

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Discipline : INFORMATIQUE

Spécialité : Réseaux et Télécommunications

Présentée et soutenue publiquement par

MAHAMAT CHARFADINE SALIM

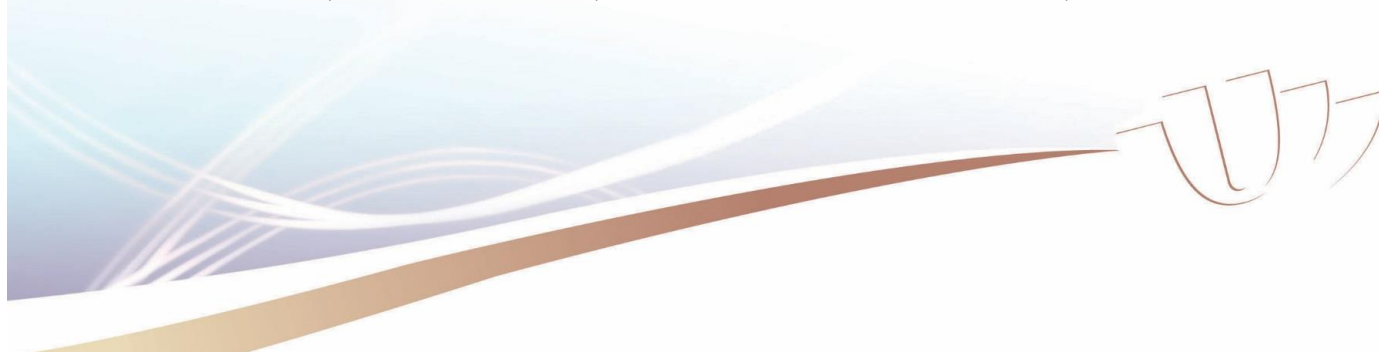
Le 2 juillet 2019

Gestion dynamique et évolutive de règles de sécurité pour l'Internet des Objets

Thèse dirigée par **OLIVIER FLAUZAC ET FLORENT NOLOT**

JURY

M. Hacène FOUCHAL,	Professeur,	Université de Reims Champagne-Ardenne,	Président
M. Olivier FLAUZAC,	Professeur,	Université de Reims Champagne-Ardenne,	Directeur de thèse
M. Florent NOLOT,	Maître de Conférences HDR,	Université de Reims Champagne-Ardenne,	Co-Directeur de thèse
M. Ibrahima NIANG,	Professeur,	Université Cheikh Anta Diop, Dakar, Sénégal,	Rapporteur
M. Antoine GALLAIS,	Maître de Conférences HDR,	Université de Strasbourg,	Rapporteur
Mme Manuele KIRSCH PINHEIRO,	Maître de Conférences ,	Université Paris 1 Panthéon-Sorbonne,	Examineur



“ À la mémoire de ma mère ”

Remerciements

À *Olivier FLAUZAC*, Professeur des Universités et *Florent NOLOT*, Maître de Conférences et HDR, pour leur encadrement, leurs précieux conseils ainsi que leur soutien moral tout au long de mes recherches.

Aux membres du jury de m'avoir fait l'honneur de bien vouloir accepter d'évaluer mes travaux.

À *Cyril RABAT*, Maître de Conférences, pour nos nombreux échanges, ses pertinents conseils et orientations durant mes travaux.

À *Frédérique RABAT* pour sa bienveillante relecture.

À *Ida Lenclume*, secrétaire du *CRéSTIC*, pour sa disponibilité et toute l'aide qu'elle m'a apportée durant ces années de thèse.

Aux membres de l'équipe *CASH* et du *CRéSTIC* pour ces moments d'échanges et de partage.

À *Carlos GONZALES* et à mes amis doctorants *Abdel-nassir MAHAMAT NASSOUR*, *Livinus TUYISENGE* et *Erick GALLEGOS*, pour leur aide, conseils pertinents et ces moments de collaboration plein de bonne humeur.

Au personnel de l'Université de Reims Champagne-Ardenne dans son ensemble et à tous ceux qui ont participé de façon directe ou indirecte au bon déroulement de cette thèse.

À ma famille et à mes amis, pour leur présence en toutes circonstances.

Table des matières

Table des matières	vii
Liste des figures	xi
Liste des tableaux	xiii
Introduction	1
Contribution de la thèse	2
Organisation de la thèse	3
Liste des publications	5
Conférences Internationales	5
Posters	5
I Etat de l’art des nouvelles architectures réseaux	7
1 Le Software Defined Networking (SDN)	9
1.1 Introduction	9
1.2 Le concept du SDN	11
1.3 Différence entre un réseau SDN et un réseau traditionnel	12
1.4 Architecture	13
1.4.1 La couche infrastructure	13
1.4.2 La couche de contrôle	16
1.4.3 La couche application	19
1.5 Les interfaces de programmation du SDN	20
1.5.1 La Southbound API	20
1.5.2 La Northbound API	25
1.5.3 La East-Westbound API	26

1.6	Conclusion	27
2	L'Internet des Objets	29
2.1	Introduction	29
2.2	Les enjeux de l'IoT	30
2.3	Architecture de l'IoT	31
2.3.1	La couche perception	32
2.3.2	La couche réseau	33
2.3.3	La couche application	33
2.4	Les protocoles de l'IoT	34
2.5	Les architectures SDN pour l'IoT	35
2.6	Conclusion	37
3	La sécurité des réseaux avec le SDN	39
3.1	Introduction	39
3.2	Mécanismes de sécurité des réseaux traditionnels	40
3.2.1	Protection par firewall	40
3.2.2	Protection par IDS/IPS	41
3.2.3	Protection par Network Access Control (NAC)	42
3.3	Avantages du SDN par rapport aux réseaux traditionnels	43
3.4	Défis de sécurité des architectures SDN	44
3.5	Historique de la sécurité des réseaux programmables	44
3.6	Analyse des menaces de sécurité du SDN	45
3.7	Solutions de sécurité pour les différentes couches du SDN	50
3.7.1	Solutions de sécurité pour la couche infrastructure	50
3.7.2	Solutions de sécurité pour la couche de contrôle	52
3.7.3	Solutions de sécurité pour la couche application	54
3.7.4	Conclusion	55
II	Nouvelles architectures de sécurité pour les réseaux du futur	57
4	Gestion intelligente de la sécurité dans un réseau avec le SDN	59
4.1	Introduction	59
4.2	Approches existantes de sécurité avec le SDN	60
4.3	Gestion intelligente de la sécurité avec le SDN	61
4.3.1	Gestion impérative des politiques de sécurité avec OpenFlow	62
4.3.2	Gestion déclarative des politiques de sécurité avec OpFlex	62

4.3.3	Notre solution de firewall Intelligent	63
4.3.4	Conclusion	69
5	Gestion automatisée du réseau en fonction des vulnérabilités détectées	71
5.1	Introduction	71
5.2	Sécurité IoT et SDN	72
5.3	Architecture de gestion distribuée et collaborative de sécurité dans un réseau . .	73
5.3.1	Collecte des données du réseau	75
5.3.2	Analyse, détection et génération des alertes	76
5.3.3	Suppression des flux malveillants	76
5.4	Conclusion	77
6	Implémentation	79
6.1	Introduction	79
6.2	Les outils utilisés	80
6.2.1	Le contrôleur OpenDaylight	80
6.2.2	Open vSwitch (OVS)	82
6.2.3	Création des machines virtuelles	83
6.2.4	Surveillance de trafic et gestion de flux malveillants	83
6.3	Mécanisme d'échanges de flux dans un réseau	85
6.3.1	Mécanisme d'échanges dans un réseau traditionnel	86
6.3.2	Mécanisme d'échanges dans un réseau SDN	87
6.3.3	Découverte de topologie dans un réseau SDN	88
6.4	Implémentation	90
6.4.1	Installation d'OpenDaylight	90
6.4.2	Réalisation de l'architecture	93
6.4.3	Sécurisation du lien entre le contrôleur OpenDaylight et l'OVS	93
6.4.4	Mise en place de Snort	95
6.4.5	Liaison de Snort avec OpenDaylight	96
6.4.6	Analyse, détection et suppression de flux	96
6.5	Quelques exemples d'attaques simulées	100
6.5.1	Déni de services	100
6.5.2	Scan de ports	100
6.5.3	Usurpation d'adresse IP ou MAC	100
6.6	Conclusion	101

Conclusion et perspectives	103
Conclusion	103
Perspectives	104
Liste des acronymes	105
Bibliographie	107

Table des figures

1.1	Processus d'encapsulation et de désencapsulation du modèle OSI	10
1.2	Comparaison entre un réseau traditionnel et un réseau SDN	12
1.3	Architecture détaillée d'un réseau SDN [1]	13
1.4	Architecture d'un commutateur OpenFlow	15
1.5	Diagramme de traitement d'un paquet dans un commutateur OpenFlow	16
1.6	Architecture du contrôleur OpenDaylight (version Beryllium)	19
1.7	Échanges des messages OpenFlow entre le contrôleur et le commutateur	22
1.8	Architecture OpFlex [2]	24
2.1	Comparaison entre le modèle TCP/IP et l'architecture IoT de l'IETF[3].	32
2.2	Architecture pour l'Internet des Objets [4].	32
3.1	Exemple d'un réseau protégé par un firewall	41
3.2	Analyse des vulnérabilités du SDN	46
4.1	Sécurité réseau avec SDN utilisant le protocole OpFlex	64
4.2	Mécanisme d'échanges de flux dans un réseau OpenFlow	65
4.3	Diagramme d'échanges OpenFlow	66
4.4	Principe de fonctionnement du protocole OpFlex	67
4.5	Diagramme d'échanges lors de l'exécution du firewall	68
4.6	Exemple d'enregistrement d'un <i>Endpoint</i>	69
5.1	Architecture distribuée et collaborative de sécurisation d'un réseau avec du SDN	74
5.2	Approche théorique de la solution	75
5.3	Intégration d'un port mirroring dans le contexte SDN	76
6.1	Architecture simplifiée d'une plateforme OpenDaylight	81
6.2	Schéma théorique d'une plateforme OpenDaylight pour la programmation réseau	82
6.3	Schéma de l'architecture Snort	85

