il garantit une cohérence forte. Il est implémenté par ONOS et ODL qui sont actuellement deux des contrôleurs les plus populaires parmi la communauté « open-source ».

RAFT est un algorithme de consensus dont l'objectif est de simplifier les algorithmes existants de ce type tel que *PAXOS* [39, 40], qui réutilise les approches de cet algorithme tout en simplifiant ces concepts afin d'avoir un algorithme plus simple. *RAFT* garde les hypothèses fortes de *PAXOS* qui vont simplifier l'aspect théorique tel que le fait d'avoir un *Leader* fort qui gère un ensemble de noeuds.

RAFT définit trois états *Leader*,*Candidate Folower*. Chaque instance membre d'un « cluster » se trouve dans l'un de ces états. En revanche, il ne peut y avoir plus d'une instance avec le statut de leader dans un « cluster ». Le statut d'une instance peut évoluer dans le temps en fonction des évènements (le *leader* ne répond plus par exemple, l'arrivée ou le départ d'une quelconque instance) intervenus dans le réseau. La désignation du *leader* est faite suivant le processus décrit par la figure 2.7.

L'idée de *RAFT* se construit sur le fait d'avoir un temporisateur (« timeout »), le *leader* envoie un *Heartbeat* (un tic d'horloge) régulièrement à chacun des *followers*. Si un *follower* n'a pas reçu de *Heartbeat* au bout d'un certain temps, il va devenir un *candidat* pour une élection qui va démarrée.

2.6 Network Function Virtualization (NFV)

C'est aussi l'une des nouvelles tendances révolutionnaires dans l'industrie des réseaux. Le NFV [41] repose sur une séparation des fonctions réseaux et du matériel sous-jacent en transférant donc

le software vers une machine virtuelle par exemple. Cela est rendu possible notamment grâce aux dernières innovations sur les technologies de virtualisation. Par conséquent, les différentes fonctions réseaux peuvent être déployées sans tenir compte de leur localisation. Elles peuvent se trouver dans des « data-centers » par exemple. En outre, cela devrait permettre de réduire le surplus de coût dû à la part du matériel dans le prix d'achat d'un équipement réseau.

Ces dernières années ont vu la multiplication des équipements connectés à Internet provoquant une augmentation considérable de la demande en matière de trafic. Ceci exige des extensions physiques du réseau de la part des opérateurs, ce qui se traduit notamment par une augmentation des dépenses en investissement CAPEX et des dépenses opérationnelles OPEX.

2.6.1 Architecture du NFV

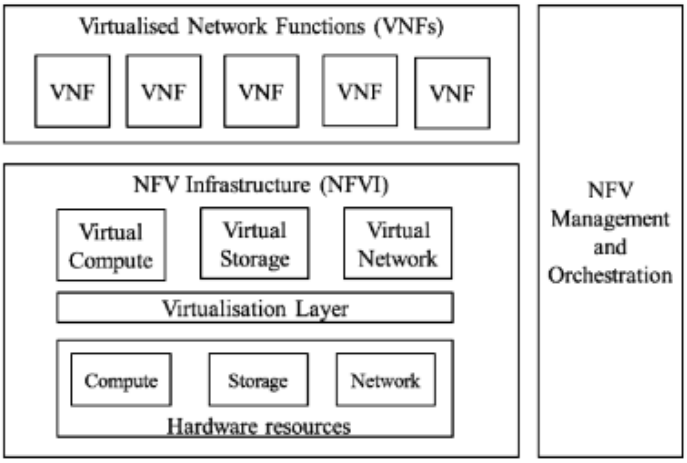


FIGURE 2.8 – Architecture de référence du NFV proposée par *ETSI* [1]

Dans *GS NFV 002* [1], l'European Telecommunications Standards Institute (ETSI) a élaboré un cadre architectural de référence et un cahier des charges à l'appui de la gestion et de l'orchestration des NFV. Cette architecture est représentée dans la figure 2.8. Nous introduirons dans cette section les composants et le rôle de chaque élément de cette architecture qui couvre également l'orchestration et la gestion du cycle de vie des ressources physiques et virtuelles.

2.6.1.1 Element Management (EM)

Composant responsable des fonctions de gestion du réseau Fault, Configuration, Accounting, Performance, and Security (FCAPS) d'un Virtualized Network Function (VNF) en fonctionnement.

2.6.1.2 VNF

Bloc fonctionnel représentant la fonction de réseau virtualisé mise en œuvre sur un serveur physique. Par exemple, un VNF routeur, un VNF commutateur OpenFlow, un VNF Firewall, etc.

2.6.1.3 NFV Infrastructure (NFVI)

Représente l'ensemble des composants matériels (calcul, stockage et réseau) et logiciels où les VNF sont déployés, gérés et exécutés.

2.6.1.4 Network Function Virtualization Orchestrator (NFVO)

C'est la composante principale, en charge de l'orchestration des ressources NFV Infrastructure (NFVI) sur plusieurs Virtualised Infrastructure Manager (VIM) et de la gestion du cycle de vie des services réseau.

2.6.1.5 VNF Manager (VNFM)

Exécute la configuration et la gestion du cycle de vie du VNF (p. ex. instanciation, mise à jour, interrogation, mise à l'échelle, terminaison) sur son domaine.

2.6.1.6 VIM

Bloc qui permet de contrôler et de gérer les ressources NFVI ainsi que l'interaction d'un VNF avec les ressources matérielles. Par exemple, OpenStack comme plate-forme Cloud et OpenDaylight et ONOS comme contrôleurs SDN.

2.6.2 Orchestration des NFV

Le terme orchestration est utilisé dans de nombreux domaines tels que le multimédia, la musique, les architectures orientées services Service-Oriented Architecture (SOA), le Cloud, le SDN et plus récemment, le NFV. Dans cette section nous allons présenter la définition de ce concept par deux des principaux organismes qui travaillent au développement et à la standardisations des NFV.

Selon la vision du National Institute of Standards and Technology (NIST) [42], l'orchestration est un processus lié à l'arrangement, la coordination et la gestion de l'infrastructure virtualisée pour fournir différents services Cloud aux clients.

L'ETSI quant à elle dans [43], définit l'orchestration comme un ensemble de processus coordonnés qui automatisent la gestion et le contrôle des systèmes d'information pour atteindre un objectif commun. l'ETSI souligne en outre que l'orchestration pourrait être fournie en plusieurs blocs fonctionnels, sans primauté sur les uns sur les autres.

L'IETF propose une définition très proche de celle de l'ETSI.

2.6.3 Projets et plate-formes NFV

Ces dernières années, une multitude de plates-formes de déploiement et d'expérimentation NFV open-source ont émergé, apportant de nouvelles fonctions pour le plan de données, de contrôle et de gestion. Ces plates-formes modifient considérablement les architectures réseau existantes. Nous présenterons dans cette section les plus pertinentes d'entre elles de notre point de vue.

2.6.3.1 OPNFV

Open Platform for Networks Functions Virtualization (OPNFV) [44] est un projet open-source fondé, mis en place et hébergé par la fondation linux (Linux Foundation), et composé de fournisseurs d'équipements et de Fournisseur d'Accès à Internet (FAI). L'objectif est d'établir une référence open-source intégrée, de qualité opérateur, qui peut être utilisée pour valider des solutions NFV interopérables multi-fournisseurs.

OPNFV vise à valider les spécifications des normes existantes, d'apporter des améliorations aux projets open source pertinents en amont et de développer les nouvelles fonctionnalités nécessaires dans les projets OPNFV et en amont. En particulier, il se concentre sur la mise en œuvre des exigences du NFV [45] fournies par l'ETSI. La première *release (A)rno* est sortie en *juin 2015* et elle était à la version 8.0 *release (H)unter* au moment de la rédaction ces lignes.

2.6.3.2 OpenMANO

OpenMANO [46, 47] est un projet open source mené par *Telefonica*, qui vise à mettre en œuvre le groupe de travail ETSI NFV MANO et à traiter les aspects liés aux performances et à la portabilité des fonctions réseau virtualisées.

Son architecture est composée de trois modules logiciels :

— **Openmano (composant clé)** : C'est une implémentation de la référence de l'orchestration des NFV (Network Functions Virtualisation Orchestrator (NFV-O)) tel que le définit l'ETSI, qui permet la création de scénarios de réseaux virtuels complexes. Il s'interface avec un VIM pour NFV par le biais de son Application Programming Interface (API) et offre une interface vers le nord, basée sur REST, où des services NFV sont offerts, y compris la création et la suppression de services de réseau ou VNF.

— **Openvim :** C'est une implémentation de la référence *NFV-PER001* [43] de l'ETSI, avec support pour des performances élevées et prévisibles. OpenVIM s'interface avec les nœuds de calcul de l'infrastructure et un contrôleur SDN pour offrir des capacités de calcul et de mise en réseau pour déploiement des machines virtuelles. Il offre aussi une interface OpenStack[48] *(OpenVim API)*, où des services Cloud améliorés sont offerts, y compris la création, la suppression et la gestion des images, des saveurs, des instances et des réseaux.

— **Openmano-gui** : C'est l'interface graphique (web) pour interagir avec l'API *Openmano* d'une manière graphique, une interface en ligne de commande est également disponible.

2.6.3.3 ONAP

Open Network Automation Platform (ONAP) [49], c'est un autre projet open-source également soutenu par la fondation Linux. C'est le résultat de la fusion de deux autres initiatives open-source sur le Management Orchestration (MANO) *OPEN-O* et *OpenECOMP*. La première version (*Release 1.0.0 Amsterdam*) a été publiée en Novembre 2017, la version actuelle (premier semestre 2019) est la *release Dublin*.

C'est une plate-forme complète pour l'orchestration et l'automatisation en temps réel des fonctions de réseaux physiques et virtuels, qui permettra aux fournisseurs et développeurs de logiciels, de

services réseaux et de services Cloud d'automatiser rapidement de nouveaux services et de prendre en charge la gestion du cycle de vie complet.

Il existe plusieurs autres projets et plates-formes tels que CloudNFV [50], X-MANO [51], XOS [52] dont l'objectif est de proposer une seule fonction réseau virtualité comme un routage [53], un Deep Packet Inspection (DPI) [54] ou encore un contrôle d'accès [55].

2.7 Relation entre SDN, NFV et Cloud Computing

Ces trois technologies constituent les principales révolutions de ces deux dernières décennies dans le monde des réseaux tant pour les académiques que pour les industriels de toute nature (fournisseurs d'équipements, FAI) et autres fournisseurs multi-services tels que les Google, Amazon, Facebook, Apple et Microsoft (GAFAM). Elles constitueront sans nul doute le socle de base de toutes les évolutions et révolutions actuelles et futures telles que la *BockChain* et le *big data*. Le principal avantage de ces technologies résident dans leur complémentarité. Le Cloud permet le partage des ressources, offre de la flexibilité dans leur mutualisation, ce qui permet de réaliser des économies parfois à grande échelle. Le SDN crée des abstractions de réseau pour permettre une innovation plus rapide, une grande flexibilité et un management centralisé. Le NFV sépare les fonctions du matériel pour réduire les CAPEX et les OPEX des opérateurs réseaux et accroître l'agilité du service.

En résumé, tous ensembles, le Cloud, le SDN et le NFV offrent notamment les fonctions suivantes :
— Automatisation du déploiement et de l'orchestration des services
— Virtualisation et mutualisation des ressources et des fonctions avec une garantie d'isolation et une plus grande agilité.

2.8 Organismes de standardisation

Plusieurs organisations internationales participent à l'effort de standardisation, de normalisation du SDN. L'ONF qui est le principal organisme qui assure la définition de son architecture SDN[56], la définition des différents composants [57, 58, 59] ainsi que les spécifications techniques du protocole OpenFlow le principal de l'interface sud.

Les tableaux 2.3, 2.4, 2.5, 2.6 présentent un sommaire des activités des principaux organismes qui contribuent à l'effort de normalisation des différentes technologies introduites dans ce chapitre.

2.9 Conclusions

Dans ce chapitre nous avons introduit les différents concepts et technologies qui serviront de *background* dans le cadre de cette thèse. Nous avons présenté le SDN et ses principaux challenges actuels sur ses interfaces sud, est/ouest, l'interopérabilité entre contrôleurs SDN hétérogènes, la notion de multi-contrôleurs (*clustering* et la problématique de la synchronisation des données entre instances. Nous avons ensuite introduit brièvement la notion de NFV, présenté son architecture et défini ses composants. Nous avons présenté les principaux projets open-source portant sur la mise en oeuvre du NFV

TABLE 2.3 – Quelques groupes de travail de l'ONF sur le SDN

Organisations	Working Group	Activités
ONF	Architecture & framework	Achitecture du SDN, définition des composants et des interfaces du SDN
	Interfaces Nord	Définition des interfaces Nord pour le SDN
	Configuration & Management	OAM (Operation, Administration et Management), foctionnalités du protocole OF, Spécifications du management de la configuration du prococole OF
	Testing and Interoperability	Spécification de la suite de tests de conformité SDN
	Wireless & Mobile	Spécification des fonctionnalités du SDN pour le mobile et le WI-FI
	Optical Transport	Spécification des fonctionnalités SDN et de contrôle pour les réseaux de transport optiques
	Migration	Définir les méthodes de migration des réseaux conventionnels vers les réseaux basés sur SDN (Openflow)

tels que et ONAP. Enfin, nous avons présenté les principaux organismes qui concourent à l'effort de standardisation et de normalisation de ces différentes technologies.

Dans le chapitre 4 nous introduirons la notion des réseaux hybrides SDN. Nous allons également présenter notre approche, proposée dans le cadre du projet CARP pour adresser cette problématique.

TABLE 2.4 – Quelques groupes de travail de l'IETF/IRTF et de l'ETSI sur le SDN

Organisations	Working Group	Activités
IETF	Interface to Routing System (I2RS)	Spécification d'une interface de communication de l'interface sud compatible avec système de routage « legacy » Définition d'un modèles de données standardisés pour la communication entre les éléments du réseau
	Network Configuration (NETCONF) [60]	Spécification d'un protocole de management des éléments réseaux
	Forwarding Configuration (ForCEC)[61]	Spécification d'un protocole de communication le plan de contrôle et le plan de données
IRTF	SDN Research Group	Prospection du SDN pour l'évolution d'Internet
ETSI	ETSI ISG	Orchestration de la NF [62] y compris le contrôle combiné des ressources de calcul, de stockage et de réseau

TABLE 2.5 – Quelques groupes de travail de l'IEEE sur le SDN

Organisations	Working Group	Activités
IEEE	802	Applicabilité du SDN à l'infrastructure IEEE 802

TABLE 2.6 – Quelques groupes de travail de l'UIT sur le SDN

	Working Group	Activités
IUT-T	SG 11	Spécifications de signalisation utilisant les technologies SDN dans les réseaux d'accès à large bande
	SG 13	Spécifications fonctionnelles et architecture pour le SDN et les réseaux du futur
	SG 15	Spécification d'une architecture de plan de contrôle de réseau de transport pour supporter le contrôle SDN des réseaux
	SG 17	Spécifications des aspects de l'architecture pour la sécurité du SDN et des services en utilisant le SDN

Chapitre 3

Le paradigme des réseaux SDN hybrides

Sommaire

3.1 Introduction 56
3.2 Réseaux hybrides du SDN 56
3.3 Modèle de réseaux hybrides SDN 57
3.3.1 Modèle hybride basé sur la topologie 57
3.3.2 Sur des noeuds de bordure 58
3.3.3 Modèle hybride basé sur le service 58
3.3.4 Modèle hybride basé sur la classe 59
3.3.5 Modèle hybride basé sur l'intégration 60
3.4 Prototypes et implémentations de solutions hybrides 60
3.4.1 Routing Control Platform (RCP) 60
3.4.2 Panopticon 61
3.4.3 HybNET 61
3.4.4 HybridFlow 61
3.4.5 OpenRouteFlow et ClosedFlow 62
3.4.6 B4 62
3.4.7 Fibbing 62
3.4.8 SDN-IP 62
3.5 Architecture CARP pour les réseaux hybrides 63
3.5.1 Architecture 63
3.5.2 Fonctionnement 64
3.5.3 Positionnement de notre approche par rapport la littérature 65
3.6 Conclusions 65

3.1 Introduction

Ce chapitre introduit le concept de réseaux hybrides SDN. Il est organisé comme suit : dans la section 3.2 nous introduirons le concept de réseaux hybrides, dans la section 3.3 nous présenterons les différents modèles identifiés dans la littérature, tandis que dans la section 3.4 nous présenterons quelques prototypes de réseaux hybrides parmi les plus importants présentés dans l'état de l'art. Dans la section 3.5 nous présenterons l'architecture de CARP pour les réseaux hybrides et enfin une conclusion dans la section 3.6.

3.2 Réseaux hybrides du SDN

Un réseau hybride est caractérisé par la coexistence [63] des équipements [1] (routeurs, commutateurs) utilisés dans les réseaux traditionnels et les nouveaux équipements programmables introduits par le SDN.

Un réseau hybride consiste à remplacer une partie des équipements traditionnels par de nouveaux équipements compatibles SDN pour profiter des avantages offerts par cette technologie.

Le paradigme SDN Hybride fait référence à une architecture réseau dans laquelle à la fois les réseaux traditionnels (distribués) et le SDN (centralisé) coexistent et communiquent ensemble à différents degrés pour configurer, contrôler, changer, et gérer le comportement du réseau.

Le SDN a le potentiel pour surmonter les inconvénients des réseaux actuels, il promet également de faciliter la conception, l'exploitation, la gestion et la communication à l'intérieur des réseaux. La combinaison du SDN et du *legacy* imposent l'acheminement d'une partie du trafic en utilisant un contrôle traditionnel tandis que l'autre partie du trafic est gérée par le SDN. Cela implique donc la définition de mécanismes pour la communication entre ces deux technologies.

La mise en place des réseaux hybrides peut se faire à trois niveaux :

— Au niveau des équipements traditionnels : Peut être réalisée par l'ajout d'un « firmware » (logiciel) sur des équipements traditionnels (« Legacy ») pour supporter le SDN. Même si cette cette approche est techniquement facile, sa mise en oeuvre dépend totalement de la volonté des fabricants d'équipements réseaux. Un cycle d'innovation assez long, une grande diversité des fabricants, l'impossibilité pour les utilisateurs d'innover sur les équipements propriétaires rendent cette solution difficile à mettre en oeuvre.
— Au niveau des contrôleurs SDN : L'autre approche consiste à développer au niveau des contrôleurs SDN des mécanismes permettant de garantir une communication transparente et fiable entre des équipements complètement hétérogènes en termes de technologies. Le contrôleur doit donc assurer une traduction des règles OpenFlow pour par exemple des routeurs/commutateurs traditionnels et vice-versa.
— Combiner les deux approches : Cette approche consiste à développer une partie du mécanisme aux niveaux des contrôleurs SDN et l'autre au niveau des commutateurs. Cela peut être maté-

1. Nous utilisons indifféremment routeur et commutateur pour désigner les équipements traditionnels exécutants des protocoles distribués

rialisé par la présence de ports spécialisés au niveau des commutateurs/routeurs.

L'introduction du SDN dans un réseau existant (traditionnel) entraîne des coûts en investissement CAPEX et OPEX et des problèmes de déploiement et d'exploitation. Ainsi, il est nécessaire de trouver une méthodologie pour son introduction progressive et une approche adaptée pour son exploitation.

Plusieurs approches sont envisageables pour l'introduction progressive du SDN. Nous présenterons plus loin dans ce chapitre, une revue des différentes solutions introduites dans la littérature ainsi que notre contribution pour adresser ce problème.

Dans les sections suivantes nous présentons les différentes approches et solutions introduites dans la littérature au moment de la rédaction de ce document pour adresser cette problématique. Nous allons également présenter notre approche, proposée dans le cadre CARP pour répondre à cette problématique. Enfin, nous présenterons une description détaillée de cette architecture dans la section suivante.

Nous présentons dans cette partie les différentes solutions proposées pour la mise en œuvre des réseaux hybrides.

3.3 Modèle de réseaux hybrides SDN

Les réseaux hybrides SDN peuvent être classifiés en deux types [63] selon la couche qui servira à son introduction :

1. Coexistence seulement au niveau du plan de contrôle
2. Coexistence sur le plan de contrôle et le plan de données

Selon [64], il existe quatre modèles pour les architectures hybrides. Ces différents modèles sont présentés dans la section 3.3. Ces modèles ne sont que des descriptions de différentes approches envisageables pour la mise en œuvre des réseaux hybrides. Certains sont trop théoriques et trop complexes pour être déployés dans un environnement de production en entreprise.

3.3.1 Modèle hybride basé sur la topologie

Ce modèle est basé sur une approche où les deux technologies coexistent au niveau du plan de données et du plan de contrôle. Il consiste à diviser le réseau en deux zone distincts. Une zone pour le réseau traditionnel et une autre zone pour le réseau SDN. Les nœuds sont reparties dans le réseau selon la technologie supportée. Un nœud ne peut appartenir qu'à une seule zone. Une zone est un ensemble de nœuds interconnectés, contrôlés par le même paradigme. Pour assurer les services de bout en bout entre deux nœuds par exemple appartenant respectivement à une zone SDN et une autre de réseau traditionnel, il est nécessaire de mettre en place un service inter-zone.

Cette approche offre une grande facilité dans sa mise en œuvre. Elle est particulièrement adaptée pour un opérateur dans le cadre d'une extension (dans une nouvelle région par exemple) ou, suite à une fusion ou acquisition, d'un opérateur disposant d'une portion de son réseau en SDN. Cette approche permet en autre aux opérateurs d'acquérir une expertise dans le déploiement, la gestion et le management des réseaux SDN ou encore de bien se positionner par rapport à la maturité de cette technologie. Pour la transition des réseaux traditionnels vers le SDN, cette approche est particulièrement adaptée

pendant la phase de transition pour les opérateurs souhaitant adopter le SDN. En outre, une défaillance du contrôleur SDN n'impacte pas le bon fonction du réseau.

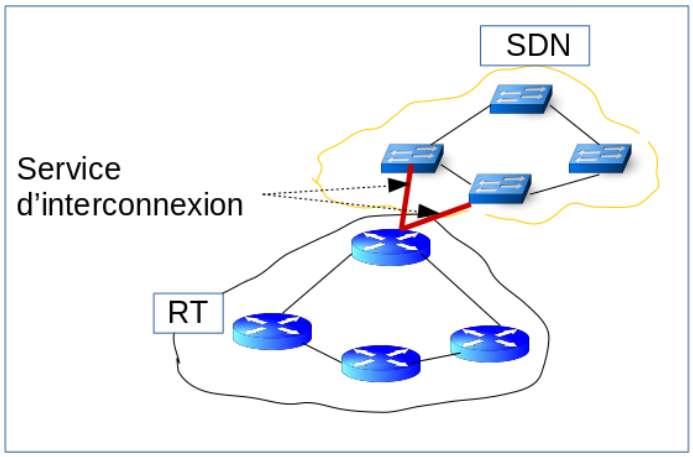


FIGURE 3.1 – Modèle hybride basé sur la topologie

La figure 3.1 représente un réseau divisé eu deux zones, une zone pour le réseau SDN et une autre zone pour le réseau traditionnel. Cette approche ne nécessite pas la modification des équipements existants mais demande par ailleurs la mise en place de quelques mécanismes spéciaux pour permettre l'échange d'informations entre les zones.

3.3.2 Sur des noeuds de bordure

Cette approche consiste à placer les noeuds SDN sur la bordure du réseau à l'image d'un routeur BGP. Les noeuds de bordure (« edge ») sont placés sous le contrôle du SDN. Le trafic dans le coeur du réseau reste assurer par le biais des protocoles distribués habituels.

Cette approche présente des avantages en terme de coût de mise en oeuvre. Elle permet de bénéficier de nombreuses fonctionnalités offertes par le SDN. En outre, le nombre de noeuds gérés par le contrôleur n'est pas très important et correspond bien au but du SDN.

3.3.3 Modèle hybride basé sur le service

C'est une approche dont le principe repose sur une segmentation du réseau basée sur les services. Chaque paradigme (SDN ou réseaux traditionnels) offre des services différents. Au contraire de l'approche basée sur la topologie, un nœud peut être contrôlé par les deux paradigmes. Ainsi, sur un commutateur, on peut avoir un ou plusieurs ports dédiés au SDN et les autres aux réseaux traditionnels. Pour la mise en œuvre de cette approche, on peut introduire progressivement des nœuds SDN auxquels on assigne un sous ensemble de services spécifiques, un « load-balanceur » par exemple. Un autre avantage de ce modèle repose sur le fait qu'il ne nécessite pas la modification des routeurs dits « legacy » lors de la phase de transition vers le SDN.

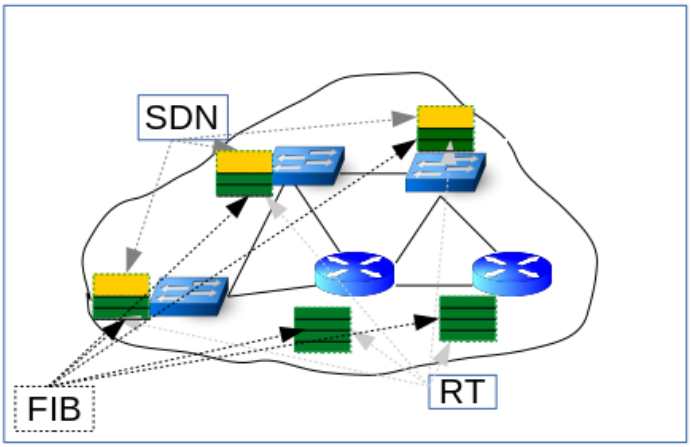


FIGURE 3.2 – Modèle hybride basé sur le service

La figure 3.2 présente un réseau hybride avec une partie des équipements compatibles SDN. Les équipements sont de deux types : les équipements purement traditionnels sans ports dédiés au SDN et les commutateurs avec des ports dédiés au SDN ainsi que des ports dédiés aux réseaux traditionnels. Cette architecture offre plus de visibilité, plus de contrôle sur une partie du réseau.

3.3.4 Modèle hybride basé sur la classe

Ce modèle consiste à partitionner le trafic en un ensemble de classes. Puis, on divise les classes en deux sous-ensembles. Le contrôle de chaque sous-ensemble est affecté respectivement au SDN et aux réseaux traditionnels. Contrairement au modèle hybride basé sur le service, ici chaque paradigme est responsable de tous les services réseaux dans le sous ensemble de classe de trafic qui lui a été assigné. La figure 3.3 est une illustration de ce modèle, une partie du Forwarding Information Base (FIB) des

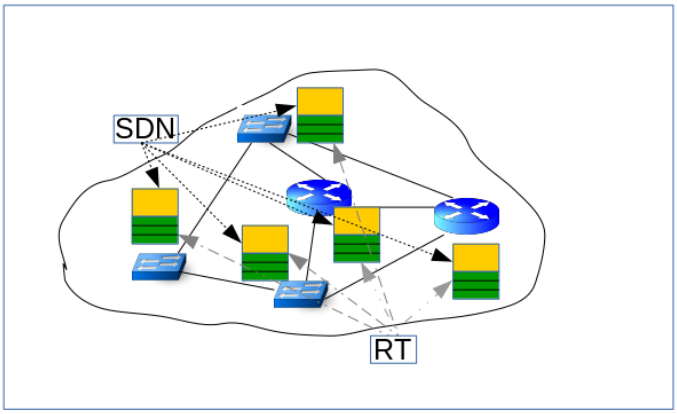


FIGURE 3.3 – Modèle hybride basé sur la classe

nœuds étant dédiés au SDN et l'autre partie au réseau traditionnel. Cependant, ce modèle nécessite une modification matérielle et logicielle des équipements traditionnels pour être mis en œuvre. L'inconvénient de ce modèle repose sur le fait qu'il peut induire des coûts importants pour être mis en œuvre.

3.3.5 Modèle hybride basé sur l'intégration

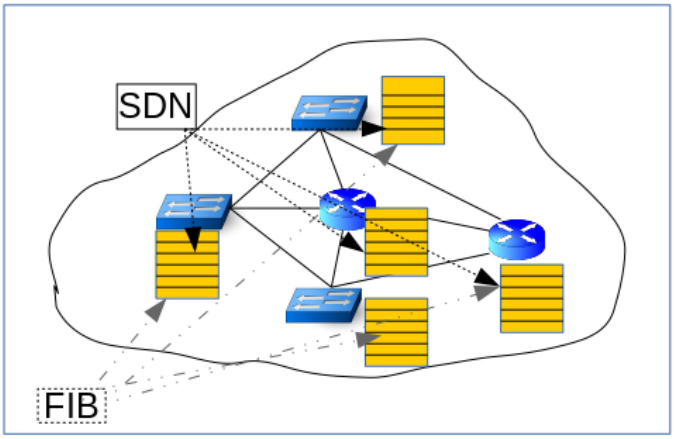


FIGURE 3.4 – Modèle hybride basé sur l'intégration

Dans ce modèle, tous les équipements sont gérés par le SDN (contrôleur centralisé). Ils exécutent tous des protocoles traditionnels. Cependant, ce modèle nécessite de déplacer le plan de contrôle au niveau du contrôleur SDN. Le contrôleur calcule le plan de contrôle de façon centralisée et les commutateurs assurent la transmission des données. Il peut par exemple injecter des routes soigneusement sélectionnées dans le système de routage.

3.4 Prototypes et implémentations de solutions hybrides

Dans cette section nous décrivons les principales propositions trouvées dans la littérature au moment où nous rédigeons ce document. Pour chaque solution, nous ferons une brève description, présenterons les avantages ainsi que quelques inconvénients. Certains propositions ne sont que des prototypes et d'autres sont dans un état plus avancé.

3.4.1 Routing Control Platform (RCP)

RCP [65] décrit une approche centralisée pour l'agrégation et l'externalisation du plan de contrôle des routeurs BGP d'un Autonomous System (AS) logiquement au sein d'un contrôleur. Seul le plan de contrôle des routeurs BGP sera centralisé, celui des routeurs Interior Gateway Protocol (IGP) reste distribué.

En centralisant une partie du plan de contrôle (BGP), on dispose ainsi d'une vue quasi globale du

réseau et en même temps on acquière plus de flexibilité et de réactivité par rapport à des évènements tels que la défaillance d'un routeur de bordure. Cependant, cette solution nécessite néanmoins une modification des équipements existants, ce qui induit un investissement pour l'acquisition de nouveaux équipements.

3.4.2 Panopticon

Les auteurs de cette solution [66] proposent un cadre d'optimisation pour déterminer l'emplacement pour un déploiement partiel du SDN de telle sorte à imposer que chaque flot du réseau doit traverser au moins un commutateur compatible SDN.

Panopticon nécessite également d'investir sur des nouveaux équipements compatibles SDN. Cette solution offre, cependant, une approche intelligente pour des entreprises souhaitant introduire progressivement le SDN mais tout en conservant des équipements traditionnels. Elle permet ainsi de tirer profit des avantages offerts par le SDN sans modifier totalement l'infrastructure du réseau.

Le fait d'imposer que chaque flot doit traverser au moins un commutateur compatible SDN limite à notre avis le champ d'application de cette solution à des réseaux de taille moyenne.

3.4.3 HybNET

HybNET [67] offre un mécanisme de configuration commun pour réseau hybride (SDN, équipements traditionnels) avec un dispositif de traduction automatique de la configuration des équipements traditionnels vers le SDN et vice versa.

Son principal avantage réside dans le fait qu'il ne nécessite pas une modification importante des équipements existants en termes de protocoles supportés ; il faut juste ajouter le module de traduction des règles des API de l'interface sud du SDN pour les équipements « legacy » et vice-versa.

Le dispositif de traduction (prxoy) des règles doit être présent sur tous les commutateurs/routeurs du réseau, cette tâche peut s'avérer compliquée pour les équipements propriétaires dont seul leur fabricant peut activer une nouvelle fonctionnalité.

3.4.4 HybridFlow

HybridFlow [68] est une solution basée sur un plan de contrôle SDN léger. Le principe consiste à remplacer partiellement une partie des commutateurs traditionnels par des commutateurs OpenFlow et d'obliger tous les flots à passer par au moins un commutateur SDN.

HybridFlow minimise le coût du passage au SDN en minimisant le nombre des équipements à remplacer, il ne nécessite aucune modification au niveau des équipements traditionnels. La traduction des règles OpenFlow est assurée par un module de *Traduction de messages openflow*.

Comme *HybNET*, cette solution est applicable sur une infrastructure de petite taille avec une forte dépendance au protocole OpenFlow.

3.4.5 OpenRouteFlow et ClosedFlow

OpenRouteFlow [69] et ClosedFlow [70] sont deux autres solutions introduites dans la littérature qui offrent respectivement un mécanisme permettant de disposer d'une vue globale sur les réseaux distribués sans déplacer le plan de contrôle et un mécanisme pour supporter le protocole OpenFlow sur des équipements (commutateurs ou routeurs) « legacy ».

3.4.6 B4

C'est la solution proposée et utilisée par *Google* pour interconnecter ses data-centers [10] par le SDN. *Google* dispose de plusieurs data-centers éparpillés un peu partout dans le monde. L'interconnexion intra-site reste assuré par les protocoles traditionnels tandis que l'interconnexion entre les différents sites est gérée en SDN. Chaque site est sous la responsabilité d'un contrôleur. Un contrôleur global permet de disposer d'une vue globale de l'ensemble de l'infrastructure. La communication entre les équipements SDN et traditionnels est assurée par une application dédiée (Routing Application Proxy).

Cette solution est particulièrement adaptée aux très grands réseaux (WAN), c'est l'un des tout premiers déploiements du SDN dans un environnement de production de l'histoire. *B4* offre l'une des meilleures pistes aux très grandes entreprises pour l'intégration du SDN dans leur infrastructure réseau. *B4* permet ainsi de déployer le SDN pour des services spécifiques sans perturber le fonctionnement normal de l'infrastructure existante. Cette solution nécessite néanmoins la modification des équipements de bordures existants au sein de différents sites.

3.4.7 Fibbing

Le principe de cette proposition repose sur l'ajout d'un nœud fictif(« fake node ») dans la topologie du réseau de manière à altérer la vision de certains nœuds sur l'état du réseau [71]. Cette solution est différente de toutes celles présentées jusqu'à présent. Elle est hybride dans la mesure où elle propose l'introduction d'un contrôleur centralisé dans un réseau composé exclusivement d'équipements traditionnels exécutant des protocoles de routage distribués à état de lien tels que OSPF.

Cette solution est adaptée à la couche L3, cette solution ne nécessite aucune modification de niveau matériel. Cependant, elle ne peut pas être déployée avec des technologies fonctionnant au niveau de la couche L2 tel que le SDN qui utilise cette couche pour maintenir la topologie du réseau par exemple.

3.4.8 SDN-IP

SDN-IP[11] est une application ONOS qui permet à un réseau SDN de se connecter à des réseaux externes sur Internet en utilisant le protocole standard BGP. D'un point de vue externe, du point de vue BGP, le réseau SDN apparaît comme un seul système autonome (SA) qui se comporte comme tout SA traditionnel. Au sein de l'AS, l'application *SDN-IP* fournit le mécanisme d'intégration entre BGP et ONOS. Au niveau du protocole, *SDN-IP* se comporte comme un *speaker* BGP ordinaire. Du point de vue du contrôleur ONOS, il s'agit simplement d'une application qui utilise ses services pour installer et mettre à jour l'état de transfert approprié dans le plan de données SDN.

Notre solution, propose une approche sans modification matérielle, ni logicielle des équipements actuels. Elle permet de disposer d'une vue globale et d'un contrôle partiellement centralisé du réseau. Le contrôleur entre en action de façon périodique, suite à des événements préalablement définis. L'intervention du contrôleur est déclenchée par des événements préalablement définis, la défaillance d'un lien par exemple.

3.5 Architecture CARP pour les réseaux hybrides

Nous avons présenté dans ce document les avantages des dernières générations des architectures réseaux. Nous avons également présenter les difficultés pour l'intégration et le déploiement du SDN qui sont dues notamment à son incompatibilité aux réseaux traditionnels. Pour répondre à ces difficultés, nous avons pensé à concevoir une nouvelle architecture permettant aux entreprises de profiter du SDN pendant cette phase de transition.

Dans cette section, nous allons présenter une description de l'architecture proposée. Nous allons également présenter une description de son fonctionnement.

3.5.1 Architecture

La figure 3.5 présente l'architecture proposée. Elle est composée d'un contrôleur centralisé et des agents chargés de collecter des informations sur les dispositifs de routage. Contrairement au SDN classique, le plan de contrôle reste localisé au niveau des routeurs ou commutateurs. Nous allons présenter le rôle de chaque élément de l'architecture ainsi que son fonctionnement.

3.5.1.1 Plan de contrôle

Le plan de contrôle est responsable de l'élaboration des tables de routage. Il utilise des protocoles distribués tels que IS-IS, OSPF pour élaborer ses décisions de routage. Ces décisions peuplent ensuite la RIB (Routing Information Base).

3.5.1.2 Plan de données

Le plan de données quant à lui est responsable de l'acheminement des données. Il se sert pour cela du calcul effectué par le plan contrôle. Il utilise une table appelée FIB.

3.5.1.3 Agent I2RS

L' Agent est responsable de la collecte des informations sur le dispositif de routage auquel il est associé. Il transmet les informations au contrôleur par le biais d'un canal sécurisé. C'est un élément clé de l'architecture.

3.5.1.4 Contrôleur

Le Client I2RS ou le contrôleur a pour rôle de centraliser toutes les informations sur l'état du réseau. Il utilise les informations remontées par les Agents pour envoyer des instructions aux routeurs.

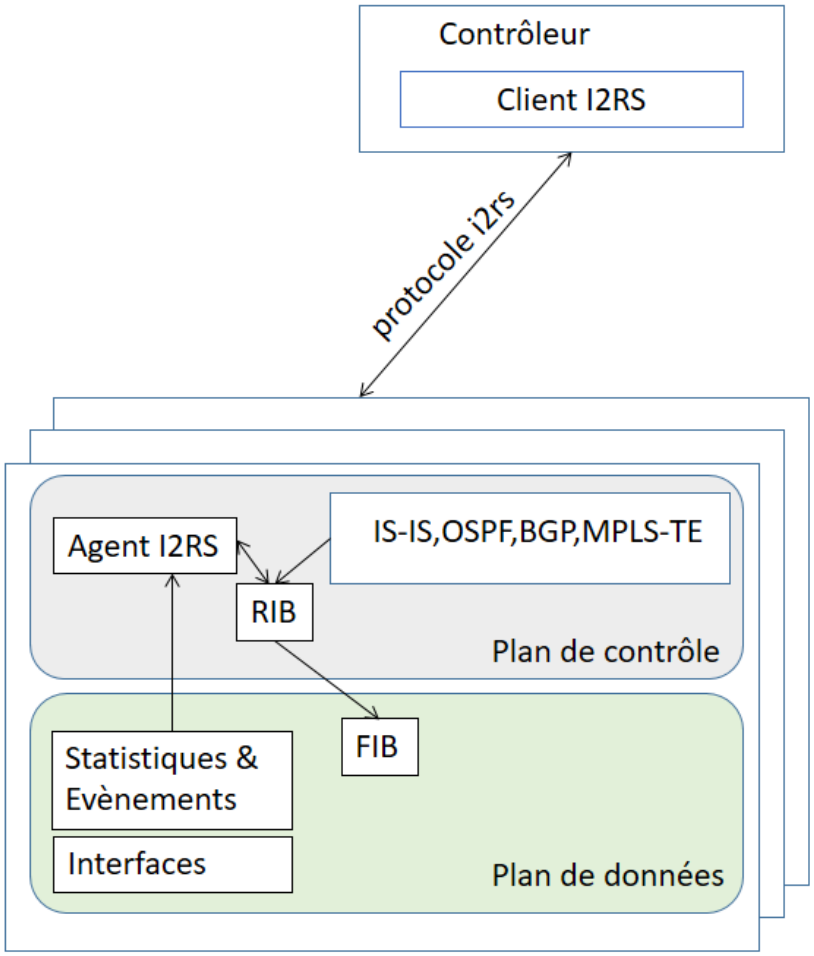


FIGURE 3.5 – Nouvelle architecture centralisée pour les réseaux distribués

Il communique d'une façon sécurisée avec le système de routage à travers les Agents I2RS.

3.5.2 Fonctionnement

Avec le SDN classique, le contrôle du réseau est assuré intégralement par le contrôleur, l'architecture que nous proposons maintient le plan de contrôle au niveau des routeurs. Toutefois, le contrôleur peut reprendre la main de façon périodique et injecter des décisions au niveau des routeurs. Il effectue donc un contrôle partiel sur les équipements. Le contrôleur dispose d'une vue sur l'état du réseau mais n'intervient que seulement sur des cas précis. Pour assurer les échanges entre le contrôleur et les routeurs de façon sécurisée, on utilise le couple Netconf [72], YANG [26].

3.5.3 Positionnement de notre approche par rapport la littérature

Dans cette section, nous décrivons le positionnement de notre approche vis à vis des principales solutions introduites dans la littérature. Nous avons décrit plus haut les principaux obstacles au déploiement du SDN. Nous avons également présenté dans les sections 3.3 et 3.4, l'état de l'art sur les réseaux hybrides. Les principales solutions introduites dans la littérature imposent pour la plupart d'entre elles une mise à niveau des équipements traditionnels ou l'acquisition des nouveaux équipements pour être mises en œuvre. C'est le cas pour les solutions **CloseFlow**, **HybNET, HybridFlow**, **Panopticon**, **B4**, **Telekinesis**. Nous rappelons que l'un des objectifs des approches hybrides vise à minimiser le coût d'intégration du SDN en minimisant notamment le nombre d'équipements à remplacer. Même si **Fibbing** peut être mise en oeuvre sans mise à niveau des équipements actuels, sans la nécessité d'en acquérir de nouveaux, cette solution n'est pas adaptée aux équipements totalement SDN qui utilisent la couche L2 pour le routage.

3.6 Conclusions

Dans ce chapitre, nous avons introduit la notion de réseaux hybrides SDN. Nous avons présenté les différentes approches introduites dans la littérature pour ce type de réseau. Nous avons également présenté l'architecture hybride que nous avons proposée qui a fait l'objet d'une publication dans une conférence internationale. Nous présenterons une évaluation des performances de l'architecture proposée sur la tolérance aux pannes dans le chapitre 5. Dans le chapitre suivant (chapitre 4) nous présenterons une analyse des performances des contrôleurs SDN basée sur la découverte de la topologie.

Chapitre 4

Évaluation des performances : découverte de la topologie

Sommaire

4.1 Introduction 68
4.2 Techniques d'évaluation des performances du SDN 68
4.2.1 Modèle analytique 68
4.2.2 Simulation 69
4.2.3 Approche expérimentale 70
4.3 État de l'art sur l'analyse des performances du SDN 70
4.4 Choix des contrôleurs ONOS et ODL 71
4.5 Gestion de la topologie dans le SDN 72
4.5.1 Découverte des hôtes 72
4.5.2 Découverte des commutateurs 73
4.5.3 Découverte des liens entre commutateurs 73
4.6 Plate-formes expérimentales 76
4.6.1 Architecture de la plate-forme expérimentale 76
4.6.2 Méthodologie pour mesurer le temps de découverte de la topologie 78
4.7 Analyse des résultats 79
4.7.1 Découverte de la topologie : Cas d'un contrôleur unique 80
4.7.2 Découverte de la topologie : Cas de plusieurs contrôleurs (*cluster*) 81
4.8 Conclusions 81

4.1 Introduction

Le SDN est le nouveau paradigme réseau avec une grande potentialité permettant d'augmenter l'efficacité de la gestion du réseau. Le SDN offre une grande flexibilité et facilite l'innovation dans les réseaux grâce à la programmabilité. Le concept clé du SDN est la séparation du plan de données et du plan de contrôle. Contrairement aux réseaux IP traditionnels, où les routeurs réalisent à la fois le transfert de paquets et le cacul du routage, la fonction de routage est déplacée des routeurs vers le contrôleur SDN. Dans l'architecture SDN, les commutateurs réalisent seulement la fonction de transfert des données sur la base des règles établies et installées par un ou plusieurs contrôleurs SDN .

Ce chapitre est organisé comme suit. Dans la section 4.2 nous présentons les différentes techniques utilisées pour l'évaluation des performances des contrôleurs. Dans la section 4.3 un état de l'art sur les études comparatives des performances des contrôleurs SDN est fourni. Dans la section 4.4 nous présentons les raisons pour lesquelles nous avons concentré nos travaux sur les contrôleurs ONOS et ODL, dans la section 4.5 nous présentons en détails le mécanisme de gestion de topologie dans le SDN. Dans la section 4.6, nous décrivons la plate-forme expérimentale utilisée dans le cadre de nos travaux pour l'évaluation des performances des contrôleurs sur la découverte de topologie. Dans la section 4.7 nous analysons des résultats obtenus. Enfin, la section 4.8 termine le chapitre par une conclusion.

4.2 Techniques d'évaluation des performances du SDN

Généralement, trois techniques sont utilisées pour évaluer les performances des réseaux. Ce sont la méthode analytique, la simulation et la méthode expérimentale. Pour l'analyse des performances du SDN, la méthode expérimentale est plus utilisée même s'il existe quelques études basées sur les modèles analytiques. Dans cette section, nous présentons brièvement ces trois techniques, nous présenterons ensuite un état de l'art des principales études basées sur chacune des techniques mentionnées précédemment.

4.2.1 Modèle analytique

Depuis le début nos travaux dans le cadre de cette thèse, nous n'avons pas rencontré beaucoup d'études sur les performances du SDN basées sur l'approche analytique. Au moment où nous rédigeons ces lignes, il n'existait que très peu d'études. Nous présentons les principales d'entre elles.

Dans [73, 74], les auteurs utilisent la théorie des files d'attente pour analyser les performances des commutateurs OpenFlow et du contrôleur SDN en terme de temps de service. Ils ont démontré la corrélation entre le temps de service et l'augmentation du nombre de commutateurs dans le réseau d'une part et le taux d'arrivée des messages *Packet-IN* d'autre part.