6.4	Échanges de flux dans un réseau en IPv4	86
6.5	Mécanisme d'échanges de flux dans un réseau OpenFlow	87
6.6	Messages OpenFlow échangés entre le contrôleur SDN et l'OVS	88
6.7	Structure d'un LLDPDU	89
6.8	Découverte des liens avec OFDP	89
6.9	Maquette de mise en oeuvre de sécurisation d'un réseau avec du SDN	90
6.10	Ecran de lancement OpenDaylight	91
6.11	Interface graphique d'OpenDaylight Beryllium-SR4	92
6.12	Inventaire des noeuds sur l'interface graphique d'OpenDaylight Beryllium-SR4 .	92
6.13	Fichier de configuration OpenFlow pour supporter TLS	94
6.14	Ecran d'affichage de flux installés sur l'OVS	98
6.15	Résultats de l'écran d'exécution d'alerte et de désinstallation automatisée de flux malveillants	98
6.16	Ecran d'exécution de l'algorithme	99

Liste des tableaux

1.1	Exemple des commutateurs OpenFlow disponibles sur le marché	14
1.2	Champs de correspondance OpenFlow	15
1.3	Quelques types de contrôleurs et leurs caractéristiques	18
1.4	Fonctionnalités des différentes spécifications d'OpenFlow	21
2.1	Technologies de Communications pour l'Internet des Objets [5]	34
3.1	Quelques types de menaces de sécurité du SDN [6]	50

Introduction

Les réseaux permettent de communiquer, de collaborer et d'interagir de plusieurs manières. Ils sont utilisés pour accéder aux pages web, pour participer à des échanges vidéo, pour acheter sur Internet, pour suivre des formations sur Internet et bien plus encore avec les terminaux mobiles et les objets connectés.

En 2030, il est annoncé plus de 500 milliards¹ d'équipements connectés à Internet avec une variété d'usage entraînant des problèmes de sécurité et une augmentation du trafic sur les réseaux qui sera estimée en Zeta(10^{21}) octets. Or, actuellement, les architectures de sécurité déployées dans les réseaux sont principalement issues de l'expérience et des travaux sur les réseaux filaires. Ces architectures sont principalement basées sur des équipements centralisés, dont le rôle principal est de contrôler les informations qui sont échangées entre le réseau de l'entreprise et l'extérieur. Il n'est donc pas possible de contrôler les informations échangées entre un équipement terminal qu'un utilisateur va venir connecter sur son ordinateur. L'exemple du téléphone est un des cas les plus problématiques. Sur un réseau d'entreprise, les utilisateurs peuvent connecter leur téléphone sur leur ordinateur, via du Bluetooth par exemple et ainsi, l'ordinateur devient une nouvelle porte d'entrée sur le réseau. Avec l'Internet des Objets ou *Internet of Things* (IoT), nous avons des capteurs, des thermostats, des webcams, des montres connectées à nos téléphones, eux-mêmes éventuellement connectés à Internet ou à nos ordinateurs. Comment pouvons-nous donc effectuer le contrôle des informations qui proviennent de cette grande masse d'équipements hétérogènes ? Avec l'augmentation du nombre de ces équipements hétérogènes, la complexité dans leur administration devient croissante. Ce qui nécessite une vérification de la cohérence des configurations de tous les équipements réseaux d'une entreprise, par exemple les règles de sécurité et les droits utilisateurs.

Depuis ces dernières années, il se développe un nouveau concept de gestion de réseau appelé *Software Defined Networking* (SDN)². Son objectif principal est de simplifier la gestion et la

1. <https://www.cisco.com/c/dam/en/us/products/collateral/se/internet-of-things/at-a-glance-c45-731471.pdf> (Online : accessed on 04/09/2018)

2. <https://www.opennetworking.org/sdn-definition/> (Online : accessed on 04/09/2018)

configuration des réseaux et de faciliter l'innovation. Son principe est la séparation du plan de contrôle des équipements réseaux (commutateurs, routeurs...) du plan de données et le placer vers un point de contrôle centralisé. Appelé contrôleur SDN, ce point de contrôle centralisé permet de programmer les équipements réseaux à travers des interfaces programmables. Il permet également d'avoir une vue globale du réseau. Avec le SDN, les administrateurs réseaux n'ont plus à configurer chaque équipement séparément comme dans les réseaux traditionnels où le risque d'erreur est élevé.

Du point de vue sécurité, le SDN permet de programmer des politiques de sécurité et de pousser sur les équipements réseaux grâce au contrôleur SDN. Le SDN présente aussi des avantages multiples, entre autres : la programmabilité, l'évolutivité, la flexibilité, la scalabilité, la réduction des coûts d'investissement et d'exploitation des réseaux, mais soulève également des préoccupations liées à la centralisation du contrôle sur un équipement unique, qui devient un point critique et l'introduction des nouvelles interfaces vulnérables aux attaques.

Dans le cadre de ce travail de thèse, notre objectif est d'étudier et de réaliser de nouvelles solutions de sécurité des échanges dans un réseau en exploitant le concept du SDN.

Contribution de la thèse

Nos contributions dans ce projet de recherche sont la conception et la mise en oeuvre d'un nouveau type d'architecture de sécurisation des échanges dans un réseau exploitant le concept du SDN. Ce nouveau paradigme de gestion des réseaux, nous a permis de proposer une architecture de gestion centralisée, distribuée et collaborative de la sécurité grâce à son contrôleur. L'avantage du contrôleur SDN est qu'il permet, non seulement d'avoir une vue globale du réseau, mais aussi de programmer les politiques de sécurité sur les équipements. Ce qui fait que notre solution est programmable, évolutive et adaptée aux réseaux IoT.

Après une étude des architectures des solutions de sécurité avec le SDN, nous avons proposé notre solution globale de sécurisation des échanges basée sur le SDN, combinée à un système d'analyse pour détecter et isoler une menace en cas de besoin. Notre travail, qui vient compléter la limitation des techniques existantes basées sur les firewalls et les IDS pour sécuriser un réseau traditionnel, sera étendu aux objets connectés.

Nos expérimentations ont été faites en environnement virtuel. Ce qui nous a permis de simuler des réseaux SDN et de tester nos solutions de sécurité. Nous avons mis en oeuvre plusieurs scénarios d'attaques comme du déni de service (DoS) ou des scans de port. Cette solution est évolutive et adaptable.

Les résultats obtenus présentent une simplification de la gestion des réseaux et de leur sécurité, même en cas d'augmentation de leur taille. Grâce à nos tests de mise en oeuvre

d'un réseau SDN et l'implémentation des politiques de sécurité, nous pouvons dire que des perspectives s'ouvrent pour sécuriser les objets connectés.

Nos simulations nous permettront à l'avenir de tester et de valider des solutions sur des réseaux de grandes tailles.

Organisation de la thèse

Notre travail de thèse est structuré de la manière suivante :

- Le premier chapitre présente un aperçu du concept de SDN. Ce chapitre introduit la définition du SDN, puis une présentation de son architecture, de ses protocoles, de son déploiement et d'autres services réseaux qui le caractérisent. Ce nouveau concept d'architecture réseau propose une architecture centralisée et programmable des réseaux grâce à son protocole OpenFlow que nous allons détailler dans ce chapitre.
- Le deuxième chapitre décrit les généralités sur la notion de l'IoT. Ensuite, nous présenterons les protocoles utilisés dans l'IoT et leur interconnexion.
- Nous proposerons dans le troisième chapitre un état de l'art sur la sécurité des réseaux utilisant le concept du SDN. Il s'agit de faire une étude sur les solutions existantes de sécurisation des échanges dans un réseau afin de connaître leurs performances et leurs limites. Ce qui nous permettra de proposer notre approche pour sécuriser un réseau en utilisant le concept du SDN étendu à l'IoT.
- Au quatrième chapitre, nous allons présenter notre solution de gestion distribuée de la sécurité dans un réseau avec le paradigme SDN.
- Nous proposons au cinquième chapitre une nouvelle solution de gestion automatisée de la sécurité dans un réseau en fonction des vulnérabilités détectées.
- Le sixième chapitre sera exclusivement consacré à l'implémentation de notre solution et à la discussion des résultats obtenus lors des scénarios des tests, pour démontrer la faisabilité et l'efficacité de notre solution.

Nous concluons cette thèse par des perspectives que nous avons identifiées.

Liste des publications

Conférences Internationales

1. C. GONZALES, **S. MAHAMAT CHARFADINE**, O. FLAUZAC, and F. NOLOT, **SDN-based security framework for the IoT in distributed grid**. In : The IEEE International Multidisciplinary Conference on Computer and Energy Science (SPLITECH) 2016, Split, Croatia.
2. **S. MAHAMAT CHARFADINE**, O. FLAUZAC, F. NOLOT, C.RABAT and C. GONZALES, **Secure exchanges activity in function of event detection with the SDN**. In : 10th EAI International Conference on e-Infrastructure and e-Services for Developing Countries AFRICOMM 2018, Dakar , Sénégal.

Posters

1. **S. MAHAMAT CHARFADINE**, O. FLAUZAC, F. NOLOT, **Perspectives de sécurité pour l'Internet des Objets (IoT)** : Journée des Doctorants du CReSTIC 2016 Reims, France.
2. **S. MAHAMAT CHARFADINE**, O. FLAUZAC, F. NOLOT **Sécurisation des échanges via du SDN, pour l'Internet des Objets (IoT)**. SDN DAY 2017 à l'IRT SystemX PARIS, France.