Dans [75], les auteurs proposent une métrique sur l'évolutivité pour les plans de contrôle SDN. Ils ont étudié trois structures de plan de contrôle SDN - les architectures centralisées, distribuées et hiérarchiques. Les chercheurs ont proposé enfin des modèles mathématiques pour l'évaluation des

performances, en particulier le temps de réponse, sur la base desquels ils ont analysé l'évolutivité de ces trois structures.

4.2.2 Simulation

La simulation est une technique utilisée par les chercheurs pour étudier des systèmes sur des entités simulées sans réaliser des expériences sur des entités réelles.

Plusieurs outils [76] de simulation ont été développés dans le cadre des réseaux informatiques en général et particulièrement pour simuler ou émuler le SDN. Nous utiliserons indifféremment les termes émulation ou simulation dans la suite de ce document. Dans cette section, nous présentons les principaux outils utilisés pour la simulation du SDN. Nous présenterons également les principaux travaux basés sur la simulation avec l'outil *OMNeT++* [77].

— *Mininet* [12] est un émulateur de réseau qui crée un réseau virtuel d'hôtes, de commutateurs, de contrôleurs et de liens. Les hôtes Mininet utilisent un logiciel réseau Linux standard et ses commutateurs prennent en charge OpenFlow pour un routage sur mesure très flexible et une mise en réseau définie par logiciel.
— *OMNeT++* est un simulateur très populaire dans le monde des réseaux. En *2013*, *Omnet++* propose une option supportant le protocole OpenFlow *v1.0*. En 2015, une extension qui est la version *1.3* du protocole OpenFlow a été proposée [78].
— *NS-3* [79] présente une option permettant la simulation de la version *0.8.9* d'OpenFlow.
— *Cbench* [80] (« controller benchmarker ») est un programme pour tester les contrôleurs OpenFlow en générant des événements *Packet-IN* pour de nouveaux flux. Cbench émule un tas de commutateurs qui se connectent à un contrôleur et envoient des messages. Cbench fonctionne essentiellement sur deux modes : « throughput mode » pour le débit et « latency mode » pour la latence. En mode débit (« throughput mode »), il envoie autant de paquets que possible pour calculer le nombre maximum de paquets traités par le contrôleur. En mode latence (« latency mode »), Cbench envoie un paquet et attend la réponse pour calculer le temps de traitement d'un seul paquet par le contrôleur.
— *Iperf* [81] est un outil de mesure active de la bande passante maximale possible sur les réseaux IP. Il prend en charge le réglage de divers paramètres liés à la synchronisation, aux tampons et aux protocoles (TCP, UDP, SCTP avec IPv4 et IPv6). Pour chaque test, il indique le throughput, le taux de perte et d'autres mesures.

Dans [78], les auteurs proposent une implémentation d'un modèle du système OpenFlow dans le framework *INET* [82] pour *OMNeT++*. Ils ont ensuite présenté les résultats de performance pour valider l'exactitude de leur modèle.

Dans [83], les auteurs proposent un nouveau module sur la base d'une extension de [78] développé pour la simulation de réseaux basés sur OpenFlow *1.3* avec *OMNeT++*, un « framework » de simulation modulaire bien connu et largement utilisé, qui offre un haut degré de support pour les expériences. Il se concentre principalement sur la modélisation du commutateur et du contrôleur OpenFlow avec une attention particulière sur les aspects liés à l'équilibrage de charge, la tolérance aux pannes et la qualité de service.

Dans [84], les chercheurs proposent un algorithme de routage économe en bande passante avec

SDN pour minimiser l'énergie globale pour le trafic des centres de données dans le temps. Avec les simulations mises en œuvre dans *OMNeT++*, nous montrons que notre algorithme peut réduire l'énergie globale par rapport au volume de trafic et réduire le temps de réalisation des flux en moyenne. Dans [85], les auteurs proposent un cadre de simulation basé sur *OMNeT++* pour évaluer la performance des architectures de contrôleurs SDN distribués.

La difficulté principale à l'utilisation des outils *OMNeT++* et *NS-3* dans le cadre des travaux de recherches sur le SDN est liée au décalage entre l'évolution de celui et la disponibilité des mises jour permettant la prise en charge de ses évolutions par ces outils. Ainsi *NS-3* et *OMNeT++* ne supporte respectivement que seulement la version *0.8.9* d'OpenFlow et que *1.3* au moment de la rédaction de ce manuscrit.

4.2.3 Approche expérimentale

La méthode expérimentale consiste à effectuer des mesures sur une infrastructure réelle (sur des équipement physique ou virtualisés). C'est cette approche que nous avons utilisée dans le cadre de cette thèse.

4.3 État de l'art sur l'analyse des performances du SDN

L'architecture du SDN date des années 2000, POX/NOX [86] et RYU [87] en furent les premières implémentations. Aujourd'hui, il existe plusieurs contrôleurs SDN classés principalement dans deux catégories les solutions SDN proposées par les équipementiers tels que CISCO ou encore JUNIPER ainsi que des projets non commerciaux dits open-sources tels que ONOS [4], ODL [5] ou encore floodlight [88].

SDX [89] propose une liste des contrôleurs commerciaux et open-sources . Cependant deux projets open-source se sont imposés ces dernières années

La couche contrôle est un élément critique de l'architecture SDN. De nombreuses études sur l'évaluation des performances et l'analyse comparative des différentes contrôleurs SDN ont été proposés dans la littérature. La latence et le débit sont les principales métriques utilisées le plus souvent pour l'évaluation des performances des contrôleurs SDN.

Dans cette section, nous présentons les principales études sur l'analyse des performances des contrôleurs SDN introduites dans la littérature.

Dans [90], les auteurs proposent une étude comparative des performances des contrôleurs POX/NOX, Beacon [91] et Maestro [92]. Ils concluent que le contrôleur POX/NOX offre les meilleures performances et qu'il est capable de répondre jusqu'à 1,6 million de requêtes par seconde avec un temps de réponse moyen de 2 ms.

Dans [93], les auteurs proposent une étude comparative des contrôleurs SDN basée une approche qualitative des fonctions offertes par les différentes solutions SDN. Ils proposent de prendre en compte dans la comparaison des contrôleurs SDN des caractéristiques telles que le support du protocole TLS,

la virtualisation, open-source ou commercial, les interfaces nord et sud, la qualité de l'interface graphique proposée, les API RESTful, la documentation, la productivité, la modularité, la compatibilité avec les principales plate-formes (Linux, Windows, Mac, ...), la compatibilité avec les interfaces Neutron d'Openstack, l'âge du contrôleur, la compatibilité avec le protocole OpenFlow. Cette étude, réalisée par une méthode Multi-Criteria Decision Making (MCDM) (une méthode de prise de décision multicritères) porté sur cinq contrôleurs POX, Ryu, Trema, Floodlight et Opendaylight, conclut que Ryu offre les meilleurs résultats.

Dans [94], les auteurs utilisent leur propre outil appelé HCProbe pour analyser les contrôleurs NOX, POX, Beacon, Floodlight, MuL, et Ryu. Cette étude a mis en évidence plusieurs vulnérabilités sur les contrôleurs analysés. En termes de performances les auteurs concluent que Beacon offre les meilleurs résultats. Dans [95], les auteurs analysent l'impact du choix de langage de programmation sur les performances et la probabilité des contrôleurs sur divers plate-formes. Ils concluent que le langage de programmation *Java* constitue le meilleur choix en raison de ses qualités sur le *multi-threading*. Les langages *C/C++* souffrent de limites sur la gestion de la mémoire. Le langage *.NET* est particulièrement dépendante de la plate-forme Windows et reste incompatible à *Linux* à ce jour. Les auteurs concluent finalement que le contrôleur Beacon offre les meilleurs résultats. Dans [96], les auteurs ont étudié les contrôleurs SDN les plus récents. Les résultats de leur étude démontrent que les contrôleurs développés avec le langage de programmation *C* tel que *MUL* [97] offrent les meilleures performances sur le débit (nombre de requêtes traitées par seconde). Les contrôleurs développés avec le langage *Java* tels que Beacon, IRIS [98] et Maestro offrent les meilleurs résultats après ceux développés en *C*. Ils démontrent également que les contrôleurs développés en *C* et *Java* offrent les meilleurs performances lorsque le nombre de *threads* augmente au contraire de ceux développés en *Python*. Ils affirment dans leurs conclusions que même si le choix d'un contrôleur est fortement dépendant de son domaine d'application, ODL constitue le meilleur candidat grâce notamment à la diversité des fonctions offertes par ce contrôleur. Dans [99] les auteurs utilisent l'outil *Cbench* pour analyser différents contrôleurs en terme de latence et de débit. Ils démontrent que les performances des contrôleurs sont fortement dépendantes du nombre de commutateurs gérés par le contrôleur. Ils concluent enfin que le contrôleur ONOS offre globalement les meilleures résultats sur le passage à l'échelle. Cette étude démontre notamment la montée en charge du nombre de commutateurs ou encore du nombre de machines hôtes n'impacte que faiblement les performances de ce contrôleur. Les auteurs concluent que cette performance s'explique par une gestion optimisée des adresses MAC (Media Access Control) par une collection de *hashes* réduisant ainsi le temps de consultation des règles de transfert de données par les commutateurs. Toutes ces études se focalisent sur le débit et la latence pour analyser les performances des différentes solutions SDN.

Le tableau 4.1 montre les principales critères qualitatives utilisées dans la plupart des études pour comparer les différentes solutions SDN. Notre contribution dans ce chapitre se focalise quant à elle un aspect qui n'était que très peu ou pas adressé par la littérature à savoir la découverte de topologie.

4.4 Choix des contrôleurs ONOS et ODL

Dans cette section nous présentons les raisons qui ont prévalue le choix de concentrer nos travaux sur les contrôleurs ONOS et ODL. Dans le tableau 4.1, nous avons présenté une liste des contrôleurs

SDN avec un ensemble de critères qualitatifs tels que le langage de programmation utilisé dans la majorité des études comparatives des performances des contrôleurs SDN. Une liste des contrôleurs open-sources et commerciaux est régulièrement mise à jour par la plate-forme *SDXCENTRAL* [89].

ONOS et ODL sont les contrôleurs les plus riches en fonctionnalités parmi les projets open-sources du moment. En s'appuyant sur *OSGi* (Open Services Gateway initiative (OSGi)) [100], ces contrôleurs sont très modulaires et ont d'excellents temps d'exécution pour le chargement des paquets. Ces deux projets ont l'avantage d'une interface graphique assez conviviale pour les développeurs d'applications avec une large communauté de contributeurs et une documentation abondante. En outre, ils proposent tous les deux une architecture distribuée idéale pour le déploiement du SDN dans un environnement réaliste. Ils proposent également une large palette des protocoles au niveau l'interface sud adaptés au réseau SDN hybride. Enfin, ils offrent une interface avec OpenStack via le module *Neutron* permettant ainsi l'orchestration des NFV et cloud qui reste un impératif majeur pour la gestion des ressources virtuelles.

4.5 Gestion de la topologie dans le SDN

La gestion de la topologie est une fonction critique de l'architecture du SDN. La découverte de la topologie implique l'ensemble des mécanismes permettant au contrôleur de découvrir les machines hôtes, les commutateurs et les liens inter-commutateurs. Les machines hôtes sont les noeuds physiques ou virtuels connectés aux commutateurs utilisées par les utilisateurs pour leurs services. Les commutateurs sont responsables de l'acheminement du trafic d'une source à une destination. Ils exécutent un protocole tel que OpenFlow pour communiquer avec le contrôleur. Les liens inter-commutateurs sont des liens physiques ou virtuels entre les commutateurs utilisés pour le transfert des paquets au sein d'un réseau. La découverte et la mise à jour de ces entités sont fondamentales au contrôleur pour disposer d'une vue globale et cohérente de la topologie du réseau.

Le tableau 4.2 présente les entités (commutateurs, liens et hôtes) ainsi que la procédure par laquelle elles sont découvertes par le contrôleur SDN. Dans cette section, nous présentons en détails ces différents mécanismes.

4.5.1 Découverte des hôtes

La découverte d'un hôte implique deux aspects. Le premier est le fait que le contrôleur doit connaître l'existence d'un noeud physique ou virtuel. Le deuxième aspect consiste, pour le contrôleur, du fait de savoir à tout moment quelle est la localisation exacte d'un hôte. Le contrôleur doit pouvoir localiser un hôte même suite à une migration d'un centre de données (« data-center ») à un autre quand il s'agit d'une machine virtuelle.

La découverte des machines hôtes permet donc au contrôleur de connaître la localisation d'un hôte dans le réseau. Ceci permet au contrôleur de mettre en oeuvre les politiques de sécurité, de surveillance et de qualité de service dans le réseau. En règle générale, tous les contrôleurs implémentent un module pour assurer la fonction de localisation des hôtes qui permet de déterminer pour chaque hôte le port et le commutateur par lequel il est attaché au réseau. Pour assurer un suivi efficace des hôtes, le contrôleur

maintient une table avec le profil de chacun des hôtes du réseau. Cette table permettra également au contrôleur de supprimer le profil du hôte quand celui-ci quitte le réseau.

Le contrôleur découvre la présence et la localisation exacte d'un hôte lors la réception du premier message *Packet-IN* envoyé par celui-ci. Le *Packet_IN* [3] est un message asynchrone envoyé d'un commutateur vers le contrôleur. Il intervient généralement lorsqu'un commutateur reçoit un paquet dont les informations de contrôle ne correspondent à aucune des règles déjà installées. C'est le cas notamment du premier paquet d'un nouveau flux. Il sollicite alors le contrôleur pour connaître le traitement approprié à ce flux.

Le contrôleur construit donc le profil de chaque hôte à la réception du message *Packet-IN* qui contient les informations telles que l'adresse Internet Protocol (IP), l'adresse Media Access Control et d'autres méta-données comme le numéro de port ou encore le Datapath ID (DPID). Pour le cas d'une machine virtuelle changeant de localisation à la suite d'une migration vers un autre centre de données, ces informations sont mises à jours.

4.5.2 Découverte des commutateurs

Connaître la présence et l'emplacement exacte des commutateurs OpenFlow est un élément essentiel pour le bon fonctionnement de l'architecture SDN. Le contrôleur et le commutateur échangent des messages **HELLO** lors de la phase de démarrage. Cette phase permet également aux commutateurs de découvrir l'adresse IP ainsi que le numéro port TCP du contrôleur (*6633 ou 6653*) avec lesquels ils établiront la connexion par la suite. Le contrôleur découvre l'emplacement des commutateurs OpenFlow lors du processus de « handshake » initial. A l'arrivée d'un nouveau commutateur au sein du réseau, le contrôleur découvre donc son existence et il enregistre par la suite les paramètres clés de celui-ci tels que l'adresse Media Access Control, le nombre de ports, etc. Lorsque la connexion est établie avec commutateur, le contrôleur l'envoie une requête de type *FEATURE_REQUEST_MESSAGE* qui répond avec un message de type *FEATURE_REPLY_MESSAGE* contenant l'identifiant du commutateur, la liste des ports actifs ainsi que leurs adresses MAC respectives.

4.5.3 Découverte des liens entre commutateurs

La découverte des liens inter-commutateurs est d'une importance capitale, il permet en effet au contrôleur de disposer d'une topologie correcte et cohérente pour les différents services qu'il doit assurer. Les liens inter-commutateurs déterminent comment les commutateurs communiquent entre eux. Ils permettent aussi au contrôleur et aux applications de définir la meilleure manière d'utiliser le réseau en fonction de leur besoins. Le protocole OFDP [101] est utilisé par la plupart des contrôleurs SDN pour la découverte de la topologie. Cette section est consacrée à la présentation de ce protocole.

4.5.3.1 OpenFlow Discovery Protocol (OFDP)

Le protocole OFDP est lui-même basé sur le protocole LLDP [102]. OFDP est le protocole defacto de découverte de topologie dans le SDN. Il est implémenté par la plupart des contrôleurs actuels. OFDP se sert du protocole LLDP notamment pour annoncer les fonctionnalités implémentées par les commutateurs ainsi que les informations de voisinage des nœuds du réseau.

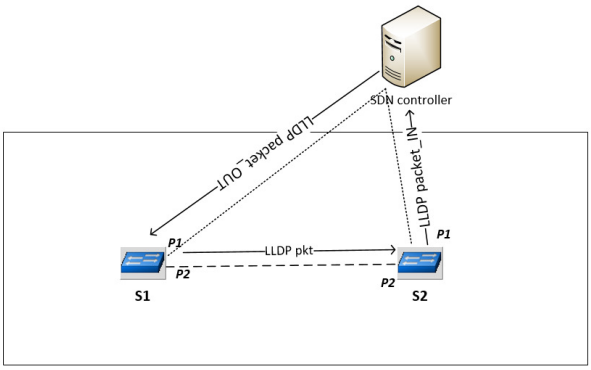


FIGURE 4.1 – Scénario basic du LLDP

Les informations extraites des paquets LLDP échangés dans le cadre de la découverte des liens sont stockées dans une base de données Management Information Base (MIB) au sein des commutateurs, ces informations sont accessibles via un protocole tel que Simple Network Management Protocol (SNMP). Dans la version classique du protocole OFDP, les paquets LLDP sont envoyés via tous les ports actifs des commutateurs. Un paquet LLDP est encapsulé dans une trame Ethernet (*IEEE 802.1AB, EtherType=0x88cc*). La trame LLDP présentée par la Figure 4.2 comporte une information appelée LLDP Data Unit (LLDPDU) contenant un certain nombre de structures de type Type-Length-Value (TLV). Les TLV obligatoires sont l'identifiant unique de chaque commutateur (*Chassis ID*), l'identifiant unique de port *Port ID* et le Time to Live (TLL).

Tous les commutateurs OpenFlow disposent d'une règle pré-installée par défaut dans leurs Table de « Flow » indiquant que tout paquet LLDP reçu sur n'importe quels ports à l'exception de celui *contrôleur* doit être transférer à ce dernier via un message *Packet-IN*.

La Figure 4.1 présente le fonctionnement du protocole OFDP dans un réseau constitué de deux commutateurs s_1 et s_2. Chaque commutateur dispose de deux ports actifs (p_1 et p_2). Pour la découverte du lien unidirectionnel $s_1 \rightarrow s_2$, le contrôleur encapsule le paquet LLDP dans un message *Packet-OUT* et l'envoie à s_1. Le message *Packet-OUT* contient les instructions pour envoyer le message LLDP à s_2 via le port p_2 de s_1. En recevant le paquet LLDP via le port p_2, s_2 l'encapsule dans un message *Packet-IN* et le renvoie en direction du contrôleur. A la réception du paquet LLDP, le contrôleur conclut qu'il existe une liaison unidirectionnelle de $s1$ vers $s2$. Le même processus est effectué pour la découverte du lien dans le sens $s_2 \rightarrow s_1$.

Avec les réseaux de plus en plus dynamiques tels que les réseaux du Cloud à « multi-tenant » ou encore les « data-center », le contrôleur doit avoir à tout instant une vision cohérente et correcte sur l'état de l'infrastructure. Disposer des données à jour sur la topologie par le contrôleur (la couche contrôle) est une des fonctions les plus critiques du SDN. Les commutateurs/routeurs arrivant ou quittant le réseau créent ainsi dynamiquement des changements dans la topologie qui affectent les décisions de routage que le contrôleur doit prendre en permanence. Pour disposer toujours d'une topologie à jour, le contrôleur doit répéter les étapes suivant le processus décrit dans la Figure 4.1 périodiquement. La périodicité de ce processus doit être définie avec soin en tenant compte des paramètres tels que la

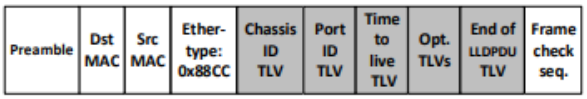


FIGURE 4.2 – Structure d'une trame LLDP

stabilité, l'instabilité de la topologie, la complexité, la taille, les capacités (CPU, RAM, capacités des liaisons réseaux...). Une période de *15 secondes* tel que le cas par défaut pour le contrôleur *FloodLight* peut ne pas convenir à un réseau très dynamique, car elle peut introduire un retard pouvant atteindre les *15 secondes* avant qu'un changement de topologie ne soit pris en compte. De la même manière une période assez courte d'*une ou deux secondes* par exemple ne peut également pas convenir dans le cas d'un un réseau très peu dynamique avec une taille importante et une certaine complexité. En effet, cela consommerait ainsi inutilement les ressources du réseau (couches contrôle et infrastructure) et pourrait affecter considérablement les performances du contrôleur.

Ainsi à la fin de chaque période T que nous avons appelé *cycle de découverte de la topologie* dans le cadre des travaux de cette thèse, le contrôleur déclenche l'envoi des messages *Packet-OUT* selon les équations suivantes :

$$P_{IN\_OFDP} = 2L \tag{4.5.1}$$

$$P_{OUT\_OFDP} = \sum_{i=1}^{N} P_i \tag{4.5.2}$$

P_{IN_OFDP} et P_{OUT_OFDP} correspondent respectivement aux nombres de messages de type *Packet-IN* et de type *Packet-OUT* envoyés pour un *cycle de découverte de la topologie*. $N = |\{S_1, S_2, ..., S_N\}|$ détermine le nombre de commutateurs dans la topologie. L représente le nombre de liens unidirectionnels entre les commutateurs *OpenFlow* présents dans le réseau et P_i correspond au nombre de ports actifs du commutateur S_i.

4.5.3.2 OFDP version 2

La charge et les performances du contrôleur sont des éléments critiques pour le passage à l'échelle du SDN. Le processus de la découverte de la topologie est un service s'exécutant généralement en arrière plan des contrôleurs, il est important que le protocole utilisé soit optimisé pour minimiser les ressources qui lui sont consacrées. La charge du contrôleur dû à OFDP est déterminée par le nombre de messages *Packet-OUT* envoyés aux commutateurs et le nombre de messages *Packet-IN* qu'il doit recevoir et traiter. Les équations 4.5.1 et 4.5.2 montrent le nombre de messages utilisés OFDP dans sa versions classique pour un cycle de la découverte de topologie.

Dans [103], les auteurs ont proposé une évaluation du « overhead » de la version d'OFDP actuellement implémentée par les contrôleurs SDN, ils ont également proposé dans le cadre de la même étude une optimisation du protocole OFDP qu'ils ont appelé OFDP *version 2* (*OFDPv2*). Ils ont démontré

qu'OFDP a permis de réduire l'« overhead » par rapport à la version classqiue avec un gain *G* de l'ordre de *45%* à *75%* en fonction de la complexité de la topologie.

$$P_{IN\_OFDP} = P_{IN\_OFDPv2} = 2L \tag{4.5.3}$$

$$P_{OUT\_OFDPv2} = N \tag{4.5.4}$$

$$G = \frac{P_{OUT\_OFDP} - P_{OUT\_OFDPv2}}{P_{OUT\_OFDP}} = \frac{\sum_{i=1}^{N} P_i - N}{P_{OUT\_OFDP}} = \frac{\sum_{i=1}^{N} P_i - N}{\sum_{i=1}^{N} P_i} = 1 - \frac{N}{\sum_{i=1}^{N} P_i} \tag{4.5.5}$$

Comme le montre les équations (4.5.1) et (4.5.3), les protocoles OFDP et OFDP*v2* ne présentent aucune différence au regard du nombre de messages *Packet-IN* reçus et traités par le contrôleur. Avec les deux versions du protocole, le nombre de messages reçus et traités est égale au double de liens unidirectionnels entre les commutateurs Openflow présents dans le réseau.