Première partie

Etat de l'art des nouvelles architectures réseaux

CHAPITRE 1

Le Software Defined Networking (SDN)

Résumé : Dans ce chapitre, nous allons présenter le SDN, un nouveau concept émergent d'architecture réseau qui a introduit la programmabilité des réseaux et qui a fait émerger des nouveaux protocoles tels que OpenFlow et OpFlex. Nous expliquerons en détails le principe de fonctionnement du protocole OpenFlow pour comprendre le comportement des paquets dans un réseau SDN, ainsi que les différents types de messages échangés. Ensuite nous allons décrire le protocole OpFlex qui est aussi proposé pour définir et implémenter des politiques de sécurité dans un réseau basé sur l'approche SDN.

1.1 Introduction

D'une manière générale, un réseau informatique est un ensemble d'équipements hétérogènes (commutateurs, routeurs, ordinateurs etc.) interconnectés entre eux pour échanger des informations. Des modèles standards permettent aux différents équipements du réseau de s'échanger des informations. Les deux principaux modèles utilisés pour la planification et la mise en oeuvre des réseaux sont : le modèle *Open System Interconnexion* (OSI) [7] et TCP/IP¹. Le modèle OSI est une norme de communication réseau, développé par *International Organisation for Standardization* (ISO) en 1984, précisant comment les équipements doivent communiquer entre eux. Son architecture est composée de sept couches, à savoir de haut en bas : application, présentation, session, transport, réseau, liaison de données et physique. Les quatre premières couches (1-4) sont des couches réseau et servent, de service de communication aux couches applicatives (5-7). Chaque couche est indépendante des autres avec un rôle spécifique à accomplir. À titre d'exemple, lors de l'envoi de données, on parcourt le modèle OSI de haut en bas comme l'indique la figure 1.1, en traversant toutes les couches.

1. <https://tools.ietf.org/html/rfc1180>(Online : accessed on 04/09/2018)

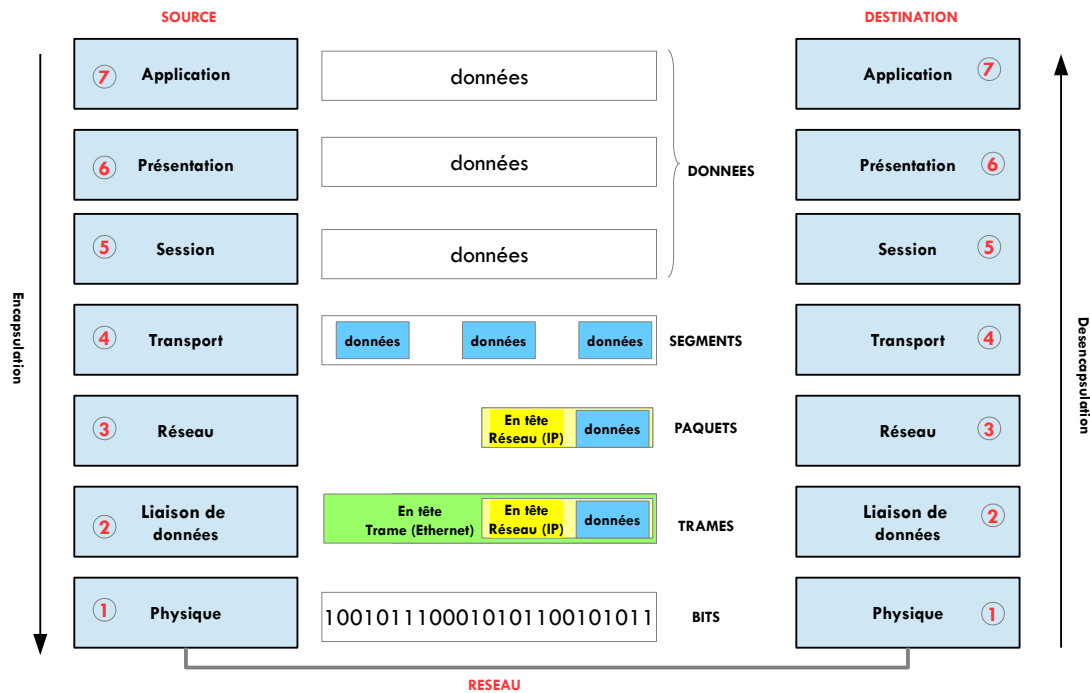


FIGURE 1.1 – Processus d’encapsulation et de désencapsulation du modèle OSI

Le modèle TCP/IP est un modèle de communication en couches, similaire au modèle OSI, composé de quatre couches : Application, Transport, Internet et Accès Réseau.

Ces réseaux traditionnels sont devenus de plus en plus complexe à administrer et à sécuriser [8] malgré leur large adoption. C’est-à-dire que les administrateurs des réseaux passent beaucoup de temps à configurer manuellement les équipements réseaux à travers des lignes de commande. En plus, ces équipements réseaux sont intégrés verticalement et tournent le plus souvent avec des logiciels propriétaires [9]. Par exemple, dans un équipement réseau traditionnel, la partie qui contrôle le routage des paquets et la partie qui gère l’acheminement des paquets sont ensemble sur un seul dispositif ce qui rend difficile son évolution. Autrement, ce couplage limite l’innovation puisque toute introduction d’un nouveau protocole ou service sur l’équipement réseau, passe obligatoirement par le fabricant et peut être parfois très long. C’est pour pallier ces problèmes de rigidité architecturale, et bien d’autres encore, que le SDN a été introduit depuis 2011.

1.2 Le concept du SDN

Le SDN est né d'un large mouvement intellectuel, motivé par la question du pourquoi les équipements réseaux ne devraient-ils pas être programmables comme les autres plates-formes informatiques, et de la nécessité de résoudre les problèmes que l'on rencontre dans les réseaux informatiques : difficulté à gérer et à faire évoluer les réseaux. Le terme SDN a été introduit pour la première fois par Martin Casado², de l'Université de Stanford en Californie. Il n'était pas le seul à travailler sur le SDN mais il y avait aussi d'autres personnes comme Nick McKeown ou Scott Shenker qui font partie des gens souvent cités dans le domaine du SDN. Autrement dit, le SDN est le résultat des efforts de plusieurs chercheurs travaillant sur la problématique de l'architecture des réseaux, visant à rendre les réseaux programmables, flexibles, évolutifs et innovants [10, 11].

Le SDN est un nouveau concept émergent de gestion des réseaux. C'est une nouvelle façon de concevoir, de construire, d'exploiter et de sécuriser les réseaux. Il est basé sur une gestion centralisée des flux réseaux par le découplage du plan de contrôle, responsable d'associer une décision de routage aux paquets du plan de données représentant l'infrastructure physique ou virtuelle et qui s'occupe uniquement de l'acheminement des paquets rendant les réseaux flexibles et programmables³.

Avec le SDN, l'intelligence du réseau est externalisée et gérée par un équipement externe appelé "contrôleur" qui gère les fonctions du plan de contrôle. Cette entité intelligente qu'est le contrôleur peut être sur une ou plusieurs machines physiques ou virtuelles distribuées. La configuration du réseau reviendra à programmer le contrôleur via des *Application Programming Interface* (API) ouvertes appelées *Northbound API*. La communication entre le contrôleur et les équipements réseaux se fait à travers le *Southbound API* (par exemple OpenFlow) via un canal sécurisé.

L'engouement des acteurs du numérique, tels que Google et Microsoft [12, 13] dans le déploiement du SDN au niveau de leurs *data center*, donne des belles perspectives au concept du SDN qui commence à devenir une réalité.

En d'autres termes, l'objectif du SDN est de permettre aux réseaux d'être agiles, flexibles et programmables afin de rendre leur gestion simple. Ce concept de SDN est aujourd'hui globalement reconnu comme une architecture permettant d'ouvrir les réseaux aux applications et offre beaucoup d'avantages [14].

2. https://en.wikipedia.org/wiki/Martin_Casado(Online : accessed on 04/09/2018)

3. <https://www.opennetworking.org/sdn-definition/>(Online : accessed on 04/09/2018)

1.4 Architecture

Comme représenté sur la figure 1.3, l'architecture d'un réseau SDN est divisée en trois couches à savoir la couche infrastructure, la couche de contrôle et la couche application [9, 1]. La communication entre ces différentes couches est faite à travers les *Southbound*, *Northbound* et *East-Westbound* APIs comme l'indique la figure 1.3. Dans cette partie, nous allons détailler étape par étape ces différentes couches et APIs pour mieux expliquer l'architecture du SDN.

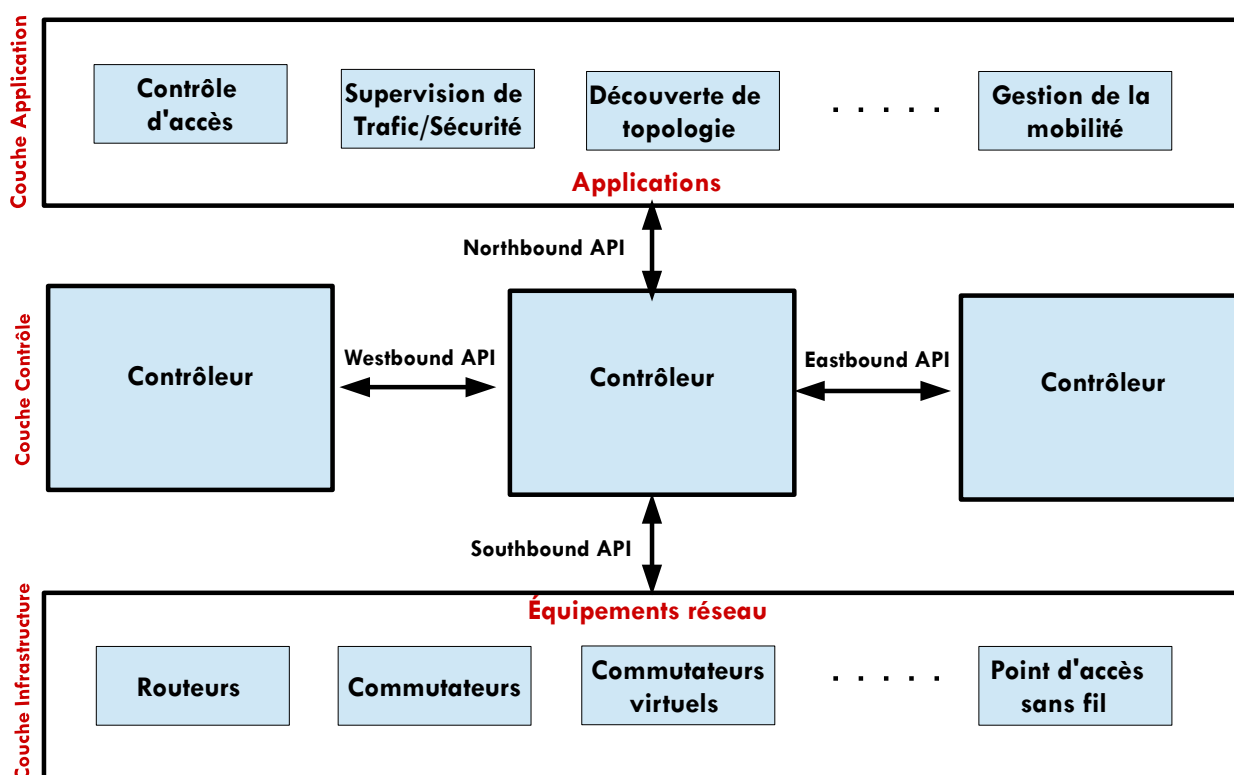


FIGURE 1.3 – Architecture détaillée d'un réseau SDN [1]

1.4.1 La couche infrastructure

Cette couche définit la fonctionnalité du plan de données. Le plan de données est constitué des équipements réseaux tels que commutateurs, routeurs... incluant des composants d'acheminement des paquets et des APIs [10]. Un équipement réseau est une entité qui reçoit des paquets dans ses ports et réalise une ou plusieurs fonctions réseaux (par exemple un paquet reçu sur un port peut être transféré, supprimé ou son entête modifiée pour une action spécifique).

D'une manière générale, un commutateur est un équipement réseau qui a pour fonction le transfert des paquets. Il est composé de deux parties fonctionnelles : le plan de données et le

plan de contrôle. Le plan de données est responsable de l'acheminement des paquets de la source à la destination. Il récupère les paquets de l'interface d'entrée, puis consulte la table de routage pour déterminer l'interface de sortie avant de transférer les paquets. Le plan de contrôle, quant à lui, est chargé de la construction et du maintien de la table de routage. Avec les commutateurs classiques le plan de données et le plan de contrôle sont sur un même équipement.

Le SDN a introduit une séparation entre le plan de données et le plan de contrôle. Le plan de données est toujours géré par le commutateur qu'on appelle commutateur OpenFlow. Il est conçu autour d'une interface de programmation ouverte et standard, qui lui permet d'assurer sa configuration et son interopérabilité avec le plan de contrôle et autres commutateurs SDN. Le plan de contrôle qui gère les décisions de routage de haut niveau est externalisé vers un dispositif de contrôle appelé Contrôleur SDN que nous avons précédemment évoqué. Le commutateur OpenFlow et le contrôleur communiquent via le protocole OpenFlow que nous allons détailler par la suite. Il y a deux types de commutateurs OpenFlow [15] : le commutateur OpenFlow Only qui supporte uniquement OpenFlow et le commutateur OpenFlow Enable qui joue en même temps le rôle de commutateur traditionnel et OpenFlow. Il y a sur le marché plusieurs marques de commutateurs OpenFlow comme l'indique le tableau 1.1.

Fabricant	Série
Arista	Arista extensible modular operating system (EOS), Arista 7124FX application switch
Ciena	Ciena Coredirector running firmware version 6.1.1
Cisco	Cisco cat6k, catalyst 3750, 6500 series
Juniper	Juniper MX-240, T-640
HP	HP procurve series- 5400 zl, 8200 zl, 6200 yl, 3500 yl, 6600
NEC	NEC IP8800
Pronto	Pronto 3240, 3290
Toroki	Toroki Lightswitch 4810
Dell	Dell Z9000 and S4810
Quanta	Quanta LB4G
Open vSwitch	Software switch. Latest version : 2.6

TABLE 1.1 – Exemple des commutateurs OpenFlow disponibles sur le marché

Comme l'indique la figure 1.4, un commutateur OpenFlow est composé des tables de flux en pipeline, une table de groupe et un canal sécurisé pour l'échange avec le contrôleur OpenFlow. Le contrôleur OpenFlow gère le commutateur OpenFlow à travers le protocole OpenFlow.

Une table de flux d'un commutateur OpenFlow est une liste de flux d'entrées. Chaque flux d'entrées est composé principalement : (1) d'un champ de correspondance dont le contrôleur se base pour faire correspondre les paquets, afin de vérifier le port d'entrée ou l'entête du paquet pour appliquer une action, (2) des instructions pour appliquer une liste d'actions sur les paquets

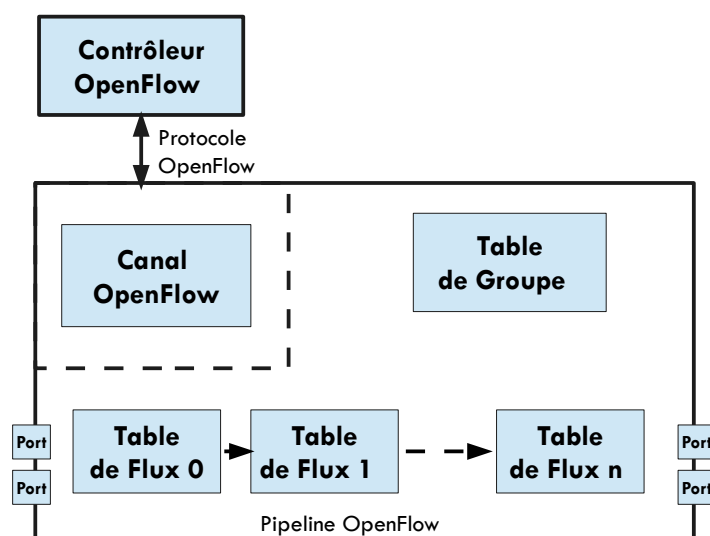


FIGURE 1.4 – Architecture d'un commutateur OpenFlow

et (3) des compteurs qui tiennent des statistiques sur le nombre de paquets traités.

Le nombre des champs de correspondance supportés par le protocole OpenFlow 1.3 est détaillé sur le tableau 1.2. Ils sont au nombre de 14. Ce nombre varie en fonction des versions d'OpenFlow.

Field	Description
OXM_OF_IN_PORT	Physical or logical port
OXM_OF_ACTSET_OUTPUT	Egress port from action set
OXM_OF_ETH_DST	Ethernet destination address
OXM_OF_ETH_SRC	Ethernet source address
OXM_OF_ETH_TYPE	Ethernet type of the OpenFlow packet payload
OXM_OF_IP_PROTO	IPv4 or IPv6 protocol number
OXM_OF_IPV4_SRC	IPv4 source address
OXM_OF_IPV4_DST	IPv4 destination address
OXM_OF_IPV6_SRC	IPv6 source address
OXM_OF_IPV6_DST	IPv6 destination address
OXM_OF_TCP_SRC	TCP source port
OXM_OF_TCP_DST	TCP destination port
OXM_OF_UDP_SRC	UDP source port
OXM_OF_UDP_DST	UDP destination port

TABLE 1.2 – Champs de correspondance OpenFlow

Les actions appliquées par un commutateur OpenFlow sont similaires à celles d'un commutateur traditionnel et sont : (1) transférer un paquet vers un ou plusieurs ports de sortie, (2) rejeter ou supprimer le paquet, (3) modifier les champs d'entête d'un paquet et (4) encapsu-

ler un paquet et ensuite l'envoyer au contrôleur SDN. Cette dernière action est spécifique aux commutateurs OpenFlow.

Quand un paquet arrive sur un port d'entrée du commutateur OpenFlow, le processus de correspondance des paquets commence par la Table 0, première table du pipeline comme l'indique la figure 1.5 et peut continuer au cas où plusieurs tables de flux existent. La correspondance des paquets se fait par ordre de priorité, si une correspondance est trouvée, alors les instructions associées aux flux d'entrées seront exécutées. Si aucune correspondance n'est trouvée, la suite du processus dépendra de la configuration par défaut du commutateur. Trois cas de figure sont possibles ; le paquet peut être envoyé au contrôleur supprimé ou passé à la table de flux suivante. En interne, un commutateur utilise une mémoire adressable de contenu ternaire, en anglais *Ternary Content Addressable Memory* (TCAM), et une mémoire d'accès aléatoire (*Random Access Memory* (RAM)) pour traiter les paquets.