La principale modification apportée dans OFDP*v2* est la réduction du nombre de messages *Packet-OUT* envoyés par le contrôleur aux commutateurs OpenFlow. En effet, au lieu d'envoyer un message *Packet-OUT* par port actif de chaque commutateur comme présenté par l'équation 4.5.2 ($\sum_{i=1}^{N} P_i$), ce nombre a été réduit à seulement N correspondant au nombre de commutateurs OpenFlow présents dans le réseau.

L'équation 4.5.5 présente le gain (*G*) de OFDP*v2* par rapport à OFDP. En prenant l'exemple la *Topologie.1* présentée dans le Tableau 4.3 qui est constituée de *63* commutateurs, de *124* liens unidirectionnels et de *188* ports actifs,le gain est de *67%*. D'autres travaux tels que ceux présentés par dans [104, 105, 106] présentent une amélioration d'OFDP*v2*

4.6 Plate-formes expérimentales

Dans cette section nous présentons la plate-forme utilisée pour réaliser nos expériences pour l'analyse des performances des contrôleurs SDN. Notre étude a été principalement focalisée sur ONOS et ODL qui sont aujourd'hui les contrôleurs les plus populaires. Nous allons présenter premièrement une description de la plate-forme utilisée pour la réalisation de différentes expériences avant de proposer une description de chacune des expériences dans un second temps.

4.6.1 Architecture de la plate-forme expérimentale

La plate-forme d'expérimentation est constituée de trois serveurs DELL Precision Tower 7810 disposant des caractéristiques suivantes : Intel (R) Xeon (R) CPU E5-2630 v3 at 2.40 GHz, 32 GB DDR avec 2NICs de 1GBps. Ces serveurs connectés au réseau du Laboratoire Informatique de Paris 6 (LIP6). Dans le cas de l'architecture avec deux contrôleurs (*cluster*), les deux serveurs qui jouent le rôle des contrôleurs sont directement connectés via un réseau local dédié.

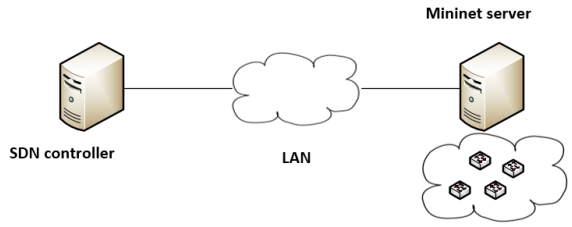


FIGURE 4.3 – Plate-forme expérimentale avec un seul contrôleur

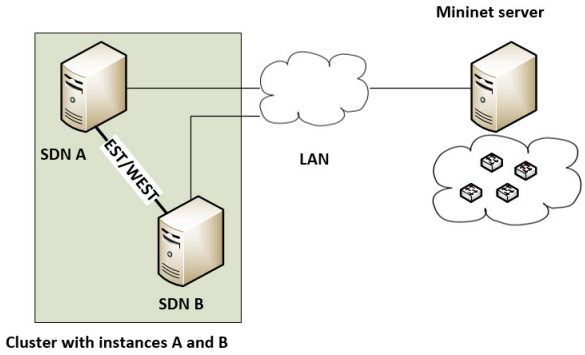


FIGURE 4.4 – Plate-fome expérimentale avec deux contrôleurs

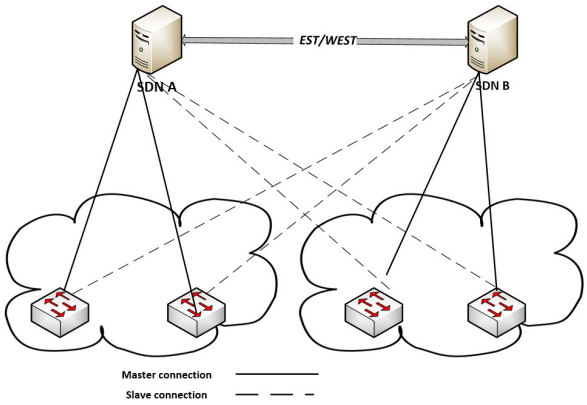


FIGURE 4.5 – Cluster avec deux instances

La première série d'expérimentation a pour but de mesurer le temps de découverte de la topologie dans le cas d'un contrôleur unique ainsi que celui nécessaire pour mettre à jour sa vision de la topologie suite à un changement (arrivée, départ d'un commutateur, mise hors tension ou activation d'une interface réseau d'un commutateur). Cette série d'expériences est réalisée par le biais de la plate-forme présentée par la Figure 4.3. Elle est constituée de deux serveurs comme présentée à la Figure 4.3. Le

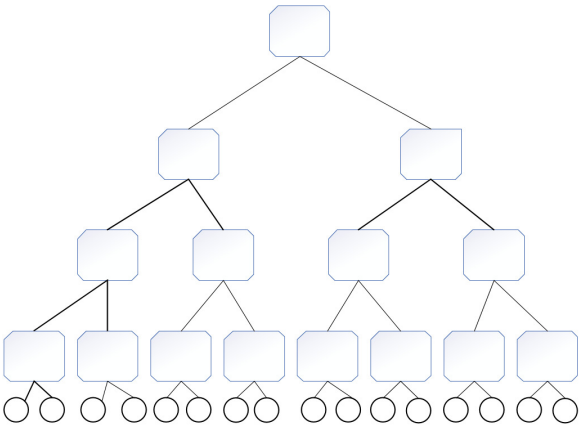


FIGURE 4.6 – Exemple de topologie en arbre binaire avec une profondeur $d = 4$

premier serveur est utilisé pour héberger le contrôleur SDN à tester *(ODL ou ONOS)*. Le second serveur servira à exécuter l'émulateur de réseaux virtuels *Mininet version 2.2.1* [12] qui émule l'infrastructure d'un réseau SDN avec des commutateurs virtuels, des hôtes virtuels, les liens inter-commutateurs ainsi que les liens entre commutateurs et hôtes. Pour chaque expérience, nous avons lancé la topologie désirée sur la machine *Mininet* et nous avons ensuite capturé le trafic au niveau du contrôleur à l'aide de l'outil *tcpdump*. Le trafic capturé a été par la suite analysé par *Wireshark*.

La Figure 4.3 présente la plate-forme utilisée pour la réalisation de cette série d'expériences. Sur le premier serveur est installé le contrôleur à étudier et sur le second on installe l'emulateur SDN *Mininet* [12]. Pour chaque expérience de cette série, nous lançons la topologie concernée sur le serveur *Mininet* et capturons le trafic sur un fichier que nous analyserons avec *wireshark* par la suite.

La seconde série d'expérimentation a pour but de mesurer le temps de découverte de la topologie dans le cas d'un environnement avec plusieurs contrôleurs (multi-contrôleurs). Cette série d'expériences a été réalisée par le biais d'une plate-forme illustrée par la Figure 4.4. En plus de deux serveurs présentés utilisés dans le cadre de l'expérience avec un contrôleur unique, un troisième serveur est utilisé pour construire un « cluster » de deux contrôleurs. ONOS et ODL utilisent respectivement les ports *TCP 9876* et *2550* pour la communication inter-contrôleurs. La communication *EST-OUEST* est présentée dans la Figure 4.4. Ces numéros de port sont utilisés dans *Wireshark* pour filtrer le trafic entre les contrôleurs dans nos mesures. La figure 4.5 présente quant à elle la plate-forme utilisée pour la réalisation de cette série d'expérimentation.

4.6.2 Méthodologie pour mesurer le temps de découverte de la topologie

Dans cette section nous présentons la méthodologie utilisée pour mesurer le temps de découverte de la topologie.

Pour mesurer les performances des contrôleurs en ce qui concerne la découverte de la topologie, nous avons généré trois topologies de réseau en arbre avec *Mininet*. Chacune est caractérisée par trois

paramètres : le nombre de nœuds dans le réseau N, la profondeur de l'arbre d et le nombre de branches par niveau (« fanout ») f.

La Figure 4.6 montre une topologie avec 15 commutateurs et 16 hôtes où les carrés et les cercles représentent respectivement les commutateurs et les hôtes. Cette topologie a une la profondeur $d = 4$ et un nombre de branches « fanout » $f = 2$ parce qu'il y a 4 niveaux de commutateurs et chaque commutateur au niveau i est connecté à 2 commutateurs au niveau $i+1$. La relation entre ces paramètres N, d et f est présentée par l'équation 4.6.6 et peut être facilement vérifiée à ladite figure.

Le Tableau 4.3 présente les trois topologies utilisées pour la réalisation de nos expériences ainsi que les paramètres clés de chacune d'elles. Le nombre de liens unidirectionnels interconnectant les commutateurs dans le réseau, à l'exception des liens vers les hôtes, est égal à $2 \times (N-1)$. Le nombre total de ports, y compris les ports connectés aux hôtes, est calculé comme suit : $2 \times (N-1) + f^d$

$$N = \sum_{i=0}^{d-1} f^i \tag{4.6.6}$$

Pour analyser le temps de découverte de la topologie, nous avons suivi la procédure composée des étapes suivantes :

- *Étape 1 :* Nous lançons le serveur (le cas du contrôleur unique) ou les deux contrôleurs (pour les cas des contrôleurs multiples).
- *Étape 2 :* Sur le serveur (*Mininet*), nous lançons la topologie avec *Mininet*.
- *Étape 3 :* Nous attendons la fin du premier cycle du LLDP, qui prend plus de temps que les autres cycles, notamment en raison du processus d'établissement de la connexion entre les commutateurs et le contrôleur.
- *Étape 4 :* Nous enregistrons le temps d'envoi du premier message LLDP du contrôleur aux commutateurs.
- *Étape 5 :* Nous enregistrons ensuite le temps de réception par le contrôleur du dernier message LLDP envoyé par les commutateurs.
- *Étape 6 :* Pour chaque cycle, nous calculons T_{cycle} qui est la différence entre le temps d'envoi du premier message LLDP ($T_{First\_Packet\_OUT}$) par le contrôleur et l'heure de réception du dernier message *LLDP* ($T_{Last\_Packet\_IN}$)

$$T_{cycle} = T_{Last\_Packet\_IN} - T_{First\_Packet\_OUT} \tag{4.6.7}$$

4.7 Analyse des résultats

Dans cette section nous présentons les résultats obtenus sur l'évaluation des performances des contrôleurs ONOS et ODL sur la découverte de la topologie. Cette section est comme suit : Dans la sous-section 4.7.1 nous présentons les résultats obtenus sur la découverte de la topologie dans le cas d'un seul contrôleur. Puis nous présentons les mêmes analyses dans le cas d'un environnement avec plusieurs contrôleurs configurés en *cluster* dans la sous-section 4.7.2.

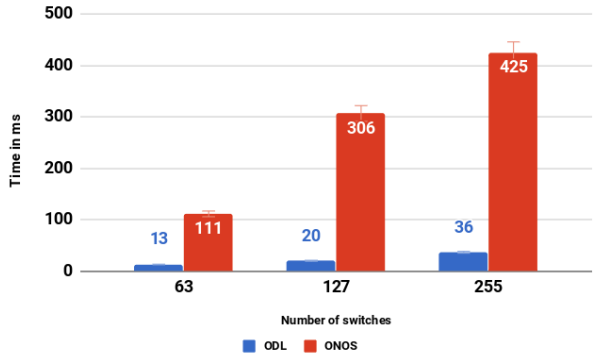


FIGURE 4.7 – Temps de découverte de topologie des contrôleurs ONOS et ODL en "single mode"

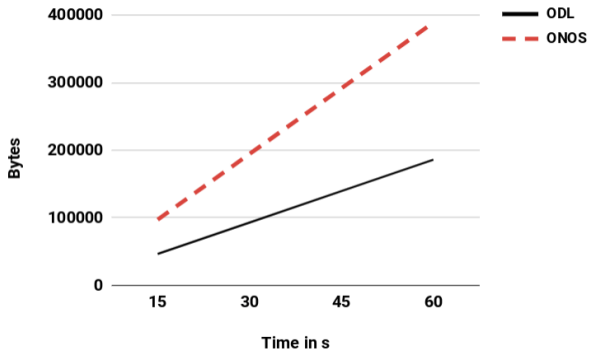


FIGURE 4.8 – Volume de données de contrôle pour la découverte de topologie avec ONOS et ODL

4.7.1 Découverte de la topologie : Cas d'un contrôleur unique

La Figure 4.7 montre le temps moyen (*T_cycle*) d'un cycle de découverte topologique pour ODL et ONOS dans le cas d'un contrôleur unique. Nous pouvons observer que le temps de découverte de la topologie augmente avec la taille du réseau mesurée en nombre de commutateurs. Nous pouvons également observer que le contrôleur *Opendaylight (ODL)* offre de meilleures performances sur cette métrique par rapport à ONOS. Pour la topologie disposée en arbre binaire de *63* commutateurs, *ODL* n'a besoin que de *13 ms* en moyenne pour découvrir la topologie complète alors que ONOS prend *111 ms*. Pour la topologie arborescente avec *127* commutateurs, ODL consomme *20 ms* contre *306 ms* pour ONOS.

La meilleure performance du contrôleur ODL par rapport à celui d'ONOS en termes de temps de découverte de topologie en mode contrôleur unique s'explique par l'implémentation du protocole OFDP dans chaque contrôleur. ODL implémente une version d'OFDP assez proche d'OFDP*v2* qui peut réduire jusqu'à 80% [103] le nombre de messages de contrôle nécessaires pour un cycle de découverte topologique. Le contrôleur ONOS implémente la première version d'OFDP qui n'offre aucune optimisation sur le nombre de messages de contrôle.

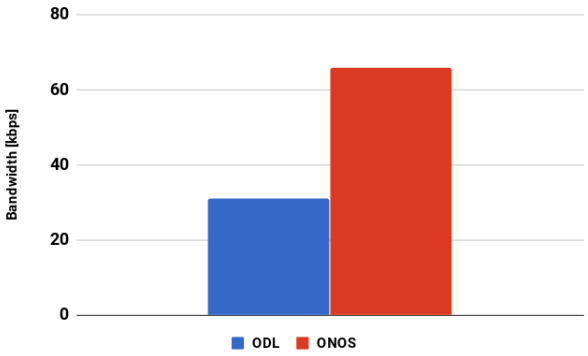


FIGURE 4.9 – Mesure du trafic inter-contrôleurs

Pour obtenir ces résultats, nous avons utilisé la même topologie pour les tests avec les deux contrôleurs. Pour chaque test, nous démarrons la topologie comme décrit dans la section 4.6, puis analysons le trafic enregistré généré par le filtrage *Wireshark* par les paquets LLDP. La Figure 4.8 montre que pour la même topologie, ONOS utilise presque un double volume de messages de contrôle pour la découverte topologique. Nous pensons que l'ONOS devrait mettre en œuvre OFDPv2 pour réduire le sur-coût introduit par les messages de contrôle liés au processus de découverte topologique.

4.7.2 Découverte de la topologie : Cas de plusieurs contrôleurs (*cluster*)

La Figure 4.10 montre que le temps de découverte de topologie en mode « cluster » est plus important que celui en mode contrôleur unique pour une même topologie. Ceci est dû au temps dont les instances de « cluster » ont besoin pour synchroniser leur vue de topologie. Nous pouvons également observer que ODL dépasse également ONOS en mode « cluster » pour ce qui est de la découverte de la topologie. Pour une topologie disposée en arbre binaire avec *63* nœuds par exemple, ODL prend trois fois plus de temps que le temps nécessaire en mode simple (*39* ms en mode « cluster » contre *13 ms* en mode contrôleur unique) alors que ONOS prend plus de treize fois plus de temps (de *111 ms* en mode contrôleur unique à plus de *1500 ms* en mode « cluster »). Bien qu' ONOS et ODL utilisent le même protocole en partie pour assurer la cohérence des données en mode « cluster », leur implémentation est cependant différente. Dans cette étude [107] portant sur le contrôleur ONOS montre que le nombre de messages de contrôle reste élevé même en absence de toute topologie, ce qui explique les mauvaises performances en mode « cluster ».

4.8 Conclusions

Les tâches liées à la gestion de la topologie par un contrôleur SDN ont un impact très important sur les performances du réseau. Dans ce chapitre qui a fait l'objet d'une publication dans une conférence internationale, nous avons évaluer expérimentalement les performances des contrôleurs ONOS et ODL en termes de temps de découverte de la topologie. Les mesures du temps de découverte de la topologie ont été effectuées à la fois dans le cas d'un contrôleur unique et mais également dans d'un environnement multi-contrôleurs appelé « cluster ». Les résultats obtenus montrent que ODL offre de

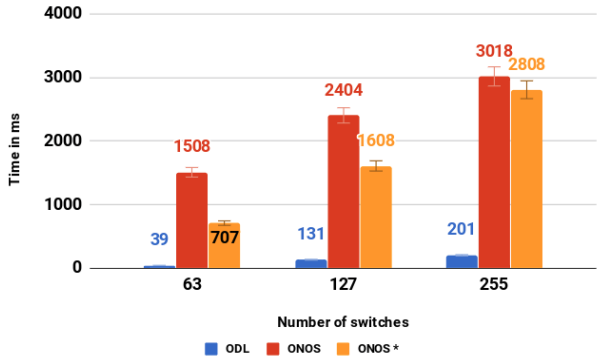


FIGURE 4.10 – Temps de découverte de topologie des contrôleurs ONOS et ODL en mode cluster

meilleures performances sur la découverte de la topologie.

En conclusion, le choix d'un contrôleur dépend largement de son domaine d'application, cependant sur un aspect fondamental du SDN qui est la gestion de la topologie, la facilité de mise en oeuvre, la facilité du déploiement et de configuration d'un environnement avec plusieurs contrôleurs en « cluster », la qualité de l'interface graphique et du Command Line Interface (CLI). Dans le chapitre 5 nous nous intéresserons à la problématique liée au changement de la topologie.

TABLE 4.1 – Critères de comparaison des contrôleurs SDN

	ONOS	ODL	Floodlight	RYU	POX	NOX
Langage	Java	Python	Java	Java	Python	C++
GUI	Iterface web	Iterface web	Interface	Interface web		
Documentation	Très bonne	Très bonne	bonne	bonne	moyenne	moyenne
Plate-formes	Linux, Windows, Mac OS	Linux, Windows, Mac OS	Linux, Windows, Mac OS	Plutôt Linux	Linux	Plutôt Linux
Centralisée/Distribuée	D	D	C	C	C	C
APIs Nord	API REST	API REST		API REST	API REST	API REST
APIs I sud	OF1.0, 1.3, 1.4, NETCONF, OVSDB, BGP/LS LISP, SNMP	OF1.0, 1.3, 1.4, NETCONF, OVSDB, PCEP, BGP/LS, LISP, SNMP	OF1.0-1.3	OF1.0-1.4 NETCONF OFCONFIG	OF1.0	OF1.0
Partenaires	ON.LAB, AT and T, Ciena, Cisco, Ericsson, Fujitsu, Huawei, Intel,NEC Nsf.Ntt, Comunnication, Sk Telecom	Linux Foundation with Memberships Covering Over 40 Companies, Such As Cisco, IBM, NEC	Nippo Telegraph et Telephone Corporation	Big Switch Networks	Nicira	Nicira
Multithreading	OUI	OUI	OUI	OUI	NON	NOX MT
OpenStack	OUI	OUI	OUI	OUI	Non	Non
Domaine	Datacenter,WAN Transport	Datacenter	Campus	Campus	Campus	Campus

TABLE 4.2 – Méthodes pour la découverte de topologie dans le SDN

Entités	**Moyen de découverte**
Hôtes	*Packet_IN*
Commutateurs	*Processus de « Handshake » initial*
Liens (entre commutateurs)	LLDP

Topologies	Parametères	Liens unidirectional	Ports
Topologie.1	$N = 63, d = 6, f = 2$	124	188
Topologie.2	$N = 127, d = 7, f = 2$	252	380
Topologie.3	$N = 255, d = 8, f = 2$	508	764

TABLE 4.3 – Paramètres clés des topologies

Chapitre 5

Évaluation des performances : Tolérance aux pannes

Sommaire

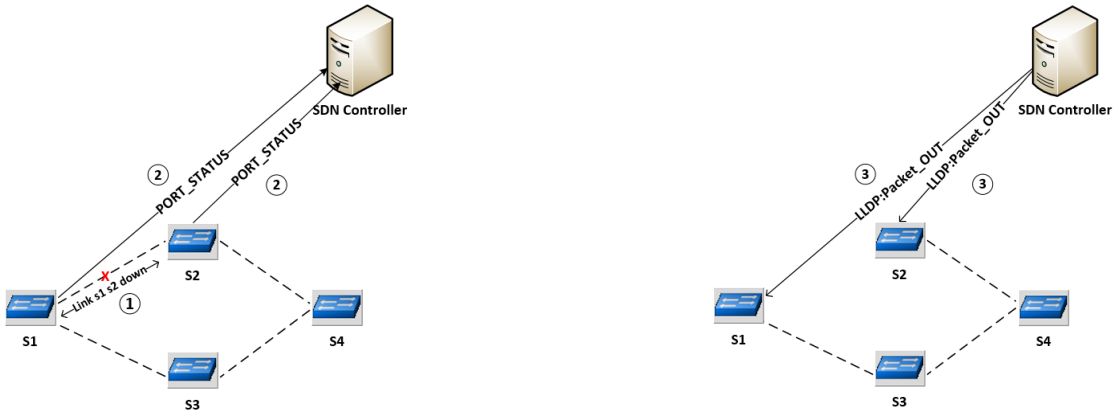
5.1 Introduction 87
5.2 Changement de la topologie dans le SDN 88
5.3 Temps d'interruption avec les contrôleurs ONOS et ODL : Cas d'un contrôleur unique 89
5.4 Distribution du temps d'interruption avec les contrôleurs ONOS et ODL 92
5.5 Tolérance aux pannes : Architecture CARP 93
5.6 Conclusions 95

5.1 Introduction

Pour assurer la mise à jour de la topologie, la plupart des contrôleurs utilisent le protocole OFDP. Cependant, l'implémentation de ce protocole peut varier d'un contrôleur à un autre ce qui n'est pas sans conséquence sur leurs performances. La gestion du changement de la topologie est une fonction critique avec les réseaux de plus en plus dynamiques et particulièrement dans le cas du SDN. Tout changement dans la topologie du réseau doit être immédiatement notifié et traité par le contrôleur pour assurer la continuité du service [14].

Dans le chapitre 4, nous avons présenté en détails les mécanismes de gestion de la topologie dans le SDN, nous avons également analysé les performances des contrôleurs ONOS et ODL selon les critères suivants :

— la découverte de la topologie
— le trafic de contrôle



(a) (b)

FIGURE 5.1 – Mécanisme de changement de la topologie

— la bande passante consommée pour la découverte de la topologie
— le nombre de requêtes traités par seconde (débit du contrôleur) et la latence

Les résultats ont montré que le contrôleur ODL offre les meilleures performances sur les trois premières critères cités tandis qu'ONOS est meilleure sur la latence et le nombre de requêtes traitées par seconde.

Dans ce chapitre, nous nous intéressons à l'analyse des performances des contrôleurs SDN (ONOS et ODL) sur le changement de la topologie.

Ce chapitre est organisé comme suit. Dans la section 5.2 nous introduirons le mécanisme de changement de la topologie dans le SDN puis dans la section 5.3 nous présenterons analyse comparative du temps d'interruption suite à une panne des contrôleurs ONOS et ODL. Nous présenterons une distribution du temps du temps de restauration de chemin suite à une panne des mêmes contrôleurs tandis la section 5.5 une évaluation des performances de l'architecture CARP présentée dans le chapitre 3 sur la tolérance aux pannes et dans la section 5.6 nous présenterons une conclusion.

5.2 Changement de la topologie dans le SDN

Les contrôleurs SDN doivent maintenir une vue à jour du réseau dans un environnement en quasi temps réel afin de permettre aux applications de fonctionner avec une vue cohérente. C'est l'une des clés de son bon fonctionnement. D'une manière générale, la mise à jour de la topologie est implémentée selon une logique de modèle basée sur les événements. Ces évènements sont déclenchés suite à la réception par le contrôleur des paquets spécifiques (*OFPT_PORT_STATUS*) envoyés par les commutateurs. Cet événement est reçu et organisé par les *listeners* abonnés, qui à leur tour solliciteront des changements de représentation topologique dans une base de données, éventuellement distribuée.

Nous nous concentrons en particulier sur ce qui se passe dans les contrôleurs ONOS et ODL lors-

qu'un paquet *OFPT_PORT_STATUS* est reçu d'un commutateur pour notifier un changement d'état d'un port après un événement de perturbation (rupture) de liaison ou après l'établissement ou le rétablissement d'un lien.

Le changement de la topologie intervient suite aux évènements suivants :

— L'activation ou la dés-activation d'une interface : Ces événements sont notifiés au contrôleur par des messages de type *Port-Status* envoyés par les commutateurs vers le contrôleur SDN. Ce message indique au contrôleur que l'état du port a changé (d'active à non active ou vice-versa).

— L'ajout ou la suppression d'un noeud : L'ajout d'un noeud génère les mêmes messages en plus de ceux nécessaires à l'établissement de la connexion TCP entre le nouveau noeud et le contrôleur SDN. Il faut également prendre en compte les messages *HELLO* et autres messages échangés entre le contrôleur et les équipements sous son contrôle.

La Figure 5.1 présente le mécanisme générale de changement de la topologie dans le SDN. Cette figure est composée de deux sous-figures (a) et (b), constituées chacune de 4 commutateurs avec 8 liens unidirectionnels. Ces figures montrent les étapes du changement de topologie après la rupture d'un lien :

— *Étape 1 :* Le lien entre $S1$ et $S2$ est désactivé (Figure (a))

— *Étape 2 :* Les commutateurs $S1$ et $S2$ envoient chacun un message de type *PORT_STATUS* pour notifier au contrôleur le changement d'état (*up* to *down*) de son port (Figure (a))

— *Étape 3 :* Le contrôleur calcule la nouvelle topologie puis installe via un message de type *Packet_OUT* les nouvelles règles sur les commutateurs (Figure (b))

5.3 Temps d'interruption avec les contrôleurs ONOS et ODL : Cas d'un contrôleur unique

La Figure 4.3 présente la plate-fome utilisée. Elle est constituée de deux serveurs physiques dont nous avons fait la présentation dans la section 4.6. Pour analyser le temps de réaction du contrôleur suite à un changement dans la topologie , nous avons réalisé une expérience avec un cas de test basique avec la topologie *GEANT* présentée à la Figure 5.2 disponible sur la plate-forme des topologies de test *topology-zoo* [108]. Puis, nous avons identifié deux commutateurs pour lesquels il n'existe qu'un seul chemin le plus court, nous avons ensuite connecté à chacun d'eux une machine terminale (hôte). La raison pour laquelle il est important de choisir deux commutateurs avec un seul chemin le plus court s'explique notamment par le fait que le contrôleur ONOS pré-calcule de manière pro-active pour chaque couple de noeuds l'ensemble des plus courts chemins possibles. Ainsi, pour un couple de noeuds avec plusieurs chemins possibles donc calculés pro-activement la basculement(suite à une défaillance de premier chemin sélectionné par exemple) d'un chemin à un autre se fait quasi-instantanément avec ce contrôleur.

Le choix de *GEANT* comme topologie de test s'explique aussi par le fait que c'est une topologie réelle, avec une taille raisonnable (40 commutateurs, 122 liens). Pour le reste de l'expérience nous avons procédé comme suit :

- *Étape 1 :* Sur le serveur dédié à *Mininet* (Figure 4.3), nous lançons la topologie du réseau avec *Mininet* puis attendions que toutes les connexions entre les commutateurs et le contrôleur soient

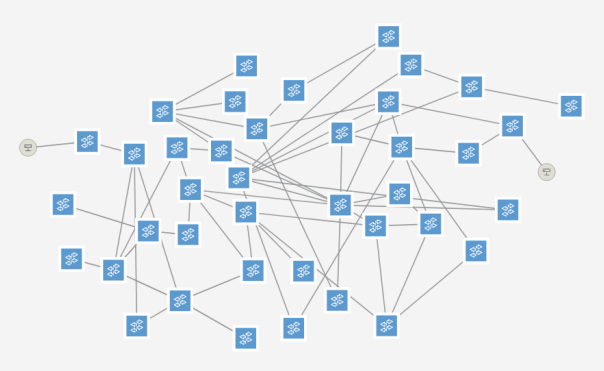


FIGURE 5.2 – Topologie du réseau *GEANT* exécutée sur ONOS

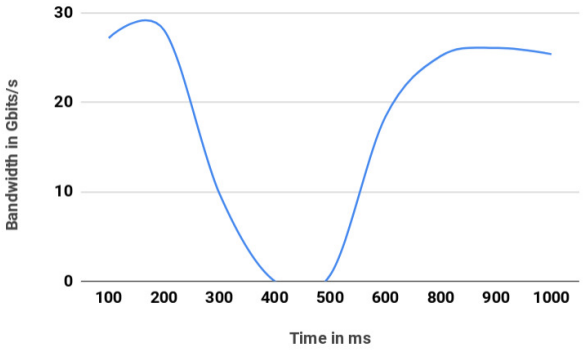


FIGURE 5.3 – Temps d'interruption en changement de topologie avec ONOS en "single mode"

totalement stabilisées. Nous vérifions la connectivité entre les deux machines terminales avec un *ping*.

- *Étape 2 :* Sur *Mininet* on lance les deux terminaux avec la commande *Xterm*
- *Étape 3 :* Sur un terminal on lance le serveur *iperf*. Puis sur le second terminal on lance le client. Nous avons présenté dans l'annexe A.5 les commandes pour la configuration d'un serveur et d'un client *iperf*.
- *Étape 4 :* Quelques secondes après le lancement du client *iperf*, nous désactivons un lien préalablement identifié de manière à provoquer au niveau du contrôleur le calcul d'un chemin pour transférer les paquets la session *iperf* ouverte précédemment.
- *Étape 5 :* Supposons que T_0 est le moment où le contrôleur reçoit la notification de la désactivation du lien décrite dans l'*étape 4* et T_{react} est l'instant où le contrôleur a fini d'installer au niveau des commutateurs la nouvelle route. Entre les instants T_0 et T_{react}, tous les paquets envoyés dans la session *iperf* sont perdus. L'instant $T_{react} + \epsilon$ le premier paquet passe par le nouveau chemin. Le temps de changement de la topologie correspond approximativement à la différence entre T_{react} et T_0.

Dans cette section, nous présentons les résultats de nos évaluations sur la promptitude et la célérité avec laquelle les contrôleurs ODL et ONOS effectuent les actions de mise à jour de la topologie.

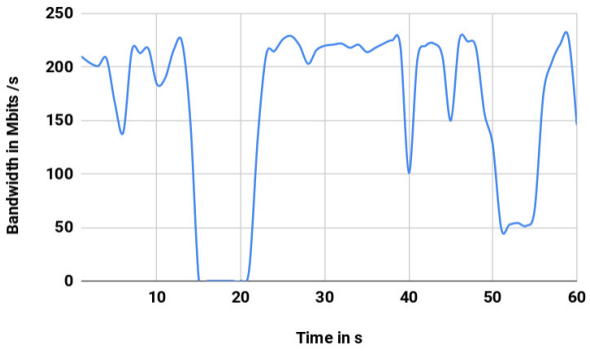


FIGURE 5.4 – Temps d'interruption en cas de changement de topologie avec ODL en "single mode"

Les Figures 5.3 et 5.4 montrent respectivement les courbes de trafic sur une session *iperf* de bout en bout. L'abscisse présente le temps et l'ordonnée le trafic (la quantité de données échangées dans la session *iperf*). Les unités de temps sont respectivement en *millisecondes* et en *secondes* pour les Figures 5.3 et 5.4. Ainsi un trafic nul signifie que tous les paquets envoyés dans la session *iperf* sont perdus. Les durées des périodes de trafic nul dans ces deux figures correspondent donc aux temps d'interruption de trafic respectivement pour des contrôleurs ONOS et ODL mesurés sur une connexion *TCP* de bout en bout dans la topologie *GEANT* émulée avec *Mininet* pour les tests dans le cas d'un contrôleur unique. Nous pouvons observer qu'ONOS offre un temps d'interruption du service beaucoup plus faible qu'ODL. Une désactivation de liaison sur un chemin a un impact très faible sur le trafic utilisateur avec un temps d'arrêt du service inférieur à une *seconde* en moyenne alors que le temps d'interruption du service avec ODL est supérieur à 5 secondes pour le même scénario de test.

La réaction rapide d'ONOS en cas de changement de topologie est due à deux facteurs. Premièrement, dès la réception d'un message *OFPT_PORT_STATUS* déclenché par un événement *link down* ou *link up*, le contrôleur ONOS traite immédiatement cette notification et les messages *Packet_OUT* correspondants sont envoyés aux commutateurs concernés. Deuxièmement, ONOS calcule à l'avance tous les chemins équidistants entre tout couple de nœuds du réseau. Ainsi avec le contrôleur ONOS lorsque le chemin sélectionné par défaut n'est plus disponible (suite à une défaillance par exemple), le trafic est immédiatement commuté sur un autre chemin équivalent s'il en existe. Sinon, il recalcule un autre plus court chemin.

Le manque de réactivité d'ODL suite à un changement dans la couche infrastructure d'un grand réseau est dû au fait que ce contrôleur prend plus de temps pour traiter une notification reçue via un message de type *OFPT_PORT_STATUS*. En effet, à la réception de ce type de message notifiant un événement *link down* ou *link up*, ODL ne réagit pas immédiatement. Ce contrôleur attendra le prochain cycle LLDP, envoyés toutes les 5 *secondes* par défaut. Cette période peut être modifiée. Ainsi, le temps de réaction est plus grand lorsque l'interruption intervient juste après le dernier *cycle LLDP*. A l'inverse il est plus faible dans le cas où elle intervient avant le déclenchement du prochain. Lors du cycle suivant, le contrôleur prendra donc en compte le changement de la topologie intervenu précédemment. En outre, le trafic utilisateur n'est effectif qu'après l'envoie des messages de contrôle de type *FlowMod* à tous les commutateurs du réseau. Ainsi, avec ce contrôleur, le message *FlowMod* n'est pas envoyé par ONOS

lors d'une réaction suite à un changement de la topologie.

5.4 Distribution du temps d'interruption avec les contrôleurs ONOS et ODL

La Figure 5.5 présente la topologie utilisée pour réaliser ce test. Deux hôtes, $H1$ et $H2$, sont connectés par un seul chemin composé de 6 liens et 5 commutateurs SDN (commutateurs Open vSwitch (OVS)), et échangeant des paquets User Datagram Protocol (UDP) via une session *Iperf*. Nous utilisons la version *Quali* pour ONOS et la version *Oxygen* pour ODL. La topologie de la Figure 5.5 est déployée sur la plate-forme 4.3 tel que nous l'avons décrit dans la section 4.6.2. Nous avons également introduit le mécanisme de découverte de la topologie dans la section 4.5.3. Pour maintenir la topologie le contrôleur SDN envoient périodement des messages aux commutateurs selon les équations 4.5.1 et 4.5.2. Pour ONOS et ODL, cette période est respectivement de 3 et 5 secondes par défaut.

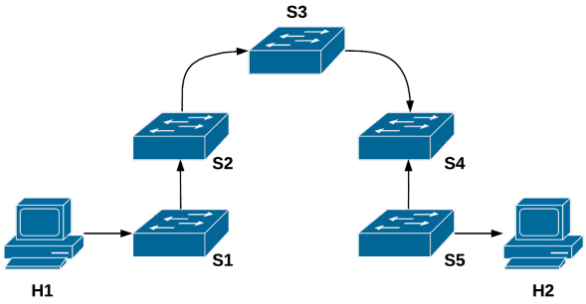


FIGURE 5.5 – Topologie du réseau du scénario de test

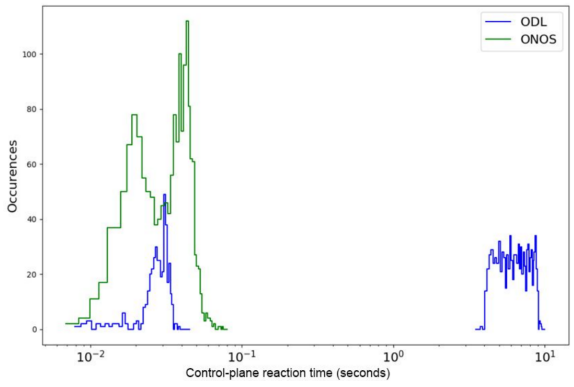


FIGURE 5.6 – Distribution du temps de réaction (PDF) pour des contrôleurs ONOS et ODL

La topologie présentée par la figure 5.5 montre qu'il n'existe qu'un seul chemin possible entre les hôtes $H1$ et $H2$. Pour chaque test, une session *Iperf* est mise en place entre $H1$ et $H2$. Notre expérience a consisté à désactiver aléatoirement un lien sur le chemin dans le but de causer des pertes de paquet

dans la session *Iperf* puis de ré-activer le lien immédiatement dans le but d'obliger le contrôleur à ré-calculer le chemin. Cet test est réalisé 1400 fois (*script automatisé*).

La Figure 5.6 présente la fonction de distribution de probabilité empirique (PDF) du temps de réaction des contrôleurs ONOS et ODL. Pour le contrôleur ODL, dans 70% des cas, le temps de réaction se situe dans l'intervalle entre 3 et 7 secondes alors que pour ONOS ce temps se situe entre 0.01 secondes et 0.06 secondes dans plus de 90% des cas mais pouvant atteindre 0.1 secondes. Nous avons analysé dans la section précédente les raisons de cette manque de performances pour le contrôleur ODL.

Les résultats présentés dans cette section ont publié dans le $3nd$ Dans [16] nous avons présenté une fonction de distribution de probabilité empirique (PDF) du temps de réaction suite à un changement de la topologie observée sur 1400 tests (1400 sessions *iperf*) sur les contrôleurs ONOS et ODL.

Le temps de réaction dépend aussi de plusieurs paramètres et particulièrement de la taille du réseau (nombre de noeuds), des capacités (CPU, (Random-Access Memory (RAM)),...) de calcul et de la configuration (physique ou virtuelle) du serveur qui héberge le contrôleur. D'autres paramètres peuvent également avoir un impact sur ce temps de réaction tels que l'emplacement (à distant ou en local) du contrôleur SDN qui peut impacter le temps aller-retour entre celui-ci et l'infrastructure du réseau notamment pour les notifications de défaillance (changement d'état d'un port d'un commutateur) d'un lien ou de l'installation des nouvelles règles pour l'acheminement du trafic. Nous avons également démontré dans le Chapitre 4 que l'environnement multi-contrôleurs (cluster) peut également avoir un impact sur le temps de réaction d'un contrôleur notamment à cause du temps nécessaire à la synchronisation entre les instances du cluster.

5.5 Tolérance aux pannes : Architecture CARP

Dans cette section nous allons présenter une expérimentation réalisée sur un réseau TRILL [109] et démontrer l'efficacité de notre solution pour augmenter la tolérance aux pannes. Nous allons montrer ensuite comment notre solution permet de réduire le temps de convergence dans les réseaux distribués utilisant les protocoles à état des liens augmentant ainsi la tolérance aux pannes.

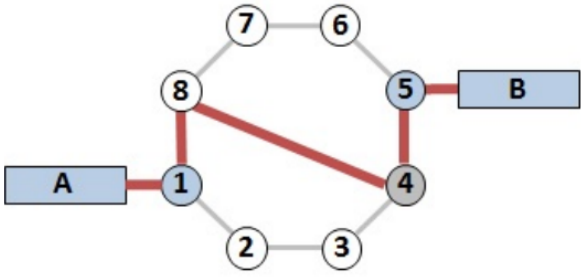


FIGURE 5.7 – Topologie d'un réseau basé sur TRILL

Cette expérimentation est réalisée dans un réseau TRILL [109] avec huit nœuds (RBridges) dans un environnement composé des machines virtuelles. La Figure 5.7 présente la topologie du réseau utilisée

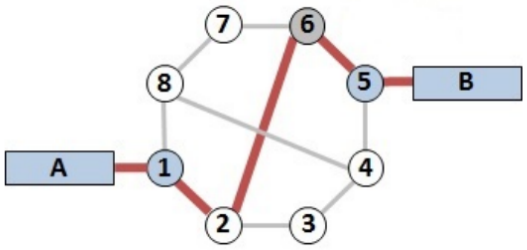


FIGURE 5.8 – Nouveau temps de convergence dans un réseau TRILL avec I2RS

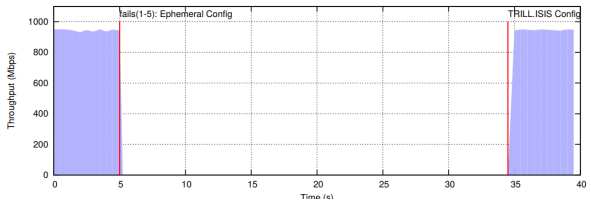


FIGURE 5.9 – Temps de convergence sans l'intervention du contrôleur

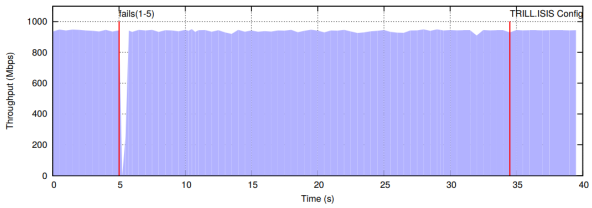


FIGURE 5.10 – Nouveau temps de convergence avec l'intervention du contrôleur

pour réaliser cette expérimentation. La machine virtuelle (VM) **A** envoie du trafic à la VM **B**. Deux chemins sont possibles avec le même coût : **A-1-2-6-5-B** et **A-1-8-4-5-B**. Le trafic est initialement envoyé par la machine A via le chemin **A-1-2-6-5-B**. Pour notre test, nous avons désactivé le lien entre les RBridges (5) et (6) afin d'évaluer la durée d'indisponibilité du réseau.

La figure 5.9 présente la durée d'indisponibilité du réseau suite à la défaillance d'une lien dans un réseau maillé TRILL. Nous observons que le trafic a été interrompu pendant **30 s**. Cette période correspond au temps nécessaire au protocole IS-IS utilisé par TRILL pour recalculer le chemin alternatif.

La figure 5.10 montre comment la période d'indisponibilité a été considérablement réduite lorsque le contrôleur reprend la main pour injecter le nouveau chemin au niveau de routeur (1). L'intervention du contrôleur a permis ramener la durée d'indisponibilité du système à seulement quelques millisecondes.

Cette architecture permet de détecter en quasi temps réel la défaillance du lien 5-6. L'agent associé au routeur détecte et notifie la panne au contrôleur. Le contrôleur active automatiquement le chemin alternatif au routeur(RBridge) **1**. Le trafic est alors immédiatement ré-acheminer via le chemin **A-1-8-**

4-5-B.

Les protocoles de routage distribués tels TRILL ou OSPF sont évolutifs et robustes, mais ils ne convergent pas rapidement en cas de défaillance d'une liaison. Pour réduire leur temps de convergence suite à une défaillance de lien par exemple, nous avons proposé un contrôleur centralisé qui injecte des entrées de routage éphémères dans le Routing Information Base (RIB) sur la base des chemins déjà calculés par les protocoles de routage multi-trajets. Le contrôleur centralisé permet une récupération immédiate des liens. L'évaluation de performance présentée est basée sur une émulation de l'agent I2RS a démontré l'efficacité de cette approche. L'architecture hybride présentée dans ce chapitre est un prototype d'un cas d'utilisation introduit notamment dans le cadre du projet du CARP dont l'objectif principal est de démontrer les avantages de l'introduction progressive du SDN.

La sécurité ainsi que la continuité du service sont des préoccupations majeures pour les opérateurs. Cette préoccupation devient davantage critique quand l'infrastructure de l'entreprise est constituée par des data-centers où plusieurs serveurs virtuels partagent physiquement les mêmes « hosts », la non disponibilité d'un serveur physique peut donc avoir un impact négatif majeur sur les services offerts par une entreprise à ses clients. Ce problème est potentiellement réglés par la mise en place d'une architecture redondante. Cependant, si la défaillance concerne un lien entre deux équipements réseaux dans un réseau traditionnel sous IS-IS ou OSPF, il faudra attendre :

— le temps nécessaire aux routeurs adjacents détectent la panne
— puis celui nécessaire au recalcule des tables de routage pour arriver à une nouvelle convergence réseau avant que le service ne ce rétablisse

Dans un réseau SDN, ce problème est réglé, car une défaillance d'un commutateur ou d'un lien est automatiquement connue du contrôleur qui agit conséquemment en modifiant rapidement les règles d'acheminement des données. Nous démontré les avantages pouvant être obtenus en introduisant un contrôleur avec une vue globale sur un réseau hybride. Dans le chapitre 4 nous nous intéresserons à la manière de choisir ce contrôleur en faisant une étude comparative des contrôleurs SDN actuellement proposés dans le monde open-source. En outre, au-delà de la tolérance aux pannes, cette approche pourra être envisagée pour diverses application pour l'ingénierie du trafic. Notre solution permet par ailleurs d'introduire progressivement le SDN sans modifier les équipements actuels. Cette approche satisfait donc parfaitement la contrainte visant à minimiser les coûts d'intégration du SDN. Elle permet également d'améliorer les performances des traditionnels sur des situations telles que :

— Réduire le temps de convergence en cas de défaillance d'un lien dans les réseaux traditionnels,
— Ré-router le trafic en cas d'attaque DDoS par exemple,
— Faire de l'équilibrage de charge, etc,...

5.6 Conclusions

Dans ce chapitre nous avons évalué expérimentalement les performances des contrôleurs ONOS et ODL sur leur rapidité de réaction suite à un changement de la topologie dans un environnement mono-contrôleur. Les résultats de nos expériences SDN montrent que le contrôleur ONOS réagit mieux aux pannes de liaison dans un réseau (changement de topologie), avec un temps de réaction inférieur à 100 ms (topologie *GEANT*). Nous avons également analysé les performances de l'architecture hybrides SDN proposée dans le cadre du projet CARP que nous avons présenté dans le chapitre 3.