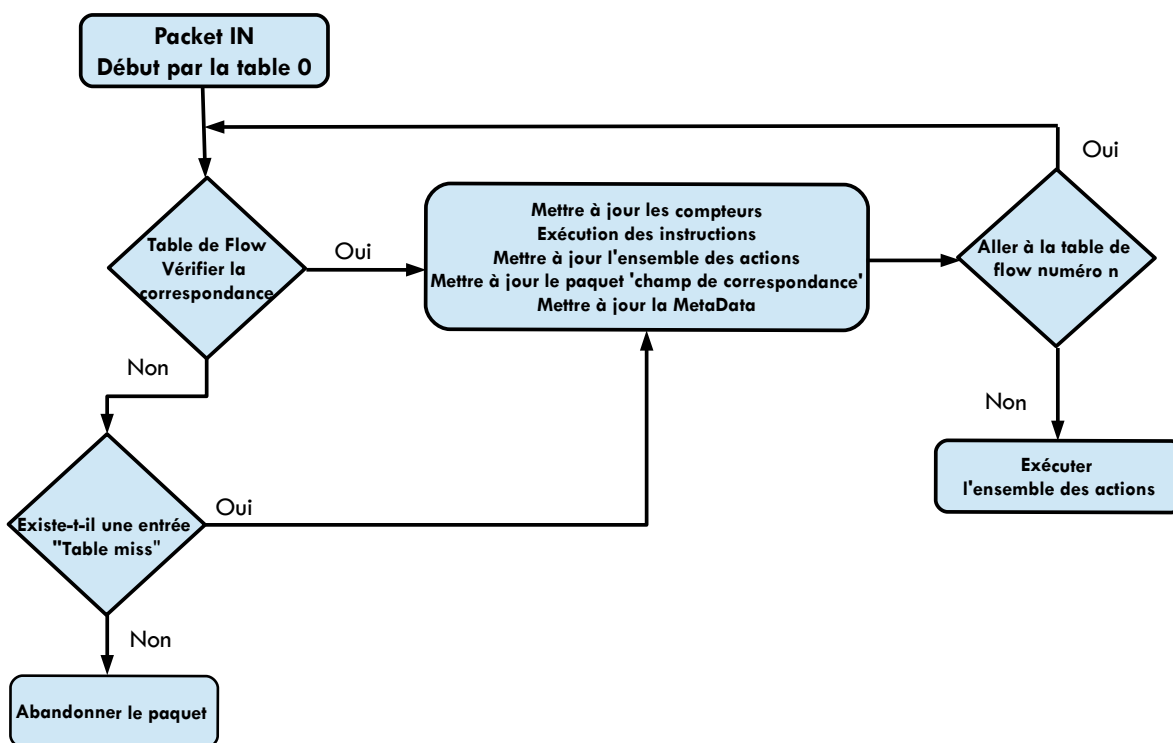


FIGURE 1.5 – Diagramme de traitement d'un paquet dans un commutateur OpenFlow

4

1.4.2 La couche de contrôle

La couche de contrôle de l'architecture SDN est constituée d'un contrôleur logiciel appelé contrôleur SDN qui centralise toute l'intelligence du réseau. Considéré comme la composante

la plus importante du concept SDN, le contrôleur SDN est un programme logiciel comparable à un système d'exploitation réseau, responsable de la manipulation de la table de flux d'un commutateur à travers le *Southbound* API. Autrement dit, le contrôleur SDN a la charge de la gestion et de la configuration du réseau en installant des règles d'acheminement des paquets, appropriées sur les équipements réseaux (commutateurs, routeurs) de la couche infrastructure à travers le *Southbound* API. Cette interface de programmation est très importante car elle offre la possibilité d'innovations pour les programmeurs et permet d'accélérer le développement et l'implémentation des services réseaux [16]. OpenFlow est le *Southbound* le plus utilisé. Il permet au contrôleur SDN d'ajouter, de modifier et de supprimer des entrées dans la table de transfert d'un commutateur OpenFlow. Un canal sécurisé relie le contrôleur à un commutateur OpenFlow. Grâce à ce canal qui sert d'interface de connexion, le contrôleur gère les commutateurs OpenFlow, reçoit des paquets des commutateurs OpenFlow et envoie des paquets aux commutateurs OpenFlow [17].

Il y a plusieurs types de contrôleur logiciel dont les plus connus sont ONOS⁵, OpenDaylight⁶, POX⁷, RYU⁸, Floodlight⁹ et Beacon¹⁰. Ils se distinguent par les langages de programmation utilisés et le protocole supporté. À titre d'exemple, et comme l'indique le tableau 1.3, le contrôleur OpenDaylight exploité dans le cadre de nos travaux est conçu en Java et en Python.

Le contrôleur tient une vue globale du réseau à travers son service de découverte de topologie, très important pour une exploitation correcte des autres services réseaux du contrôleur, comme la configuration réseau par exemple. Certains contrôleurs sont centralisés (Beacon, Floodlight) et d'autres distribués (ONOS, OpenDaylight) et intègrent des API supplémentaires comme *Eastbound* et *Westbound*. Les contrôleurs distribués permettent de relever les limites sur l'usage d'un contrôleur unique, qui peut non seulement devenir un point critique du point de vue de la sécurité, mais ne permet pas de gérer des larges réseaux avec un nombre élevé des équipements réseaux.

OpenDaylight est un contrôleur SDN Open source modulaire qui permet de concrétiser la virtualisation des fonctions réseaux à travers une définition cohérente de ses interfaces de programmation. Considéré comme une référence dans la programmabilité des réseaux, OpenDaylight permet à ses utilisateurs et équipementiers de personnaliser leurs solutions de gestion des réseaux selon leurs besoins.

Il y aussi des contrôleurs SDN propriétaires tels que XNC (Extensible Network Controller)

5. <http://www.onosproject.org>(Online : accessed on 14/09/2018)

6. <https://www.opendaylight.org/>(Online : accessed on 14/09/2018)

7. <https://openflow.stanford.edu/display/ONL/POX+Wiki>(Online : accessed on 14/09/2018)

8. <https://osrg.github.io/ryu/>(Online : accessed on 14/09/2018)

9. <http://www.projectfloodlight.org/floodlight/>(Online : accessed on 14/09/2018)

10. <http://www.beacon-project.eu/>(Online : accessed on 14/09/2018)

Contrôleur	Langages de programmation	Développé par	Version Open-Flow supportée	Open Source
NOX	C++	Nicira	1.0 , 1.3	oui
POX	Python	Nicira	1.0 , 1.3	oui
Trema	Ruby et C	NEC	1.0	oui
Maestro	Java	Rice University	1.0	oui
Beacon	Java	Stanford University	1.0	oui
Floodlight	Java	Big Switch	1.0 , 1.3, 1.4	oui
Ryu	Python	NEC	1.0, 1.2, 1.3, 1.4, 1.5	oui
ONOS	Java	ON.Lab	1.0	oui
Helio	C	NTT	1.0, 1.3	non
OpenDaylight	Java, Python	ONF	1.0 , 1.3	oui
Node.Flow	JavaScript	DreamersLab	1.0	oui
HP VAN	Controller Java, REST	HP	1.0 , 1.3	non
Cisco XNC	Java, Python, REST, C	Cisco	1.0 , 1.3	non

TABLE 1.3 – Quelques types de contrôleurs et leurs caractéristiques

développé par Cisco et compatible avec le protocole OpenFlow et OnePk (Cisco ONE Platform Kit).

Depuis son introduction, OpenDaylight a connu plusieurs versions dont chaque version apporte soit des correctifs, des nouveaux services ou le support des nouveaux protocoles. OpenDaylight est très largement adopté par les intégrateurs et par l'industrie des équipements.

L'architecture de la plateforme OpenDaylight, représentée sur la figure 1.6, est basée sur le *Model-Driven Service Abstraction Layer* (MD-SAL) où les équipements réseaux et les applications sont représentés sous forme d'objets ou de modèles dont les interactions sont traitées dans la couche d'abstraction appelée *Service Abstraction Layer* (SAL). Le SAL est un élément très important du contrôleur OpenDaylight, permettant d'assurer le mécanisme d'échange et d'adaptation de données entre les modèles YANG¹¹, représentant les équipements réseaux et les applications. Il permet aussi de faciliter l'intégration d'une nouvelle *Southbound* API pour le développement des services sans lien avec les API utilisées pour les implémenter.

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

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

11. <https://tools.ietf.org/html/rfc7950>(Online : accessed on 14/09/2018)

réseaux.

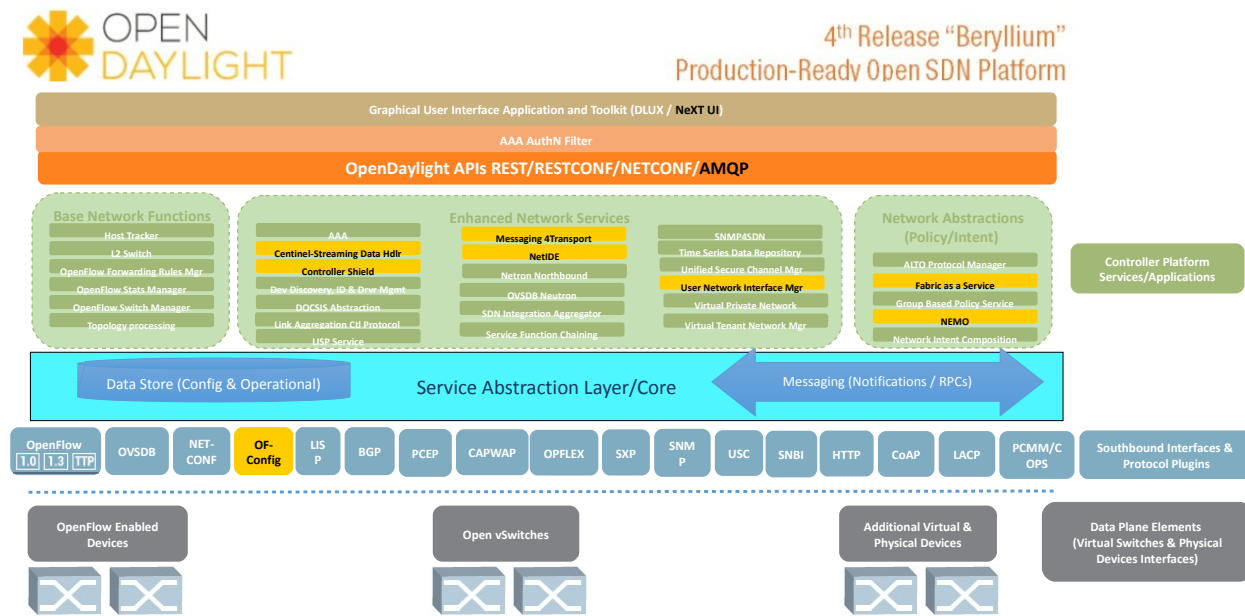


FIGURE 1.6 – Architecture du contrôleur OpenDaylight (version Beryllium)

12

1.4.3 La couche application

Elle est la couche la plus haute de l'architecture SDN où les règles de gestion sont définies et implémentées. Ces règles sont un ensemble d'applications telles que IDS, firewalls, load balancing... conçues pour fournir un service spécifique au réseau. Cette couche apporte l'automatisation des applications grâce aux APIs et permet également d'interagir et de manipuler le comportement des équipements réseaux à travers le contrôleur SDN. Grâce à la couche application, les applications bénéficient de la visibilité des ressources réseaux de manière spécifique, par exemple pour la découverte des liens ou de la topologie du réseau, il est nécessaire de faire un couplage entre les applications et les équipements réseaux sous-jacents tels que commutateurs et routeurs. C'est ainsi qu'en SDN, les fournisseurs des services et opérateurs pourront utiliser les applications pour la configuration, le contrôle et la supervision de leur réseau.

Dans l'approche SDN, le plan de contrôle fournit une vue globale du réseau et l'ensemble des informations sur les éléments du réseau aux applications via le *southbound* API du contrôleur. Comme nous l'avons déjà évoqué précédemment, OpenFlow est l'API standard de facto pour implémenter les fonctions réseaux sous la forme d'applications OpenFlow.

Avec cette couche, du point de vue de la sécurité, divers services peuvent être implémentés au-dessus du contrôleur OpenFlow en tant qu'applications de sécurité.

1.5 Les interfaces de programmation du SDN

Le SDN a introduit plusieurs types d'interfaces de programmation afin de permettre aux différents éléments de son architecture SDN d'interagir entre eux [18]. Dans cette section, nous allons détailler ces interfaces de programmation qui sont les *Southbound* API, les *Northbound* API et les *East-Westbound* API.

1.5.1 La Southbound API

L'interface entre la couche infrastructure et la couche de contrôle est appelée *Southbound* API. Cette API est la clé du concept SDN car c'est elle qui permet concrètement l'externalisation du plan de contrôle [1]. OpenFlow est la *Southbound* API la plus souvent utilisée dans l'architecture SDN permettant la programmation du plan de données. À travers cette interface, le contrôleur gère l'ensemble des flux des commutateurs ou routeurs sous son autorité. Bien qu'OpenFlow est la *Southbound* API la plus utilisée et même le standard de facto, il y a aussi plusieurs autres *Southbound* API pour gérer les communications entre ces deux niveaux de l'architecture SDN, telles que *Forwarding and Control Element Separation* (ForCES)¹³ et OpFlex¹⁴ que nous allons aussi aborder dans cette partie.

Protocole OpenFlow

OpenFlow est un protocole standardisé par *Open Network Foundation*(ONF)¹⁵, un consortium industriel à but non lucratif de plusieurs membres, créé en 2011 qui travaille dans le développement du SDN. Il a été proposé pour faciliter la configuration des réseaux, simplifier leurs gestions et virtualiser les réseaux afin de déployer plus facilement tous types de services (mobiles, sécurité...). OpenFlow est un bon moyen pour innover les réseaux mais aussi faire face à plusieurs challenges notamment de sécurité. Il existe plusieurs versions (voir le tableau 1.4) de spécification d'OpenFlow dont la plus récente au moment où nous écrivons ces lignes est la version 1.5.

La communication entre le contrôleur et le commutateur OpenFlow se fait à travers le protocole OpenFlow. Il traite la définition du format des messages échangés entre le contrôleur

13. <https://tools.ietf.org/html/rfc5810>(Online : accessed on 14/09/2018)

14. <http://tools.ietf.org/pdf/draft-smith-opflex-00.pdf>(Online : accessed on 14/09/2018)

15. <https://www.opennetworking.org/sdn-resources/openflow>(Online : accessed on 17/09/2018)

Spécification OpenFlow	1.0	1.1	1.2	1.3
Large déploiement	oui	non	non	oui
Nombre de table de flux	Une	Multiple	Multiple	Multiple
Matching MPLS	non	oui	oui	oui
Table de groupe	non	oui	oui	oui
Support IPv6	non	non	oui	oui
Contrôleur multiples	non	non	oui	oui

TABLE 1.4 – Fonctionnalités des différentes spécifications d'OpenFlow

et le commutateur OpenFlow via le canal sécurisé en SSL¹⁶. Les messages doivent être générés et compris par le contrôleur et le commutateur SDN. Le protocole OpenFlow donne la possibilité au contrôleur de modifier, d'ajouter, de mettre à jour et de supprimer des flux d'entrée dans la table de flux. Les messages entre le contrôleur OpenFlow et le commutateur OpenFlow sont échangés à travers le canal sécurisé et implémentés via une connexion SSL en TCP. C'est le commutateur qui initie la connexion SSL s'il connaît l'adresse IP du contrôleur. Tous les messages échangés entre le contrôleur et le commutateur commencent par une entête OpenFlow spécifiant la version du protocole OpenFlow avec un numéro, le type de message, la longueur et l'identificateur du message. Il y a trois types de messages :

Messages symétriques : les messages HELLO , ECHO_REQUEST, ECHO_REPLY et VENDOR sont les principaux types de messages symétriques échangés entre le contrôleur et le commutateur OpenFlow. Ils peuvent être initiés par le contrôleur ou le commutateur sans sollicitation. Après l'établissement du canal sécurisé SSL avec TCP, les messages HELLO sont échangés entre le commutateur et le contrôleur pour déterminer le numéro de la version du protocole OpenFlow utilisé. Une fois le numéro échangé, comme spécifie le protocole OpenFlow, la plus petite des deux versions doit être utilisée. Les messages ECHO (ECHO_REQUEST et ECHO_REPLY) sont aussi utilisés par le commutateur et le contrôleur pendant leur fonctionnement, pour suivre si le lien est toujours actif et en même temps vérifier le débit de la connexion et mesurer la latence.

Messages asynchrones : les messages asynchrones sont initiés par le commutateur sans aucune requête du contrôleur. Ils sont utilisés pour informer le contrôleur des arrivées de paquets, des changements d'état au commutateur et des erreurs.

PACKET_IN est le type message utilisé par le commutateur pour envoyer des paquets au contrôleur pour leur prise en charge. Ce type de message est envoyé, par exemple, lorsqu'aucune entrée de flux du commutateur ne correspond au paquet entrant, ou lorsqu'il est spécifié au niveau de l'action de l'entrée correspondante que le paquet doit être transféré au contrôleur. D'une manière générale le trafic du plan de contrôle est relayé au contrôleur via le message

16. Secure Socket Layer

PACKET_IN.

Le Message `FLOW_REMOVED` permet au commutateur d'informer le contrôleur en cas d'une suppression de flux d'entrée de la table de flux. Le commutateur supprime un flux d'entrée lorsqu'aucun paquet entrant n'a de correspondance avec cette entrée pendant un temps défini par le contrôleur lors de la création de ce flux d'entrée dans la table de flux du commutateur.

`PORT_STATUS` est le message qui permet au commutateur d'annoncer au contrôleur tous changements de configuration ou d'état du port.

Le message `ERROR` est utilisé pour notifier les erreurs au contrôleur. Par exemple, ce message `ERROR` est envoyé au contrôleur lorsque ce dernier tente d'ajouter une entrée de flux contenant des actions non supportées par le commutateur.

Messages entre le contrôleur et le commutateur : les messages OpenFlow échangés entre le contrôleur et le commutateur, détaillés sur la figure 1.7, sont responsables de la détection des fonctionnalités, de la configuration, de la programmation du commutateur et de la récupération d'informations. Ces messages sont, entre autres : *switch configuration*, *command from controller*, *statistics*, *queue configuration* et *barrier*.

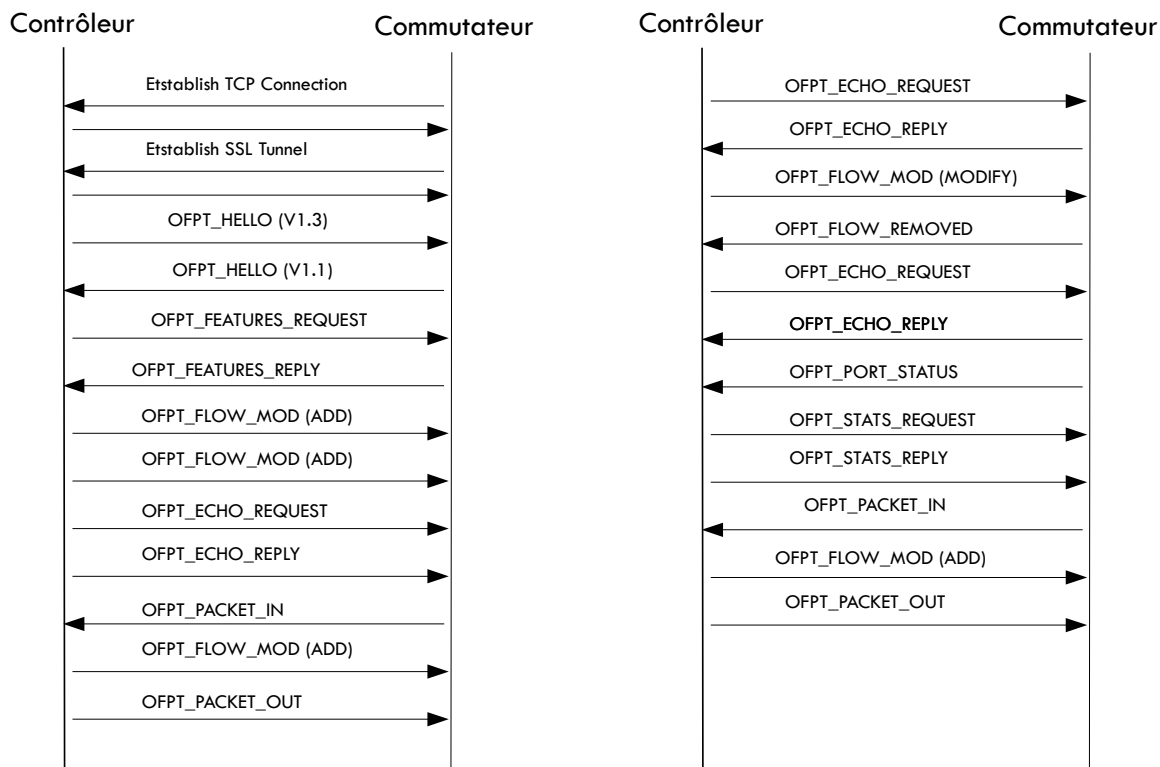


FIGURE 1.7 – Échanges des messages OpenFlow entre le contrôleur et le commutateur

Switch configuration : regroupe un message unidirectionnel et deux paires de messages requête-réponse. `SET_CONFIG` est le message unidirectionnel envoyé par le contrôleur au com-

mutateur pour charger les paramètres de configuration du commutateur. La paire de messages `FEATURES_REQUEST` et `FEATURES_REPLY` est utilisée par le contrôleur pour interroger le commutateur sur ses fonctionnalités de base et optionnelles supportées. Pour obtenir la configuration du commutateur, le contrôleur utilise `GET_CONFIG_REQUEST` et `GET_CONFIG_REPLY`

Command from controller : `PACKET_OUT`, `FLOW_MOD` et `PORT_MOD` sont les messages de *command from controller*. `PACKET_OUT` est utilisé par le contrôleur pour envoyer des paquets de données au commutateur pour leur acheminement via le plan de données. Le message `PORT_MOD` permet au contrôleur de modifier l'état d'un port d'un commutateur OpenFlow. `FLOW_MOD` permet au contrôleur de modifier les entrées des flux existants dans le commutateur.