Ce contrôleur répond mieux aux exigences de qualité de service. En effet, plus le temps de réaction après un changement de la topologie est court, mieux la continue et la qualité du service est garantie. ONOS est particulièrement adapté dans le cas réseaux très dynamiques c'est-à-dire dans lesquels les topologies changent fréquemment en fonction des services proposés. Le contrôleur ONOS est le mieux adapté pour adresser les problématiques présentées dans les chapitres 2 et 3. Dans le chapitre 6 nous analyserons les performances des contrôleurs sur le passage à l'échelle dans un environnement de *cluster*, le débit et le trafic de contrôle.

Chapitre 6

Performances du contrôleur et passage à l'échelle

Sommaire

6.1 Introduction . . . 99
6.2 État de l'art . . . 100
6.3 Passage à l'échelle . . . 101
6.4 Trafic de contrôle . . . 102
6.4.1 Approche réactive . . . 102
6.4.2 Approche pro-active . . . 102
6.4.3 Approche hybride . . . 103
6.5 Plate-forme expérimentale . . . 103
6.5.1 Débit du contrôleur . . . 103
6.5.2 Expérimentation pour la passage à l'échelle . . . 103
6.5.3 Trafic de contrôle . . . 106
6.6 Analyse des résultats . . . 107
6.6.1 Analyse du débit . . . 108
6.6.2 Trafic de contrôle . . . 108
6.6.3 Passage à l'échelle . . . 109
6.7 Conclusion . . . 109

6.1 Introduction

Depuis son émergence, le SDN fait face à plusieurs défis parfois propres à son architecture. Le point unique de défaillance à cause de son contrôleur centralisé, les performances du plan de contrôle en cas de croissance du trafic et des demandes du réseau, la robustesse et la résilience du réseau face

aux défaillances des liaisons et commutateurs du réseau et le contrôleur mais également la défaillance de liaison entre instances dans le cas d'un environnement multi-contrôleurs sont parmi les défis majeurs du SDN. Avec la 5G et la multiplication des objets connectés (IoT), le nombre d'appareils de toutes sortes connectés à l'Internet dépassera 45 milliards en 2020 [110, 111].

Pour augmenter la robustesse, la haute disponibilité et le passage à l'échelle du SDN la solution principale est la multiplication du nombre de contrôleurs assignés au contrôle d'un réseau. Cette approche offre certes des possibilités énormes et ouvre des grandes perspectives pour le déploiement progressive par les opérateurs. Cependant, il faudra noter que l'augmentation du nombre de contrôleurs par réseau n'est pas sans conséquence dans les performances globales du réseau en termes du temps de découverte de topologie, du temps de réaction au changements topologiques et de la consommation de bande passante dédiée aux échanges inter-contrôleurs. Nous analyserons l'impact du nombre de contrôleurs sur les performances du cluster pour la découverte et le maintien de la topologie dans un environnement à contrôleurs multiples. Enfin, nous analyserons également les performances des principaux contrôleurs SDN sur le passage à l'échelle c'est à dire sur leurs capacités à supporter une augmentation de la demande de trafic des utilisateurs.

Dans le chapitre 4, nous avons présenté les résultats sur l'évaluation des performances des contrôleurs ONOS et ODL sur le de temps de la topologie dans un environnement mono-contrôleur et multi-contrôleurs (« cluster »). Les résultats obtenus ont montré que le contrôleur ODL présente des meilleures performances sur la découverte de la topologie dans les deux cas. Dans le chapitre 5 nous avons les performances les mêmes contrôleurs sur le changement de la topologie. Les résultats obtenus ont montré que le contrôleur ONOS offrent les meilleures performances sur ce critère.

Ce chapitre est consacré à l'évaluation expérimentale des performances des contrôleurs ONOS et ODL sur le passage à l'échelle, sur le débit proposé et sur le trafic de contrôle. Trois type d'expérimentation seront réalisées sur trois plate-formes distinctes. Il est organisé comme suit : Dans la section 6.2 nous présentons une revue de la littérature portant sur les critères cités plus haut, dans la section 6.3 nous présentons la problématique du passage à l'échelle dans le SDN, tandis que dans la section 6.4 nous présentons celui du trafic de contrôle. Dans la section 6.5 nous présentons les plate-formes utilisées pour réaliser les différentes expérimentations et dans la section 6.6 nous présentons une analyse des résultats obtenus suivi d'une conclusion dans la section 6.7.

.

6.2 État de l'art

Dans cette section, nous présentons une revue de la littérature actuelle sur le passage à l'échelle de l'architecture SDN. Nous présentons les principales études qui adressent cette problématique.

Dans [112] les auteurs proposent un modèle pour estimer la consommation de bande passante due au trafic de contrôle sur l'interface sud du contrôleur SDN. Ils ont calculé le nombre de messages de contrôle OpenFlow nécessaires à l'installation des règles pour le trafic des applications de transfert entièrement réactives, dans lesquelles le commutateur demande des instructions au contrôleur chaque fois qu'un nouveau flux arrive. Ce travail a été réalisé sur le contrôleur ONOS. Alors que dans [113]

les auteurs proposent un nouveau protocole pour l'optimisation de la cohérence (*consistency)* dans le « cluster » ONOS. Dans [114], les auteurs ont fait le même travail sur le contrôleur OpendayLight qui est plutôt *proactive*, c'est-à-dire ce contrôleur pré-installe un certain nombre de règles pour le traitement de certains flux par avance lors de la phase de découverte de topologie. Dans [115], les auteurs présentent une évaluation des performances en terme du pasage à l'échelle du contrôleur *RYU* en mettant en œuvre de multiples scénarios de façon expérimentale avec des outils de simulation de *Mininet* et *iPerf*. Le contrôleur a été testé dans l'environnement de simulation en observant le débit du contrôleur et en vérifiant ses performances dans les conditions de réseau dynamique sur une topologie de type *Mesh* en augmentant à chaque fois de façon exponentielle le nombre de nœuds (commutateurs et machines terminales). Les chercheurs concluent que l'augmentation du nombre de noeuds impacte négativement sur les performances du contrôleur. Dans [116, 117] les auteurs proposent une approche basée sur une réduction des interactions du plan de données et du plan de contrôle en donnant plus de responsabilité aux commutateurs. Pour le passage à l'échelle du SDN, l'*overhead* du plan de données et du plan de contrôle peut être réduit en utilisant les fonctionnalités CPU, d'entrée/sortie RAM. [118] accélération matériels les auteurs présentent une évaluation des performances du système à l'aide d'un outil de *benchmark* appelé « sysbench ». Dans [114] les auteurs proposent quelques principes de base que les plans de contrôle et de données doivent respecter, pour la « scalabilité » du SDN. Ils adoptent deux approches de référence selon les principes, vues à partir des messages de contrôle dans les SDN basés sur OpenFlow.

Les chercheurs énoncent quelques principes de base que les plans de contrôle et de données doivent respecter, basés sur la tendance de la *scalabilité* du SDN. De plus, ils adoptent deux approches de référence selon les principes, vues à partir des messages de contrôle dans les SDN basés sur OpenFlow. Ils concluent que leur évaluations démontrent que les approches peuvent maintenir la généralité du plan de données et améliorer le passage à l'échelle (« scalabilité ») du plan de contrôle. Dans [119], les auteurs ont réalisé une étude comparative des contrôleurs ONOS et ODL sur le critère du débit, c'est-à-dire le nombre de réponses que chacun des contrôleurs est capable de traiter par seconde, ils concluent que le contrôleur ONOS offre de loin les meilleures performances (plus d'un million pour ONOS contre un peu plus de *200K* pour ODL). Pour essayer de comprendre les raisons de cette différences entre ces contrôleurs tous développés en JAVA, ils ont analysé l'utilisation de la CPU et la gestion de mémoire. Pour la consommation de la CPU, ils concluent que l'utilisation du CPU par l'ONOS est inférieure à 10 % pour un commutateur, et ne dépasse jamais 80%, même pour 32 commutateurs. Inversement, l'utilisation de CPU par ODL est d'environ 35% pour un commutateur, et dépasse 96% pour 16 commutateurs ou plus. Sur la gestion de mémoire ils n'ont observé aucune fuite sur le contrôleur ODL, ils concluent alors que la faible performance d'ODL pourrait être causée par un « bug » sur le « plugin » OpenFlow qu'il a implémenté.

6.3 Passage à l'échelle

Le passage à l'échelle du SDN est une préoccupation importante pour de nombreux opérateurs de réseaux. Les principales particularités du SDN lorsqu'il est appliqué à un réseau de FAI sont la grande extension géographique et la nécessité de la transmission en bande du trafic de contrôle. Par conséquent, le trafic de contrôle échangé entre le contrôleur SDN et les nœuds du réseau doit être

soigneusement évalué pour la conception et le dimensionnement du réseau.

L'adoption du SDN dans les réseaux traditionnels des FAI est un défi en termes de performance et de fiabilité. La difficulté est exacerbée par le fait que les réseaux des FAI sont souvent étendus et géographiquement dispersés sur plusieurs sites imposant par conséquent un plan de contrôle à grande échelle pour satisfaire les demandes pour le trafic de contrôle et le trafic de données partageant souvent les mêmes ressources réseaux.

— Réduire l'interaction entre le plan de contrôle et le plan de donnée
— Améliorer les capacités de traitement des contrôleurs
— Accélération matériels au niveau du plan de données et du plan contrôle

6.4 Trafic de contrôle

Dans le contexte du SDN, le contrôleur établie indépendamment les instructions servant le traitement des paquets et les transmet aux commutateurs. C'est-à-dire pour chaque flux, il établie une règle de traitement. Ainsi, pour l'élaboration des instructions, appelées aussi les règles, que le contrôleur (i.e. le plan de contrôle) envoie aux commutateurs (i.e. le plan de données), deux approches sont envisageables : l'approche réactive et l'approche pro-active. On peut également parler d'une troisième approche qui résulte d'une combinaison de ces deux approches précédemment citées. Dans cette section, nous présenterons ceux deux notions avec les avantages et les limites de chacune d'elles. Nous présenterons une étude comparative des contrôleurs ONOS et ODL basée sur le trafic de contrôle et enfin l'impact des choix d'implementation sur les performances de chacun des contrôleurs.

6.4.1 Approche réactive

Avec cette méthode, la décision d'acheminement est prise flux par flux. Chaque fois qu'un nouveau flux (c'est-à-dire un flux pour laquelle aucune règle de transfert n'est disponible) arrive à un commutateur, ce dernier interagit avec le contrôleur pour installer les règles nécessaires au traitement de tous les paquets de ce flux. Le principal avantage de cette approche est qu'elle permet la mise en œuvre en ligne de politiques de routage flexibles, sans aucune connaissance préalable du trafic. En outre, des entrées (règles de transferts) de flux sont ajoutées uniquement en cas de besoin, ce qui réduit les besoins en mémoire dans les commutateurs. Cependant, pour le paquet initial de tout flux, un délai supplémentaire est requis, la charge du plan de contrôle et le nombre de messages de contrôle sont également plus élevés avec ce modèle. La forte interaction entre le plan de contrôle et le plan de données pourrait facilement épuiser le canal de contrôle/données avec des applications complexes ou de grands réseaux, et même être utilisé de manière abusive pour lancer une attaque par déni de service (DoS). Même si d'un autre côté, cette méthode réduit le risque d'altération par une entité tierce des instructions données aux commutateurs.

6.4.2 Approche pro-active

Avec cette approche, le contrôleur anticipe et pré-installe des règles au niveau des commutateurs avant même l'arrivée d'une quelconque demande de trafic. Cette méthode offre l'avantage de réduire le

nombre de fois où le plan de contrôle est sollicité suite à la présence d'un nouveau flux. Cependant, elle peut impacter négativement les performances du plan de données (les commutateurs) si le nombre de règles pré-installées devient trop important en l'absence d'un mécanisme optimisé de parcours de ces règles. En effet, cela augmente le nombre de règles à installer par défaut donc des ressources à réserver par conséquent et expose également le plan à un risque d'altération par une entité tierce malveillante des instructions données plan de données.

6.4.3 Approche hybride

Chacune de deux méthodes présentées précédemment offre des avantages mais également des inconvénients. C'est pourquoi dans la pratique, ces deux méthodes sont très souvent associées pour donner naissance une troisième approche appelée *l'approche hybride*.

6.5 Plate-forme expérimentale

Dans cette section nous présentons les différentes plate-formes d'expérimentation utilisées dans ce chapitre.

6.5.1 Débit du contrôleur

Pour mesurer le débit d'un contrôleur ainsi que son temps de traitement des requêtes, nous avons utilisé *Cbench* qui s'est imposé comme le principal outil de mesure de performances des contrôleurs SDN. *Cbench* fonctionne essentiellement sur deux modes : « throughput mode » pour le débit et « latency mode » pour la latence.

En mode débit (« throughput mode »), il envoie autant de paquets que possible pour calculer le nombre maximum de paquets traités par le contrôleur. En mode latence (« latency mode »), *Cbench* envoie un paquet et attend la réponse pour calculer le temps de traitement d'un seul paquet par le contrôleur. Nous avons présenté précédemment dans la section 4.6 une description de la plate-forme d'expérimentation utilisée pour réaliser ce test.

La Figure 6.3 présente l'architecture de la plate-forme utilisée pour étudier les performances de contrôleurs en terme de débit. Le but cette expérience est de mesurer le nombre de requêtes que les contrôleurs testés sont capables de traitées par seconde. Nous avons étudié les versions *1.12* d'ONOS et *Carbon* d'ODL. Pour cette série d'expériences, nous avons installé le contrôleur SDN sur le premier serveur, le second serveur est utilisé pour exécuter l'application *Cbench* qui génère des requêtes OpenFlow à destination du contrôleur. Le débit du contrôleur est déterminé par le nombre des requêtes traitées par seconde.

6.5.2 Expérimentation pour la passage à l'échelle

Les expériences ont été réalisée dans un environnement virtualisé avec *Virtualbox* sur un seul serveur physique avec les caractéristiques suivantes : *DELL PowerEdge R540* connectés au réseau local du *LIP6* et équipé d'un processeur Intel(R) Xeon(R) Silver 4108 CPU @ 1.80GHz (32 coeurs),32 128 Go DDR, deux interfaces réseaux *1 Gbps* sous Ubuntu 18.04 OS. Chaque machine virtuelle (VM) est

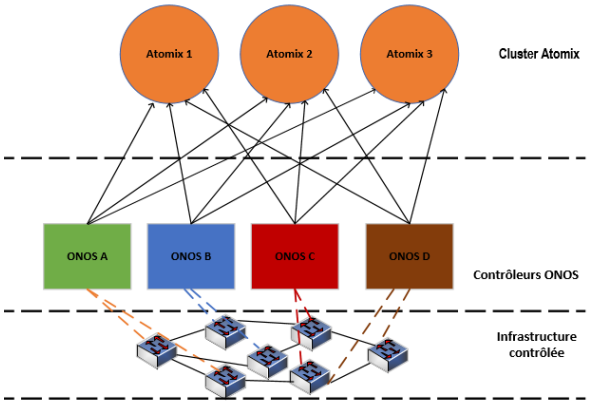


FIGURE 6.1 – Plate-forme expérimentale pour d'un cluster ONOS

configurée comme suit : *6G RAM, 4 vCPU, 2 vNIC et 80 GB vHDD avec Ubuntu 18.10* comme système d'exploitation (OS). Le nombre important des machines nécessaires à la mise en oeuvre de la nouvelle architecture du cluster d'ONOS explique notre choix de changer de plate-formes d'expérimentation par rapport au chapitre précèdent.

Pour configurer le cluster ONOS, nous avons utilisé 7 VMs. Trois VMs sont utilisées pour la mise en place cluster ONOS avec atomix, 3 VMs pour héberger chacune une instance du contrôleur ONOS et une VM pour Mininet. Depuis la parution de sa version *1.14*, ONOS propose une nouvelle architecture d'implémentation du cluster : la gestion du cluster, la découverte des services et le stockage des données persistantes sont désormais physiquement séparés du contrôleur ONOS. La gestion du cluster est assurée par le biais de la framework *Atomix* [120].

Atomix est un framework type « event-driven » (piloté par événement) permettant de coordonner des systèmes distribués tolérants aux pannes à l'aide d'une variété de protocoles de systèmes distribués éprouvés. Il fournit les éléments de base qui résolvent de nombreux problèmes courants des systèmes distribués, notamment la gestion des clusters, la messagerie asynchrone, l'adhésion à un groupe, l'élection d'un leader, le contrôle des concurrents distribués, le partitionnement et la réplication.

La figure 6.1 présente la nouvelle architecture d'un cluster ONOS. Nous avons présenté dans l'annexe un exemple de configuration d'une telle plate-forme utilisée dans le cadre de nos expériences. Cette architecture est divisée en trois grandes couches : une couche pour la gestion des cluster (atomix), une couche pour les contrôleurs ONOS et une couche pour l'infrastructure du réseau.

- **Cluster atomix :** Le *cluster atomix* est constitué de l'ensemble de serveurs sur lesquels est exécuté le programme *atomix*. Chaque serveur est identifié par un nom et une adresse IP. L'ensemble de serveurs d'un même *cluster* échangent sur un port TCP dédié. Nous avons présentons dans l'annexe A.3 un exemple de configuration d'un *cluster atomix*.
- **Contrôleur ONOS :** Cette couche est constitué de l'ensemble des instances actives ONOS. Les instances sont activités indépendamment les unes des autres en d'autres termes les instances lors de leur mise en service n'ont pas à savoir l'existence ou pas d'autres contrôleurs dans le

Topologie	# NB commutateurs	Liens unidirectionnels	Ports	# Hôtes
Topo.1	$N = 15$	28	44	16
Topo.2	$N = 31$	60	92	32
Topo.3	$N = 63$	124	188	64
Topo.4	$N = 127$	252	380	128
GEANT	$N = 41$	61	65	#4
RENATER	$N = 42$	56	116	#4

TABLE 6.1 – Paramètres clés des topologies utilisées

même réseau. Les seules informations à préciser lors de la configuration d'une instance devant se joindre à un cluster concerne les informations de celui-ci à savoir les identifiants et adresses IP des noeuds, le numéros de port TCP à utiliser.

- **Infrastructure réseaux :** Cette couche est constituée de d'ensembles des dispositifs(commutateurs et terminaux) de la couche infrastructure.

6.5.2.1 Description des expériences

Ainsi, dans le détails, la première topologie se compose de commutateurs *15* et d'hôtes *16*, la seconde dispose quant à elle de *31* commutateurs et de *32* hôtes , la troisième topologie se constituée de *63* et de *64* hôtes et la dernière comprend *127* commutateurs et *128* hôtes.

6.5.2.2 Méthodologie

Le « clustering » (environnement avec plusieurs contrôleurs) vise entre autre à adresser la problématique du passage à l'architecture traditionnelle (un seul contrôleur par réseau) du SDN.

L'augmentation du nombre des équipements (commutateurs ou machines terminales) entraînent inéluctablement une augmentation de la demande de trafic donc de charge du contrôleur sans aucune service utilisateurs. En effet, cela provoque une augmentation du trafic de contrôle, nous avons présenté cela dans le chapitre précèdent.

La méthodologie que nous avons utilisée pour analyser l'impact de l'augmentation de la demande de trafic consiste à 6 topologies différentes avec des caractéristiques différentes dont deux obtenues à partir des réseaux réalistes(*GEANT* et *RENATER*). Le tableau 6.1 présente ces topologies que nous avons utilisées dans le cadre de ces expériences ainsi que les paramètres clés de chacune d'entre elles à savoir le nombre de commutateurs, le nombre de liens unidirectionnels, le nombre total de ports actifs et le nombre de total de machines (terminaux) hôtes actives.

6.5.2.3 Scénarios de tests

Pour chaque expérience réalisée avons procédé comme suit :

- Nous lançons la topologie sur la *VM Mininet* puis nous attendons que la connexion entre les commutateurs et les contrôleurs se stabilise complètement. Nous assurons de la connectivité

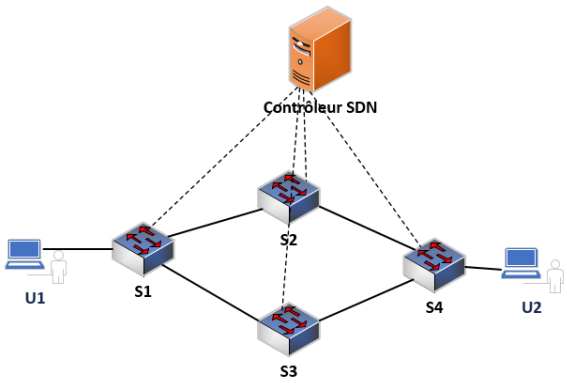


FIGURE 6.2 – Plate-forme d'expérimentation pour l'analyse du trafic de contrôle

entre les machines terminales par la commande entre les deux hôtes les plus éloignés de la topologie concernée.
- Nous accédons aux hôtes les plus éloignés par la commande *xterm* de *Mininet* présentée en annexe de ce manuscrit.
- Sur la première machine, nous exécutons le serveur *Iperf* en mode UDP puis sur l'autre machine nous laçons le client UDP en spécifiant les paramètres du serveur. Nous avons présenté dans l'annexe les commandes utilisées.
- Nous enregistrons dans le résultat des mésures d'*IPerf* dans un fichier que nous analyserons avec l'outil *gnuplot* [121].

6.5.3 Trafic de contrôle

Le trafic de contrôle dans le SDN peut être divisé en deux catégories (1) le trafic de contrôle engendré par la découverte et le maintien de la topologie, (2) et le trafic de contrôle engendré par l'installation des nouvelles règles suite à l'arrivée d'un nouveau flux. Nous réaliserons également une étude comparative des contrôleurs ONOS et ODL en nous basant sur la quantité de messages contrôle échangés suite à l'arrivée d'un nouveau flux au niveau d'un commutateur. Nous choisirons de baser notre analyse sur l'approche réactive.

6.5.3.1 Méthode d'évaluation

Le protocole OpenFlow est en constante évolution, dans sa version *1.0.0* il définissait 22 types de message contre 36 dans la version *1.5.0*. L'implémentation de certains de ces messages est optionnelle par les dispositifs supportant le protocole OpenFlow et peut être différente d'un contrôleur à un autre.

Pour évaluer le nombre de messages de contrôle échangés entre le plan de contrôle et le plan de données, nous avons utilisé deux méthodes. Nous avons premièrement utilisé un modèle mathématique basé sur une évaluation expérimentale et une analyse du code source du contrôleur ONOS proposé dans [114], qui permet de déterminer le nombre de messages de contrôle générés pour le transfert dans le cadre d'une approche réactive d'une source à destination dans le réseau.