Statistics : pour recueillir les statistiques du commutateur par le contrôleur, la paire de messages `STATS_REQUEST` et `STATS_REPLY` est utilisée par ce dernier.

Queue configuration : la paire de messages `QUEUE_GET_CONFIG_REQUEST` et `QUEUE_GET_CONFIG_REPLY` permet au contrôleur d'interroger sur la configuration, pour apprendre la configuration des files d'attente et la fourniture aux paquets des informations de niveau de qualité de service souhaité.

Barrier : `BARRIER_REQUEST` est utilisé par le contrôleur pour s'assurer que tous les messages envoyés avant ce message ont été réceptionnés et traités par le commutateur. `BARRIER_REPLY` est le message de confirmation retourné par le commutateur.

ForCES

ForCES est un protocole standard définit par *Internet Engineering Task Force* (IETF) depuis 2004 et qui propose de séparer le contrôle IP de l'acheminement des paquets dans les équipements réseaux. Dans cette approche, les dispositifs d'acheminement des paquets sont modélisés à l'aide de blocs de fonctions logiques appelés *Logical Function Blocks* (LFB), pouvant être composés de manière modulaire pour former des mécanismes de transmission complexes. Chaque LFB fournit une fonctionnalité telle que le routage IP. Les LFB modélisent un dispositif de transfert et coopèrent pour former encore plus de périphériques réseaux complexes. Les éléments de contrôle utilisent le protocole ForCES pour configurer les LFB interconnectés, pour modifier le comportement des dispositifs d'acheminement des paquets. ForCES n'est pas adopté par les acteurs des réseaux en raison du manque de langage clair de communication entre le contrôleur et l'équipement réseau.

OpFlex

OpFlex¹⁷ est un protocole utilisé dans le concept SDN pour permettre à un *Policy Repository* (PR) d'échanger avec un *Policy Element* (PE) pour la réalisation des actions abstraites. Il intègre un système d'échanges permettant à un noeud réseau de fournir des informations essentielles au PR afin de l'assister à prendre des décisions. Standardisé par l'IETF, OpFlex est un protocole open source développé par Cisco. Il est conçu initialement pour permettre l'échange de données d'objets dans le cadre d'un modèle informationnel.

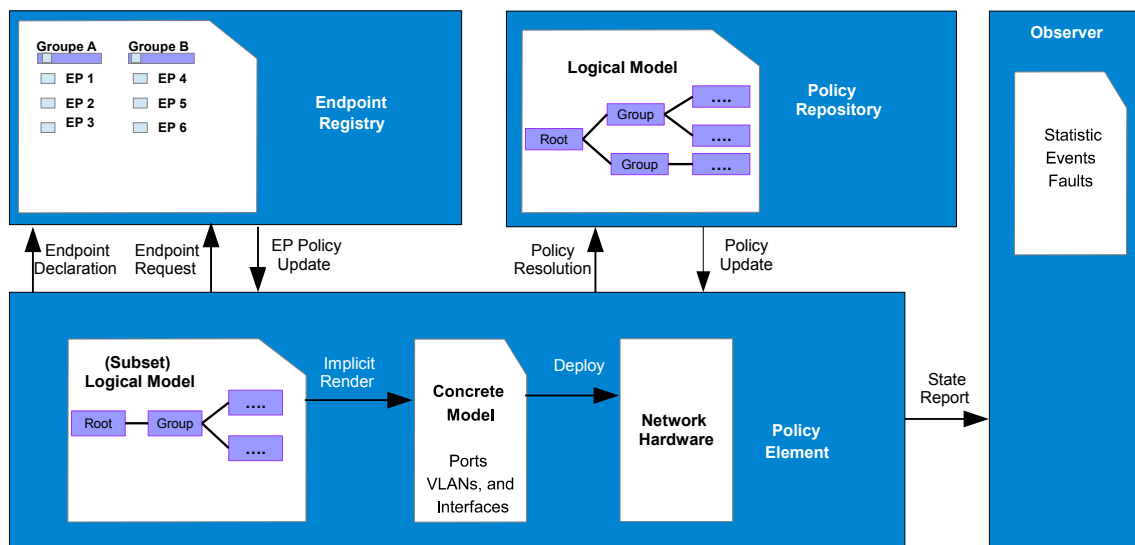


FIGURE 1.8 – Architecture OpFlex [2]

La figure 1.8 ci-dessus schématise les différents éléments d'un modèle OpFlex. Il supporte le format de données *Extensible Markup Language* (XML) et *Java Script Object Notation* (JSON). Le SSL ou le *Transport Layer Security* (TLS) sont aussi utilisés dans OpFlex pour la sécurisation des échanges de données. OpFlex est compatible avec le contrôleur OpenDaylight et *Open vSwitch* (OVS)¹⁸. Autrement dit, OpFlex est basé sur deux éléments clés : le PR et le PE. Les règles et politiques sont définies dans le PR. Puis, elles sont interprétées de manière asynchrone par le PE. Les messages échangés entre le PR et le PE sont de type JSON-RPC over TCP. Le PR sert de base de données des règles uniques qui gère l'ensemble des requêtes provenant de PE à travers un message *Policy Resolution*. Le PR répond au PE avec un message de type *Policy Update*. Le PE est une abstraction logique qui réside sur l'équipement physique ou virtuel. Il est responsable pour l'envoi des messages des équipements réseaux vers le PR en cas de changement d'état (connexion, déconnexion, ?) d'un device. Le PE est aussi chargé de transformer une règle

17. <http://tools.ietf.org/pdf/draft-smith-opflex-00.pdf> (Online : accessed on 17/09/2018)

18. <http://openvswitch.org/> (Online : accessed on 14/09/2018)

abstraite en concrète et ensuite mapper sur l'équipement réseau. Dans l'architecture OpFlex, il y a aussi le *Endpoint Registry* (ER) qui enregistre l'ensemble des informations courantes sur l'état de chaque *Endpoint* (EP). Il reçoit les informations de chaque EP du PE local et partage avec les autres PE du système. Les messages RPC échangés entre le PE et ER sont : *Endpoint Declaration*, *Endpoint Request* dans le sens du PR vers ER et *EndPoint Policy Update* du ER vers PE. Les statistiques, les défauts et événements sont récupérés dans l'Observer. Le type de messages échangés entre le PE vers *Observer* est *State Report*.

1.5.2 La Northbound API

L'interface entre la couche de contrôle et la couche application est la *Northbound API* qui est une des abstractions clés de l'environnement SDN. Elle permet l'échange d'informations entre le contrôleur et les applications s'exécutant sur le réseau. Autrement dit, la *Northbound API* est une API utilisée par l'administrateur réseau pour définir des politiques afin de programmer le plan de contrôle. Il fournit généralement une liste des fonctions réseaux de base associées au fournisseur, qui sont ensuite utilisées pour configurer un équipement spécifique à l'utilisateur, et le contrôleur l'interprète dans un langage que chaque équipement réseau peut comprendre. Il expose le modèle de données d'abstraction du réseau et les fonctionnalités de contrôleur à utiliser par les applications réseaux. La *Northbound API* est utilisée pour faciliter l'innovation et orchestrer efficacement le réseau. La raison d'être de cette interface réside aussi dans le fait que les applications réseaux ont besoin d'extraire les informations de contrôle sur le réseau sous-jacent.

Il existe de nombreuses API pouvant agir sur les équipements réseaux via la couche de contrôle. Comme mentionné dans la section précédente, il existe actuellement plusieurs types de contrôleurs SDN qui offrent une grande variété de *Northbound API*. Les *Northbound API* peuvent être classées principalement en trois catégories, à savoir les API REST (*Representational State Transfer*)¹⁹, les API ad hoc spécialisées et les langages de programmation tels que Procera²⁰, Frenetic²¹...

À ce jour, il n'existe pas de *Northbound API* standard. C'est pourquoi ONF a créé un groupe²² de travail pour élaborer un protocole standard sur cette interface.

L'API REST est la plus utilisée car elle fournit une intégration simple et permet une in-

19. <https://tools.ietf.org/html/rfc6690>(Online : accessed on 17/09/2018)

20. http://python.proceranetworks.com/current/api_reference.html(Online : accessed on 17/09/2018)

21. <https://github.com/frenetic-lang/pyretic/wiki/Dynamic-Policies>(Online : accessed on 17/09/2018)

22. <https://www.opennetworking.org/images/stories/downloads/working-groups/charter-nbi.pdf>(Online : accessed on 17/09/2018)

teraction minimale entre un client et un serveur. Elle n'est pas un protocole mais un concept architectural ayant pour objectif de faciliter la programmation des applications orientées service en utilisant le protocole HTTP. L'API REST est l'oeuvre de Roy Thomas Fielding qui a aussi participé à la définition du protocole HTTP [19]. D'ailleurs, c'est pour cette facilité d'usage que de nombreuses grandes entreprises telles que Facebook et Google l'utilisent pour fournir leurs services [20]. C'est pour ces mêmes raisons que la plupart des contrôleurs SDN aujourd'hui utilisent l'API REST pour fournir des informations réseaux aux applications.