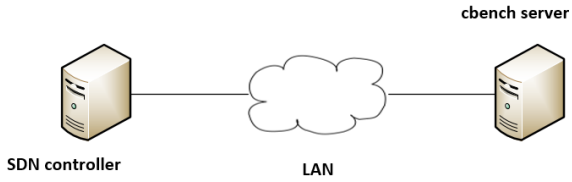


FIGURE 6.3 – Plate-forme expérimentale pour Cbench

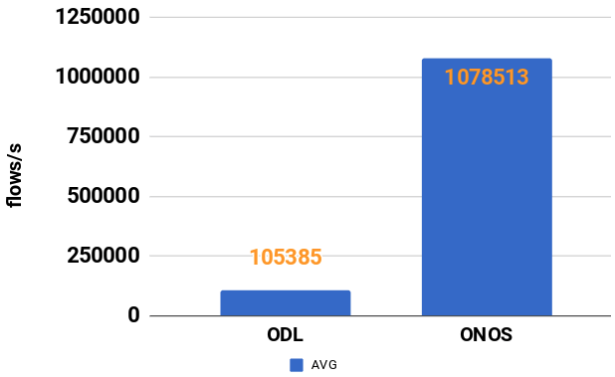


FIGURE 6.4 – Test Cbench en « throughput mode » avec 16 commutateurs

La seconde méthode que nous avons utilisée consiste à analyser expérimentalement le nombre de messages de contrôle générés par le trafic utilisateur.

6.5.3.2 Description des expériences

Les expériences ont été réalisées dans un environnement avec un seul contrôleur. La Figure 6.2 présente la plate-forme utilisée. Elle est constituée d'une topologie avec 4 commutateurs ($S1$, $S2$, $S3$ et $S4$) inter-connectés entre eux tels qu'on l'observe sur la Figure 6.2. A chaque commutateur est connecté une machine hôte. Pour chaque expérience, nous lançons les contrôleurs ONOS et OpenDay-Light respectivement sur le serveur dédié puis la topologie avec *Mininet* sur l'autre serveur. Nous avons présenté précédemment les caractéristiques de ces serveurs. Ensuite, nous exécutons la commande *ping* entre les machines hôtes $U1$ et $U2$ puis enregistrons le trafic dans un fichier que nous analyserons par la suite avec l'outil *Wireshark*.

6.6 Analyse des résultats

Dans cette section, nous présentons les résultats obtenus suite sur l'évaluation des performances des contrôleurs ONOS et ODL sur le débit, le trafic de contrôle et le passage à l'échelle.

6.6.1 Analyse du débit

La Figure 6.4 présente les résultats des études de performances des contrôleurs ODL et ONOS en termes de débit mesuré en *(réponses/s)*. Nous avons décrit précédemment les expériences réalisés. Cette figure montre également que le contrôleur ONOS est capable de traiter plus d'un million de requêtes par seconde alors que le contrôleur ODL ne peut en traiter que seulement un peu plus de cent milles requêtes par seconde.

Ces résultats nous envoient une première observation dûe à la différence de performances considérables entre ces deux contrôleurs qui disposent par ailleurs de plusieurs caractéristiques communes. Au nombre de caractéristiques communes entre ces deux contrôleurs figurent, un même langage de programmation (*JAVA*), le multi-threading qui ont un impact réel sur leurs performances du contrôleur [119, 96].

Nos résultats confirment par ailleurs les observations des auteurs dans [99] qui concluent que la meilleure performance offerte par ONOS est dû à l'utilisation d'une collection de « hashes » pour la gestion des adresses Media Access Control des périphériques SDN.

Un autre résultat que l'on peut également observer dans les Figures 5.3 et 5.4 est que le réseau contrôlé par ONOS offre un débit maximum de données utilisateur jusqu'à 30 *Gbps* alors que le débit maximum de données utilisateur dans le réseau contrôlé par ODL ne peut dépasser 512 *Mbps*. Ce résultat n'est pas lié au temps de réaction suite à un changements dans la topologie mais il peut nous aider à confirmer les résultats de l'analyse du débit de requêtes présentée dans la figure 6.4. Cela signifie que le contrôleur ONOS est mieux approprié dans d'une application générant un volume important de flux.

Cependant, d'autres études dans la littérature comme dans [119], concluent que la faiblesse du débit offert par ODL par rapport à ONOS est dû à un disfonctionnement (« bug ») du « plugin » OpenFlow implémenté par celui-ci.

6.6.2 Trafic de contrôle

Le Tableau 6.2 présente les résultats obtenus en combinant les deux méthodes. Chaque ligne du tableau présente respectivement le nombre totale de messages échangés avec ONOS et ODL suite à l'arrivée d'un nouveau flux c'est à dire un flux qui ne correspond à aucune règle déjà installé au niveau des commutateurs. La deuxième colonne *Limite inférieure* présente le nombre minimal de messages observés tandis que la dernière colonne *Limite supérieure* présente le nombre maximal de messages observés.

TABLE 6.2 – Nombre total de messages de contrôle échangés suite à l'arrivée d'un nouveau flux

	Limite inférieure	Limite supérieure
ODL	$2 * (P_{IN-NF} + P_{OUT-NF})$	$(H + 2)$ * P_{IN-NF} + 2 * P_{OUT-NF}
ONOS	$17P_{SD}$	$2L + N + 15P_{SD}$

N détermine le nombre de commutateurs dans le réseau, P_{IN-NF} et P_{OUT-NF} correspondent respectivement aux nombres de messages de type *Packet-IN* et de type *Packet-OUT* générés suite

à l'arrivée d'un nouveau flux dont aucune des règles actuellement installées sur le commutateur ne permet de satisfaire. N_{SD} détermine le nombre de commutateurs traversés sur le plus court chemin entre la source S du flux et sa destination D, L représente le nombre de liens unidirectionnels entre les commutateurs. P_{fm} détermine le nombre de message de type *FlowMod* échangés, il permet au contrôleur de modifier l'état d'un commutateur *OpenFlow*. P_{bq} et P_{br} determinent respectivement le nombre de messages de types *BarrierReq* et *BarrierRes* échangés. Le message *BarrierReq* est utilisée par le contrôleur pour définir un point de synchronisation, en s'assurant que tous les messages d'état précédents sont exécutés, le message *BarrierReq* est renvoyée en réponse à celui-ci au contrôleur pour confirmer la bonne synchronisation.

Le résultat de nos expériences montre que le contrôleur ODL offre des meilleures performances sur le nombre de messages contrôles générés suite à l'arrivée d'un nouveau flux par rapport à ONOS. Cela s'explique par le fait que le contrôleur ODL pré-installe des règles par défaut sur chaque commutateur permettant d'acheminer du trafic vers tous les voisins directs alors qu'ONOS n'installent ces règles que seulement à l'arrivée d'un nouveau flux.

6.6.3 Passage à l'échelle

La figure 6.5 montre l'évolution de la bande passante utilisateurs en fonction de l'augmentation de la demande de trafic c'est à dire une augmentation du nombres des équipements (commutateurs et machines terminales). La courbe en violet présente la mesure la bande passante pendant 120 secondes pour la topologies *Topo.1* (15 commutateurs et 16 machines terminales), le débit moyenne observé est de 1.5 *Gbits/s* tandis que la courbe en jaune présente la bande passante utilisateur moyenne pour la topologie *Topo.4* (127 commutateurs et 128 hôtes) mesurée autour de *1 Gbit/s*. Entre ces deux, nous observons le résultat de deux autres expériences avec respectivement (31 commutateurs, 32 hôtes) et (63 commutateurs et 64 hôtes). Les résultats présentés dans la figure 6.6 correspondent aux testes utilisant des topologies qui émulent des réseaux réels *GEANT* et *RENATER* avec 39 et 42 commutateurs respectivement. Sur la figure 6.6, on peut clairement observer que les courbes de trafic pour les topologies *GEANT* et *RENATER* respectivement en bleu et en violet sont quasiment identiques avec un débit moyenne autour de 1.38 *Gbit/s* alors que pour la topologie en arbre avec 63 commutateurs la moyenne du débit tombe à 1.2 *Gbit/s* environ. Ces résultats confirment les tendances observées dans les résultats présentés dans la figure 6.5.

6.7 Conclusion

Dans ce chapitre, nous avons présenté les résultats sur l'évaluation des performances du contrôleur ONOS par rapport à l'augmentation de la demande de trafic utilisateurs dans un environnement avec plusieurs instances déployées en « cluster ». Ces résultats obtenus expérimentalement nous permettent de conclure que est particulièrement adapté pour des applications avec une demande de trafic dynamique c'est à dire pouvant évoluer pouvant d'un moment à un autre dans une journée par exemple. En effet, l'adjonction d'une nouvelle instance du contrôleur dans un « cluster » en état fonctionnement est complètement transparent pour les membres qui composent alors qu' avec des solutions comparables telles ODL, une telle action nécessite obligatoirement le redémarrage de toutes les instances en cours d'exécution pour être prise en compte. En outre, l'augmentation soudaine du trafic utilisateurs ne

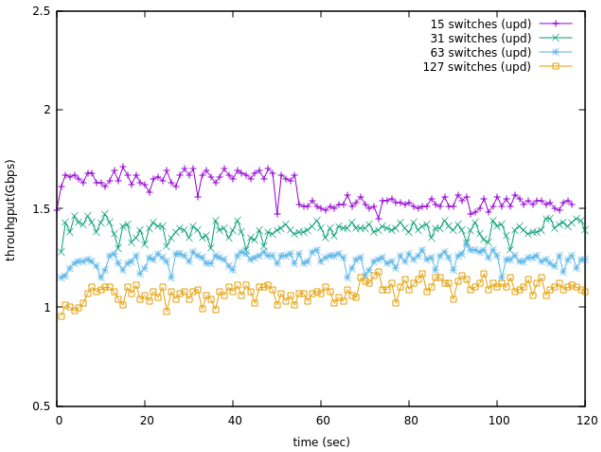


FIGURE 6.5 – Évolution du débit (UDP) en fonction du nombre de noeuds avec ONOS

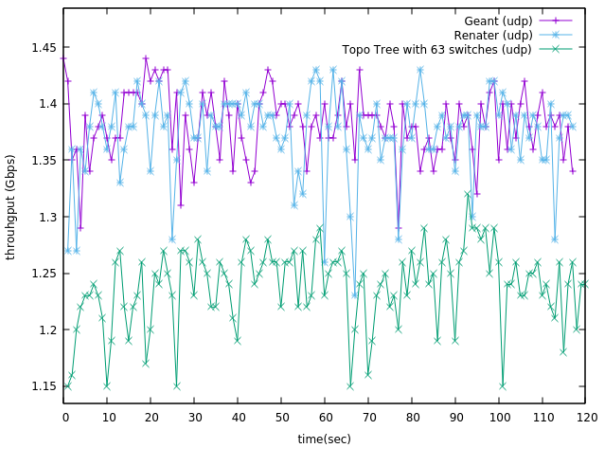


FIGURE 6.6 – Comparaison des débits entre trois topologies : *GEANT*, RENATER et 63 commutateurs en mode cluster avec ONOS

provoque un effondrement important des performances du contrôleurs de la topologie avec 15 commutateurs et 16 machines hôtes à une topologie avec 127 commutateurs, 128 machines terminales soit une augmentation du nombre total des équipements présents dans le réseau de plus 120%, les performances n'ont diminuées que de 66%.

Nous avons également dans ce chapitre évalué expérimentalement et avec un modèle mathématique l'impact du nombre de messages contrôle sur les performances des mêmes contrôleurs, les résultats montrent que le contrôleur ODL offre les meilleures performances en utilisant une approche pro-active dans l'installation de certaines nécessaires à l'acheminement des données utilisateurs.

Chapitre 7

Conclusion

Sommaire

7.1 Perspectives 114
7.2 Publications 115
7.2.1 Conférences 115
7.2.2 Rapports techniques 115

Le SDN a le potentiel pour surmonter les inconvénients des réseaux actuels déployés par la plupart des entreprises. Malgré ce potentiel, le déploiement reste encore très faible. Cette situation est due à plusieurs facteurs dont le plus important est le coût de déploiement initial. En effet, il est nécessaire de mettre à niveau ou de remplacer tous les équipements actuels. Ainsi, pour profiter à court terme des avantages du SDN, il faut réfléchir à des solutions pour assurer la transition des réseaux actuels vers le SDN. Les réseaux hybrides répondent à cette problématique. On parle aussi des réseaux transitionnels. Dans les sections suivantes nous présentons les différentes approches et solutions introduites dans la la littérature au moment de la rédaction de ce document pour adresser cette problématique.

La première contribution de cette thèse consistait à proposer une architecture pour l'intégration incrémentale du SDN au coté des réseaux actuels appelés *legacy*. L'objectif étant de profiter des avantages de celui-ci tout en réduisant les coûts CAPEX/OPEX qui font partir des obstacles à la transition vers des réseaux totalement SDN. Dans cette contribution nous avons démontré les avantages que nous pouvons tirer en introduisant un contrôleur centralisé dans un réseau distribué (*legacy*) c'est à dire avec un plan de contrôle partiellement déplacé au niveau du contrôleur. Le cas d'utilisation que nous avons expérimenté portait sur la problématique du temps de convergence dans un réseau distribué tels que TRILL suite à la défaillance d'un lien. Nous avons pu démontrer que ce temps peut être réduit à 5 secondes environ (contre 30 secondes minimum dans un réseau distribué classique) le lorsqu'un contrôleur disposant d'une vue globale sur le réseau se substitue aux routeurs pour recalculer (on peut également pré-calculer une route alternative) ou la nouvelle topologie du réseau suite à une panne. Cette

architecture qui permet tirer profit des avantages du *legacy* et SDN deux paradigmes différents s'inscrit dans la logique des réseaux hybrides indispensables pour le transition vers la nouvelle génération des réseaux.

Dans la deuxième contribution, nos travaux ont consisté dans un premier temps à faire l'inventaire des contrôleurs SDN proposés dans le monde open-source dans le but d'identifier lequel est le mieux adapté répondre aux besoins des réseaux hybrides introduits dans la contribution précédente. C'est ainsi que les contrôleurs ONOS et ODL par la diversité et le nombre des fonctions offertes se sont imposés comme les principaux candidats. Nous avons par la suite concentré nos efforts à l'évaluation de leurs performances sur de notre point de vue la fonction la plus critique de l'architecture du SDN : La gestion de la topologie. La gestion de la topologie peut se diviser en deux parties très lié à savoir la découverte de la topologie qui peut intervenir par exemple lors du premier lancement du réseau et le changement de la topologie qui intervient lors de l'arrivée ou du départ d'un équipement par exemple. Ainsi, nous avons évalué expérimentalement ces contrôleurs dans un environnement mono-contrôleur puis avec des contrôleurs multiples. Les résultats obtenus ont montré que le contrôleur ODL offre les meilleurs performances sur la découverte de la topologie dans les deux cas de figures tandis qu' ONOS réagit mieux lors d'un changement de la topologie.

Dans la troisième contribution qui s'inscrit dans la suite logique de la précédente nos avons analysé expérimentalement le débit (nombre des requêtes traitées par seconde), le trafic de contrôle de deux contrôleurs ainsi que le passage à l'échelle du contrôleurs ONOS en évaluant leurs capacités à répondre à une augmentation de la demande de trafic utilisateur. ONOS et ODL sont deux contrôleurs possédant une architecture physiquement distribuée qui permettent ainsi d'adresser au moins à deux des problématiques du SDN traditionnel à savoir la haute disponibilité et le passage à l'échelle. Les résultats obtenus montre que le contrôleur ONOS offre les meilleures performances.

7.1 Perspectives

Dans un future proche, il serait pertinent d'explorer la problématique de l'interopérabilité entre contrôleurs SDN hétérogènes. Actuellement, aucun protocole n'est normalisé pour communication entre deux contrôleurs hétérogènes tels qu'ONOS et ODL, rendant ainsi impossible l'échange de trafic entre deux systèmes autonomes voisins intégralement contrôlé par du SDN sans recourir à des mécanismes de traduction au niveau des frontières. L'autre piste à explorer serait également sur le nombre optimal d' instances d'un contrôleur à déployer dans le cadre d'un *cluster*. En effet, dans la littérature des modèles mathématiques qui n'intègrent pas certains paramètres (bande passante consommée par exemple), de synchronisation entre contrôleurs permettent d'estimer ce nombre. Cependant, des expérimentations réalisées dans le cadre de cette thèse ont montré que le délai de synchronisation entre instances qui croit en fonction du nombre de noeuds ne permet que quelques noeuds seulement. Dans nos expériences avec ONOS et ODL, à partir de 4 instances le système devient très instable avec des noeuds qui ne répondent plus alors même qu'ils sont sous tension.

7.2 Publications

7.2.1 Conférences

Publications dans les conférences internationales :

— **Mamadou Tahirou Bah and Abdelhadi Azzouni and Thi Mai Trang Nguyen and Guy Pujolle**
Topology Discovery Performance Evaluation of OpenDaylight and ONOS controllers, 2019 22nd Conference on Innovation in Clouds, Internet and Networks and Paris, France (IEEE) (ICIN 2019)

— **S. Secci, A. Diamanti, JM. Sanchez, MT. Bah, P. Vizzarreta, C. Mas Machuca, S. Scott-Hayward, D. Smith**, *Security and Performance Comparison of ONOS and ODL Controllers*, Informational Report, Open Networking Foundation (ONF), (Sept. 2019).

— **Mamadou Tahirou Bah, V. Del Piccolo, M. Bourguiba, K. Haddadou :** *A centralized controller to improve fault tolerance in TRILL-based fabric networks*, Smart Cloud Networks and Systems (SCNS), Dubai, United Arab Emirates, (IEEE) (2016)

7.2.2 Rapports techniques

— Architecture CARP V1 : Définit la première de l'architecture CARP qui comprenant logiciels, protocoles retenus et premières spécifications des interfaces

— Architecture CARP V2 : Cette version intègre les interfaces consolidées et une première version du langage de description de services réseaux

— Architecture finale CARP : Version de référence validée de l'architecture pour CARP

Annexe A

Annexes

A.1 Contribution à l'IETF (Internet Engineering Task Force)

L'IETF organise depuis quelques années des « Hackathon » (challenges) pour encourager les développeurs à discuter, collaborer et développer des utilitaires, des idées, des exemples de code et des solutions qui illustrent les implémentations pratiques de ses normes. Les meilleurs projets sont primés et présentés lors d'une session spéciale et lors des travaux des groupes thématiques concernés. Nous suivons les activités du groupe de travail I2RS (Interface to Routing System) [23, 122, 33, 123] dont l'objectif est de proposer une interface sud SDN qui tienne en compte des spécificités des réseaux traditionnels. C'est un protocole pour réseau hybride.

Lors de la session IETF 95, nous [2] avons proposé la première implémentation de I2RS (Interface to Routing System) avec des outils open-sources (Quagga, mininet/mininext). Les résultat ont fait l'objet d'une présentation pendant la session du groupe de travail I2RS.

Lors de la session 96, nous [3] avons poursuivi le projet entamé lors de la session précédente, en proposant une version améliorée. Ce projet a obtenu le troisième prix sur une vingtaine des propositions, les résultats ont fait l'objet d'une présentation lors de la session du groupe de travail I2RS. Les résultats de ces contributions sont accessibles en ligne respectivement sur les références [124] et [125]. Les équipes de travail sont indiquées respectivement par les notes (3) et (4) en bas de cette page.

A.2 Commandes Mininet

— *xterm node* : Pour accéder au terminal de la machine *node*
— *link end1 end2 [up down]* : Pour activer /désactiver un lien

2. Team I2RS ietf 95 : Don Fedyk, Susan Hares (leader), Jason Sterne, Lucy Yong, **Mamadou T. Bah**
3. Team I2RS ietf 96 : Don Fedyk, Thomas Scheffler, Edwin Cordiero, **Mamadou Bah**, Susan Hares (leader)

A.3 Cluster ONOS

Le but de cette annexe est de présenter les opérations permettant la configuration d'un cluster ONOS utilisé dans le cadre des nos expériences. La configuration d'un cluster ONOS se fait principalement en deux étapes. Premièrement il faut configurer un cluster avec *atomix* puis configurer un à un les contrôleurs (instance) qui doivent rejoindre ce cluster. Dans cette annexe, nous présentons le contenu du principal fichier de configuration d'un cluster *atomix* (*atomix.conf*) avant de présenter un exemple de configuration (*cluster.json* d'une instance d'ONOS joignant un cluster.

A.3.1 Configuration d'un cluster atomix *(atomix.conf)*

```
"cluster.discovery" {
    "type": "bootstrap"
    "nodes.1" {
      "id": "atomix-1"
      "address": "132.227.85.248:5679"
    }
    "nodes.2" {
      "id": "atomix-2"
      "address": "132.227.85.234:5679"
    }
    "nodes.3" {
      "id": "atomix-3"
      "address": "132.227.85.237:5679"
    }
  }

management-group {
  type: raft
  partitions: 1
  storage.level: disk
  members: [atomix-1, atomix-2, atomix-3]
}

partition-groups.raft {
  type: raft
  partitions: 4
  partitionSize: 4
  storage.level: disk
  members: [atomix-1, atomix-2, atomix-3]
}
```

A.3.2 Ajouter une instance ONOS à un cluster

1. *wget http ://repo1.maven.org/maven2/org/onosproject/onos-releases/1.14.0/onos-1.14.0.tar.gz*
2. *tar -xzvf onos-1.14.0.tar.gz*
3. *cd onos-1.14.0*
4. *cd onos-1.14.0*
5. *mkdir config ; cd config*
6. *vi cluster.json*
   Le fichier *cluster.json* n'existe pas par défaut, il faudra le configurer obligatoirement dans le repertoire *config* crée précdement.

```
{
  "name": "onos",
  "node": {
    "id": "onos-1",
    "ip": "192.168.56.101",
    "port": 9876
  },
  "storage": [
    {
      "id": "atomix-1",
      "ip": "132.227.85.248",
      "port": 5679
    },
    {
      "id": "atomix-2",
      "ip": "132.227.85.234",
      "port": 5679
    },
    {
      "id": "atomix-3",
      "ip": "132.227.85.237",
      "port": 5679
    }
  ]
}
```

Ce fichier contient principalement les compartiments suivant :

— Dans la section *name*, défnit le nom du *cluster* (pour ce cas on l'a nommé *onos*

— Dans la section *node*, on définit les informations du contrôleurs (*id, l'adresse IP, et le numéro de port*)dans le cadre que nous présentons ici *(onos-1, 192.168.56.101, 9876)*. Le port TCP *9876* est utilisé par les instances ONOS pour communiquer entre elles.

— Dans la section *Storage*, on définit les partitions auxquelles sont rattachées le contrôleur en précisant pour chacune *le nom de la partition, son adresse IP et le numéro de port TCP utilisé pour communication entres partitions et entres les partitions et les instances ONOS*

A.4 Cluster OpenDayLight

Le but de cette partie est de présenter les opérations réalisées dans le cadre de nos expériences pour mettre en place un cluster ODL. Il y a plusieurs manières de configurer un cluster ODL, nous présentons ici la méthode utilisant le script *configure_cluster.sh* fourni avec l'archive d'ODL.

Sur chaque instance impliquée dans le cluster on suit les étapes suivantes :

1. *wget https ://nexus.opendaylight.org/content/repositories/public/org/opendaylight/integration/karaf/0.8.4/karaf-0.8.4.zip*
2. *unzip karaf-0.8.4.zip*
3. *cd configuration/initial/*
4. *./configure_cluster.sh 1 132.227.85.234 132.227.85.248 132.227.85.237*

A.4.1 Configuration du cluster

La configuration d'un cluster ODL implique principalement trois fichiers, le fichier *akka.conf*, modules.conf et modules-shard.conf.

A.4.1.1 akka.conf

```
odl-cluster-data {
  akka {
    remote {
      artery {
        enabled = off
        canonical.hostname = "132.227.85.234"
        canonical.port = 2550
      }
      netty.tcp {
        hostname = "132.227.85.234"
        port = 2550
      }
      # when under load we might trip a false positive on the failure detector
      # transport-failure-detector {
        # heartbeat-interval = 4 s
        # acceptable-heartbeat-pause = 16s
      # }
    }
```

```
    cluster {
      # Remove ".tcp" when using artery.
      seed-nodes = ["akka.tcp://opendaylight-cluster-data@132.227.85.234:2550"
"akka.tcp://opendaylight-cluster-data@132.227.85.248:2550",
"akka.tcp://opendaylight-cluster-data@132.227.85.237:2550"]

      roles = ["member-1"]

    }

    persistence {
      # By default the snapshots/journal directories live in KARAF_HOME. You c
      # modifying the following two properties. The directory location specifi
      # The relative path is always relative to KARAF_HOME.

      # snapshot-store.local.dir = "target/snapshots"
      # journal.leveldb.dir = "target/journal"

      journal {
        leveldb {
          # Set native = off to use a Java-only implementation of leveldb.
          # Note that the Java-only version is not currently considered by Akk

          # native = off
        }
      }
    }
  }
}
```

A.4.1.2 modules.conf

```
    modules = [

{
name = "inventory"
namespace = "urn:opendaylight:inventory"
shard-strategy = "module"
},
{
name = "topology"
```

```
namespace = "urn:TBD:params:xml:ns:yang:network-topology"
shard-strategy = "module"
},
{
name = "toaster"
namespace = "http://netconfcentral.org/ns/toaster"
shard-strategy = "module"
}
]
```

A.4.1.3 module-shard.conf

```
   module-shards = [
{
name = "default"
shards = [
{
name = "default"
replicas = ["member-1",
"member-2",
"member-3"]
}
]
},
{
name = "inventory"
shards = [
{
name="inventory"
replicas = ["member-1",
"member-2",
"member-3"]
}
]
},
{
name = "topology"
shards = [
{
name="topology"
replicas = ["member-1",
"member-2",
"member-3"]
```

```
}
]
},
{
name = "toaster"
shards = [
{
name="toaster"
replicas = ["member-1",
"member-2",
"member-3"]
}
]
}
]
```

A.5 Iperf

A.5.1 Coté serveur

- *iperf -u -p 5656 -i 1*
- *u* : pour UDP
- *p* : Pour spécifier le numéro de port UDP
- *i* :Périodicité en secondes de l'affichage bande passante

A.5.2 Côté client

- *iperf -u -c 10.0.0.1 -p 5656 -b 10G -t 120*
- *c* : Pour définir l'adresse IP du serveur Iperf
- *b* : Pour UDP, bande passante à envoyer en *bits/sec* (par défaut *1 Mbit/sec*, implique *-u*)
- *t* : Pour définir la durée de l'expérience
- Commandes pour analyser les résultats :
- *cat fic.txt |grep sec | head -120 | tr - " " | awk 'print* 3,8 > *Analyse.txt*

Bibliographie

[1] FGS NFV 002. https://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf. [Online].

[2] Open Network Foundation. https://www.opennetworking.org/. [Online].

[3] Openflow. https://www.opennetworking.org/software-defined-standards/specifications/. [Online].

[4] ONOS. https://onosproject.org/. [Online].

[5] Opendaylight. https://www.opendaylight.org/. [Online].

[6] John T. Moy. OSPF Version 2. RFC 2328, April 1998.

[7] Les Ginsberg and Mike Shand. Reclassification of RFC 1142 to Historic. RFC 7142, February 2014.

[8] Border Gateway Protocol 3 (BGP-3). RFC 1267, October 1991.

[9] Ahmed Amamou, Kamel Haddadou, and Guy Pujolle. A trill-based multi-tenant data center network. *Computer Networks*, 68 :35 – 53, 2014. Communications and Networking in the Cloud.

[10] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. B4 : Experience with a globally-deployed software defined wan. *SIGCOMM Comput. Commun. Rev.*, 43(4) :3–14, August 2013.

[11] Sdn-ip. https://wiki.onosproject.org/display/ONOS/SDN-IP+Architecture/. [Online].

[12] Mininet. http://mininet.org/. [Online].

[13] Donald E. Eastlake 3rd, Dinesh G. Dutt, Silvano Gai, Radia Perlman, and Anoop Ghanwani. Routing Bridges (RBridges) : Base Protocol Specification. RFC 6325, July 2011.

[14] M. T. Bah, V. Del-Piccolo, M. Bourguiba, and K. Haddadou. A centralized controller to improve fault tolerance in trill-based fabric networks. In *2016 3rd Smart Cloud Networks Systems (SCNS)*, pages 1–6, Dec 2016.

[15] M. T.BAH, A. Azzouni, M. T. Nguyen, and G. Pujolle. Topology discovery performance evaluation of opendaylight and onos controllers. In *2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, pages 285–291, Feb 2019.

[16] JM. Sanchez MT. Bah P. Vizzarreta C. Mas Machuca S. Scott-Hayward D. Smith S. Secci, A. Diamanti. Security and performance comparison of onos and odl controllers. In *Informational Report, Open Networking Foundation (ONF)*, Sept 2019.

[17] F. Wang, H. Wang, B. Lei, and W. Ma. A research on high-performance sdn controller. In *2014 International Conference on Cloud Computing and Big Data*, pages 168–174, Nov 2014.

[18] Y. Jarraya, T. Madi, and M. Debbabi. A survey and a layered taxonomy of software-defined networking. *IEEE Communications Surveys Tutorials*, 16(4) :1955–1980, Fourthquarter 2014.

[19] OVS. `https://www.openvswitch.org/`. [Online].

[20] Juniper Junos MX-series. `https://www.juniper.net/documentation/en_US/junos/information-products/topic-collections/release-notes/19.2/index.html/`. [Online].

[21] Rob Enns, Martin Bjorklund, Andy Bierman, and Jürgen Schönwälder. Network Configuration Protocol (NETCONF). RFC 6241, June 2011.

[22] Netconf. `http://www.netconfcentral.org/.NETCONF/`. [Online].

[23] Alia Atlas, David Ward, and Thomas Nadeau. Problem Statement for the Interface to the Routing System. RFC 7920, June 2016.

[24] P4. `https:https://p4.org/`. [Online].

[25] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4 : Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3) :87–95, July 2014.

[26] Martin Björklund. YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). RFC 6020, October 2010.

[27] Yang central. `http://www.yang-central.org/twiki/bin/view/Main/WebHome/`. [Online].

[28] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, pages 305–320, Berkeley, CA, USA, 2014. USENIX Association.

[29] Amin Tootoonchian and Yashar Ganjali. Hyperflow : A distributed control plane for openflow. In *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, INM/WREN'10, pages 3–3, Berkeley, CA, USA, 2010. USENIX Association.

[30] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, and Scott Shenker. Onix : A distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 351–364, Berkeley, CA, USA, 2010. USENIX Association.

[31] Soheil Hassas Yeganeh and Yashar Ganjali. Kandoo : A framework for efficient and scalable offloading of control applications. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 19–24, New York, NY, USA, 2012. ACM.

[32] K. Phemius, M. Bouet, and J. Leguay. Disco : Distributed multi-domain sdn controllers. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–4, May 2014.

[33] Thomas Nadeau, Alia Atlas, Joel M. Halpern, Susan Hares, and David Ward. An Architecture for the Interface to the Routing System. RFC 7921, June 2016.

[34] H. Yin, H. Xie, T. Tsou, D. Lopez, P. Aranda, and R. Sidi. SDNi : A Message Exchange Protocol for Software Defined Networks (SDNS) across Multiple Domains. Internet Draft, June 2012.

[35] J. S. Choi and X. Li. Hierarchical distributed topology discovery protocol for multi-domain sdn networks. *IEEE Communications Letters*, 21(4) :773–776, April 2017.

[36] B. P. R. Killi and S. V. Rao. Controller placement with planning for failures in software defined networks. In *2016 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, pages 1–6, Nov 2016.

[37] P. Xiao, W. Qu, H. Qi, Z. Li, and Y. Xu. The sdn controller placement problem for wan. In *2014 IEEE/CIC International Conference on Communications in China (ICCC)*, pages 220–224, Oct 2014.

[38] Y. Hu, W. Wendong, X. Gong, X. Que, and C. Shiduan. Reliability-aware controller placement for software-defined networks. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 672–675, May 2013.

[39] William J. Bolosky, Dexter Bradshaw, Randolph B. Haagens, Norbert P. Kusters, and Peng Li. Paxos replicated state machines as the basis of a high-performance data store. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 141–154, Berkeley, CA, USA, 2011. USENIX Association.

[40] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2) :133–169, May 1998.

[41] Network Functions Virtualisation (NFV) ; report accelaration technologie and use cases. *ETSI GS NFV-IFA 001 V1.1.1 (2015-12)*, 2015.

[42] R. B. Bohn, J. Messina, F. Liu, J. Tong, and J. Mao. Nist cloud computing reference architecture. In *2011 IEEE World Congress on Services*, pages 594–596, July 2011.

[43] GS NFV 001. `https://www.etsi.org/deliver/etsi_gs/NFV-PER/001_099/001/01.01.02_60/gs_NFV-PER001v010102p.pdf`. [Online].

[44] Opnfv. `https://www.opnfv.org/`. [Online].

[45] Maryam Tahhan, Billy O'Mahony, and Al Morton. Benchmarking Virtual Switches in the Open Platform for NFV (OPNFV). RFC 8204, September 2017.

[46] Github openmano. `https://github.com/nfvlabs/openmano/`. [Online].

[47] Openmano. `http://www.tid.es/long-term-innovation/network-innovation/telefonica-nfv-reference-lab/openmano`. [Online].

[48] Openstack. https://www.openstack.org/. [Online].

[49] Onap. https://www.onap.org/. [Online].

[50] Cloudnfv. http://www.cloudnfv.com/. [Online].

[51] A. Francescon, G. Baggio, R. Fedrizzi, R. Ferrusy, I. G. Ben Yahiaz, and R. Riggio. X–mano : Cross–domain management and orchestration of network services. In *2017 IEEE Conference on Network Softwarization (NetSoft)*, pages 1–5, July 2017.

[52] Larry Peterson, Scott Baker, Marc De Leenheer, Andy Bavier, Sapan Bhatia, Mike Wawrzoniak, Jude Nelson, and John Hartman. Xos : An extensible cloud operating system. In *Proceedings of the 2Nd International Workshop on Software-Defined Ecosystems*, BigSystem '15, pages 23–30, New York, NY, USA, 2015. ACM.

[53] J. Batalle, J. Ferrer Riera, E. Escalona, and J. A. Garcia-Espin. On the implementation of nfv over an openflow infrastructure : Routing function virtualization. In *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*, pages 1–6, Nov 2013.

[54] Anat Bremler-Barr, Yotam Harchol, David Hay, and Yaron Koral. Deep packet inspection as a service. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT '14, pages 271–282, New York, NY, USA, 2014. ACM.

[55] Lionel Bertaux, Samir Medjiah, Pascal Berthou, Slim Abdellatif, Akram Hakiri, Patrick Gelard, Fabrice Planchou, and Marc Bruyère. Software Defined Networking and Virtualization for Broadband Satellite Networks. *IEEE Communications Magazine*, 53(3) :pp. 54–60, March 2015.

[56] Sdn. https://www.opennetworking.org/wp-content/uploads/2013/02/conformance-test-spec-openflow-1.0.1.pdf/. [Online].

[57] Sdn-requirements. https://www.opennetworking.org/wp-content/uploads/2013/04/onf2014.227_Optical_Transport_Requirements.pdf/. [Online].

[58] Uses cases optical transport for sdn. https://www.opennetworking.org/wp-content/uploads/2013/04/optical-transport-use-cases.pdf/. [Online].

[59] Sdn migration[uses cases]. https://www.opennetworking.org/images/stories/downloads/sdn-resources/use-cases/Migration-WG-Use-Cases.pdf/. [Online].

[60] Rob Enns. NETCONF Configuration Protocol. RFC 4741, December 2006.

[61] Joel M. Halpern, Robert HAAS, Avri Doria, Ligang Dong, Weiming Wang, Hormuzd M. Khosravi, Jamal Hadi Salim, and Ram Gopal. Forwarding and Control Element Separation (ForCES) Protocol Specification. RFC 5810, March 2010.

[62] Etsi nfv. https://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf/. [Online].

[63] J. Medved, R. Varga, A. Tkacik, and K. Gray. Opendaylight : Towards a model-driven sdn controller architecture. In *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, pages 1–6, June 2014.

[64] S. Vissicchio, L. Vanbever, L. Cittadini, G. G. Xie, and O. Bonaventure. Safe update of hybrid sdn networks. *IEEE/ACM Transactions on Networking*, PP(99) :1–14, 2017.

[65] Matthew Caesar, Donald Caldwell, Nick Feamster, Jennifer Rexford, Aman Shaikh, and Jacobus van der Merwe. Design and implementation of a routing control platform. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 15–28, Berkeley, CA, USA, 2005. USENIX Association.

[66] Dan Levin, Marco Canini, Stefan Schmid, Fabian Schaffert, and Anja Feldmann. Panopticon : Reaping the benefits of incremental sdn deployment in enterprise networks. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 333–345, Philadelphia, PA, 2014. USENIX Association.

[67] Hui Lu, Nipun Arora, Hui Zhang, Cristian Lumezanu, Junghwan Rhee, and Guofei Jiang. Hybnet : Network manager for a hybrid network infrastructure. In *Proceedings of the Industrial Track of the 13th ACM/IFIP/USENIX International Middleware Conference*, Middleware Industry '13, pages 6 :1–6 :6, New York, NY, USA, 2013. ACM.

[68] Siyuan Huang, Jin Zhao, and Xin Wang. Hybridflow : A lightweight control plane for hybrid sdn in enterprise networks. In *2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)*, pages 1–2, June 2016.

[69] T. Feng and J. Bi. Openrouteflow : Enable legacy router as a software-defined routing service for hybrid sdn. In *2015 24th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–8, Aug 2015.

[70] Ryan Hand and Eric Keller. Closedflow : Openflow-like control over proprietary devices. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 7–12, New York, NY, USA, 2014. ACM.

[71] Stefano Vissicchio, Olivier Tilmans, Laurent Vanbever, and Jennifer Rexford. Central control over distributed routing. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 43–56, New York, NY, USA, 2015. ACM.

[72] Simple Network Management Protocol (SNMP). RFC 1157, May 1990.

[73] J. Raychev, G. Hristov, D. Kinaneva, and P. Zahariev. Modelling and evaluation of software defined network architecture based on queueing theory. In *2018 28th EAEEIE Annual Conference (EAEEIE)*, pages 1–5, Sep. 2018.

[74] Zhihao Shang and Katinka Wolter. Delay evaluation of openflow network based on queueing model. *CoRR*, abs/1608.06491, 2016.

[75] J. Hu, C. Lin, X. Li, and J. Huang. Scalability of control planes for software defined networks : Modeling and evaluation. In *2014 IEEE 22nd International Symposium of Quality of Service (IWQoS)*, pages 147–152, May 2014.

[76] Sdn simulators. `https://www.brianlinkletter.com/open-source-network-simulators/`. [Online].

[77] Omnet++. `https://omnetpp.org/`. [Online].

[78] Dominik Klein and Michael Jarschel. An openflow extension for the omnet++ inet framework. In *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*,

SimuTools '13, pages 322–329, ICST, Brussels, Belgium, Belgium, 2013. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[79] Ns-3. https://www.nsnam.org/docs/release/3.13/models/html/openflow-switch.html#id2/. [Online].

[80] M. Jarschel, F. Lehrieder, Z. Magyari, and R. Pries. A flexible openflow-controller benchmark. In *2012 European Workshop on Software Defined Networking*, pages 48–53, Oct 2012.

[81] Iperf. https://iperf.fr/. [Online].

[82] Inet. https://inet.omnetpp.org/. [Online].

[83] M. A. Salih, J. Cosmas, and Y. Zhang. Openflow 1.3 extension for omnet++. In *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, pages 1632–1637, Oct 2015.

[84] G. Xu, B. Dai, B. Huang, and J. Yang. Bandwidth-aware energy efficient routing with sdn in data center networks. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, pages 766–771, Aug 2015.

[85] Nicholas Gray, Thomas Zinner, Steffen Gebert, and Phuoc Tran-Gia. Simulation framework for distributed sdn-controller architectures in omnet++. In Ramón Agüero, Yasir Zaki, Bernd-Ludwig Wenning, Anna Förster, and Andreas Timm-Giel, editors, *Mobile Networks and Management*, pages 3–18, Cham, 2017. Springer International Publishing.

[86] Nox. https://github.com/noxrepo/. [Online].

[87] ryu. https://osrg.github.io/ryu/. [Online].

[88] floodlight. http://www.projectfloodlight.org/floodlight/. [Online].

[89] Sdxcentral. https://www.sdxcentral.com/. [Online].

[90] Amin Tootoonchian, Sergey Gorbunov, Yashar Ganjali, Martin Casado, and Rob Sherwood. On controller performance in software-defined networks. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, Hot-ICE'12, pages 10–10, Berkeley, CA, USA, 2012. USENIX Association.

[91] Beacon. https://openflow.stanford.edu/display/Beacon/Home. [Online].

[92] Maestro. https://code.google.com/archive/p/maestro-platform/. [Online].

[93] R. Khondoker, A. Zaalouk, R. Marx, and K. Bayarou. Feature-based comparison and selection of software defined networking (sdn) controllers. In *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*, pages 1–7, Jan 2014.

[94] Alexander Shalimov, Dmitry Zuikov, Daria Zimarina, Vasily Pashkov, and Ruslan Smeliansky. Advanced study of sdn/openflow controllers. In *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia*, CEE-SECR '13, pages 1 :1–1 :6, New York, NY, USA, 2013. ACM.

[95] S. A. Shah, J. Faiz, M. Farooq, A. Shafi, and S. A. Mehdi. An architectural evaluation of sdn controllers. In *2013 IEEE International Conference on Communications (ICC)*, pages 3504–3508, June 2013.

[96] O. Salman, I. H. Elhajj, A. Kayssi, and A. Chehab. Sdn controllers : A comparative study. In *2016 18th Mediterranean Electrotechnical Conference (MELECON)*, pages 1–6, April 2016.

[97] Open-mul. `http://www.openmul.org/`. [Online].

[98] Iris. `https://openiris.etri.re.kr/`. [Online].

[99] L. Mamushiane, A. Lysko, and S. Dlamini. A comparative evaluation of the performance of popular sdn controllers. In *2018 Wireless Days (WD)*, pages 54–59, April 2018.

[100] OSGI. `https://www.osgi.org/`. [Online].

[101] Ofdp. `http://groups.geni.net/geni/wiki/OpenFlowDiscoveryProtocol`. [Online].

[102] Lldp. `http://www.ieee802.org/1/files/public/docs2002/lldp-protocol-00.pdf`. [Online].

[103] F. Pakzad, M. Portmann, W. L. Tan, and J. Indulska. Efficient topology discovery in software defined networks. In *2014 8th International Conference on Signal Processing and Communication Systems (ICSPCS)*, pages 1–8, Dec 2014.

[104] Farzaneh Pakzad, Marius Portmann, Wee Lum Tan, and Jadwiga Indulska. Efficient topology discovery in openflow-based software defined networks. *Computer Communications*, 77 :52 – 61, 2016.

[105] OFDPv2-A. `https://github.com/Farzaneh1363/RouteFlow/blob/discovery-version-2A/pox/pox/openflow/discovery.py/`. [Online].

[106] OFDPv2-B. `https://github.com/Farzaneh1363/RouteFlow/blob/discovery-version-2B/pox/pox/openflow/discovery.py/`. [Online].

[107] A. S. Muqaddas, A. Bianco, P. Giaccone, and G. Maier. Inter-controller traffic in onos clusters for sdn networks. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6, May 2016.

[108] zoo-topology. `http://www.topology-zoo.org/`. [Online].

[109] Radia Perlman and Dr. Joseph D. Touch. Transparent Interconnection of Lots of Links (TRILL) : Problem and Applicability Statement. RFC 5556, May 2009.

[110] N. Bizanis and F. A. Kuipers. Sdn and virtualization solutions for the internet of things : A survey. *IEEE Access*, 4 :5591–5606, 2016.

[111] Iqbal Alam, Kashif Sharif, Fan Li, Zohaib Latif, Md Monjurul Karim, Boubakr Nour, Sujit Biswas, and Yu Wang. Iot virtualization : A survey of software definition function virtualization techniques for internet of things, 2019.

[112] Andrea Bianco, Paolo Giaccone, Reza Mashayekhi, Mario Ullio, and Vinicio Vercellone. Scalability of onos reactive forwarding applications in isp networks. *Computer Communications*, 102 :130 – 138, 2017.

[113] F. Bannour, S. Souihi, and A. Mellouk. Adaptive state consistency for distributed onos controllers. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7, Dec 2018.

[114] A. Bianco, P. Giaccone, A. Mahmood, M. Ullio, and V. Vercellone. Evaluating the sdn control traffic in large isp networks. In *2015 IEEE International Conference on Communications (ICC)*, pages 5248–5253, June 2015.

[115] S. Asadollahi, B. Goswami, and M. Sameer. Ryu controller's scalability experiment on software defined networks. In *2018 IEEE International Conference on Current Trends in Advanced Computing (ICCTAC)*, pages 1–5, Feb 2018.

[116] Q. Zuo, M. Chen, K. Ding, and B. Xu. On generality of the data plane and scalability of the control plane in software-defined networking. *China Communications*, 11(2) :55–64, Feb 2014.

[117] Jeffrey C. Mogul and Paul Congdon. Hey, you darned counters ! : Get off my asic ! In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 25–30, New York, NY, USA, 2012. ACM.

[118] A. Kondel and A. Ganpati. Evaluating system performance for handling scalability challenge in sdn. In *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, pages 594–597, Oct 2015.

[119] M. Darianian, C. Williamson, and I. Haque. Experimental evaluation of two openflow controllers. In *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, pages 1–6, Oct 2017.

[120] Atomix. `https://atomix.io/`. [Online].

[121] Gnuplot. `http://www.gnuplot.info/`. [Online].

[122] Joe Clarke, Gonzalo Salgueiro, and Carlos Pignataro. Interface to the Routing System (I2RS) Traceability : Framework and Information Model. RFC 7922, June 2016.

[123] Eric Voit, Alex Clemm, and Alberto Gonzalez Prieto. Requirements for Subscription to YANG Datastores. RFC 7923, June 2016.

[124] IETF 95 Hackathon. `https://www.ietf.org/proceedings/95/slides/slides-95-i2rs-2.pdf/`. [Online].

[125] IETF 96 Hackathon. `https://www.ietf.org/proceedings/96/slides/slides-96-i2rs-2.pdf/`. [Online].