En dehors du contrôleur OpenDaylight, l'implémentation de l'API REST dans la plupart des contrôleurs n'a pas été encore sécurisée. C'est aussi une des raisons qui a motivé le choix du contrôleur dans nos travaux.

1.5.3 La East-Westbound API

L'API *East-Westbound* est l'interface qui permet aux contrôleurs de partager une vue commune du réseau et de coordonner l'exécution des règles et des protocoles. C'est à travers cette interface qu'est gérée la manière dont les contrôleurs du SDN interagissent les uns avec les autres pour partager des informations. D'une manière générale, nous pouvons considérer cette interface comme un canal permettant la communication entre les différents contrôleurs d'un réseau de plusieurs domaines, chacun disposant d'un contrôleur [16]. Ainsi, les contrôleurs peuvent transmettre les informations d'état de leurs réseaux et influencer les décisions de routage. Cette interface *East-Westbound* peut être utilisée pour améliorer la communication entre les différents domaines d'un réseau SDN (intra-domaine et inter-domaine) et partant améliorer l'évolutivité et l'interopérabilité pour le déploiement du SDN. L'API *East-Westbound*, qui est transparente à la configuration et capable de supporter plusieurs types de protocole, est très importante pour les grands réseaux subdivisés en plusieurs, chacun étant contrôlé par un contrôleur SDN. À titre d'exemple, lorsqu'un grand réseau est subdivisé en plusieurs sous-réseaux dont chacun dispose de son propre contrôleur, pour construire une vue globale du réseau, les contrôleurs doivent échanger entre eux à travers le *East-Westbound*. Dans cette conception globale d'un grand réseau, chaque sous-réseau exécute son propre contrôleur et celui-ci dispose uniquement d'une vue globale de son sous-réseau, incluant par exemple la topologie, l'accessibilité, les protocoles réseaux utilisés, l'état du réseau, etc. Pour obtenir une vue globale de l'ensemble du réseau, les opérateurs de réseau utilisent les API *East-Westbound* pour faire communiquer les contrôleurs les uns avec les autres et partager les vues du réseau. Une interface *East-Westbound* est également importante pour automatiser les décisions de réseau afin de limiter les interventions des administrateurs réseaux sur des réseaux opérateurs de grande envergure.

Actuellement, la plupart de ces interfaces sont du domaine ouvert (disponibles sous licence

open source), ce qui est un avantage considérable dans le développement du SDN. Ce qui aide également les opérateurs de réseau à promouvoir les solutions neutres pour leur organisation. ALTO et HyperFlow sont des exemples de protocoles (Eastbound et Westbound) utilisés entre les contrôleurs en cas d'utilisation de plusieurs contrôleurs.

1.6 Conclusion

Dans ce chapitre, nous avons introduit le concept du SDN en expliquant les différentes couches de son architecture ainsi que les API utilisées. Il était question de montrer que cette approche a permis de rendre les réseaux programmables, évolutifs et faciles à innover. Les protocoles standards OpenFlow et OpFlex ont aidé à la concrétisation du SDN. C'est aussi ce qui a nous permis d'expérimenter notre approche de sécurisation des échanges dans un réseau, que nous allons expliquer dans la deuxième partie de ce document.

CHAPITRE 2

L'Internet des Objets

Résumé : Ce chapitre introduit les fonctionnalités des technologies de l'Internet des Objets (IoT), notamment son architecture et ses protocoles. Il est présenté un résumé de certaines approches du SDN adressées aux architectures de la prochaine génération d'Internet. Ces approches ont servi d'inspiration pour le développement de notre architecture de sécurité basée sur le SDN pour l'IoT.

2.1 Introduction

Internet of Things (IoT), et dans un sens plus large, *Internet of Everything* (IoE) est un concept relativement récent. Il est considéré comme une innovation technologique et économique majeure dans l'industrie des nouvelles technologies de l'information et de la communication. L'IoT n'a pas une définition unique mais d'une manière générale, il est défini comme étant une extension de l'Internet actuel à tous les objets pouvant communiquer de manière directe ou indirecte avec des équipements électroniques, eux-mêmes connectés à l'Internet.

L'Union Internationale des Télécommunications définit l'Internet des Objets comme : ”*Une infrastructure mondiale pour la société de l'information, qui permet de disposer de services évolués en interconnectant des objets (physiques ou virtuels) grâce aux technologies de l'information et de la communication interopérables existantes ou en évolution* ”¹.

Les dispositifs IoT sont généralement des noeuds de capteurs, des étiquettes RFID (*Radio-Frequency IDentification*) et des dispositifs de communication sans fil connectés à Internet dans un environnement intelligent. Ces dispositifs très variés (téléphone, montre, réfrigérateur...) sont aujourd'hui très largement utilisés dans la vie quotidienne.

Avec le développement exponentiel de ces objets connectés aux caractéristiques hétérogènes, les réseaux du futur doivent évoluer vers de nouvelles architectures pour s'adapter à l'augmen-

1. https://fr.wikipedia.org/wiki/Internet_des_objets(Online : accessed on 25/09/2018)

tation du trafic et aussi assurer leur sécurité. La sécurité est l'un des problèmes de l'Internet d'aujourd'hui car il existe de plus en plus d'attaques de sécurité encore plus intelligentes à gérer. En plus, les attaques de sécurité pour l'IoT sont plus difficiles à gérer en raison de la capacité minimale de stockage d'énergie, de données et de traitement qui ne sont pas adaptés aux mécanismes de sécurité des réseaux existant basés sur le firewall et IDS/IPS [21]. Le concept de IoT est relativement simple mais les problèmes sont très nombreux car ces dispositifs connectés ne disposent pas suffisamment de capacité pour gérer les communications et les traitements associés aux applications.

La gestion de ces objets hétérogènes nécessite d'autres modèles de gestion des réseaux. C'est pourquoi depuis quelques années, il se développe le concept des réseaux définis par logiciel appelé SDN, que nous avons largement évoqué dans le précédent chapitre. Le SDN est considéré aujourd'hui comme une innovation majeure permettant de gérer plus efficacement les objets connectés et donc, qui peut répondre aux besoins actuels de l'IoT en matière d'hétérogénéité et de flexibilité. En fournissant un contrôle centralisé, le SDN facilite l'optimisation et la configuration d'un réseau de manière efficace et automatisée et assure l'interopérabilité entre les réseaux IoT hétérogènes. Cette centralisation de plan de contrôle peut fournir aussi une architecture sécurisée pour l'IoT que nous allons aborder dans le prochain chapitre.

Dans ce chapitre nous allons introduire le concept de l'IoT, son architecture, ses protocoles ainsi que les solutions proposées pour assurer la gestion des objets connectés.

2.2 Les enjeux de l'IoT

De l'objet qui collecte l'information au réseau qui transmet les données à la plateforme qui agrège et traite les données, les enjeux de l'IoT sont nombreux. Il y a deux tendances dans l'approche même de l'IoT, d'un côté, celle de EPCglobal (une organisation d'adoption et de standardisation à l'échelle mondiale de la technologie du code de produit électronique) qui soutient les solutions RFID comme la base de l'IoT, et de l'autre, IPSO Alliance (*Internet Protocol for Smart Objects*), acteurs actuels de l'Internet et pour qui, si un objet ne communique pas le protocole IP, il n'a pas sa place dans l'IoT. Il y a aussi les solutions M2M (*Machine-to-Machine*) qui s'appuient sur des concepts relativement différents, mais qui se réclament de l'Internet des Objets. La variété de ces éléments du réseau est un grand challenge en matière d'architecture et de sécurité pour l'Internet du futur et pose des questions assez diverses. En plus, le nombre des objets connectés augmentent chaque jour et devrait atteindre les 500 milliards d'ici à 2030 selon les estimations de Cisco ², ce qui entraînera de facto une augmentation du trafic sur les réseaux

2. <https://www.cisco.com/c/dam/en/us/products/collateral/se/internet-of-things/at-a-glance-c45-731471.pdf> (Online : accessed on 25/09/2018)

qui sera estimé en Zeta (10^{21}) octets. Cela suppose qu'il faut plus d'équipements réseaux et d'outils applicatifs pour gérer ces réseaux IoT. C'est pourquoi on constate le développement des réseaux dédiés aux objets connectés et l'existence d'une multitude de protocoles de communication tels que NFC (*Near Field Communication*), RFID, Bluetooth LE (*Low Energy*), SigFox, *Long Rang* (LoRa), réseaux cellulaires, etc avec chacun ses propres caractéristiques et contraintes.

L'avènement de l'IoT a engendré un nouveau marché permettant de créer des emplois et des métiers d'aide à la croissance des entreprises. Selon la *GSM Association* (GSMA), l'IoT est une industrie en plein essor tant sur les aspects matériels que logiciels et devrait rapporter aux opérateurs mobiles un revenu autour de 1200 milliards de dollars à l'horizon 2020.

Il y a certainement de nombreux enjeux socio-économiques qui touchent tous les domaines de la vie quotidienne (santé, éducation, agriculture...) mais nous nous focaliserons sur les enjeux techniques, notamment d'architecture et de sécurité. C'est ce que nous allons évoquer dans la prochaine section.

2.3 Architecture de l'IoT

L'objectif principal de l'IoT est de connecter des milliards d'objets, voir même plus, sur Internet. Pour cela, il faut nécessairement une architecture pour représenter, organiser et structurer l'IoT de manière à lui permettre de fonctionner efficacement. En particulier, la nature hétérogène de l'IoT nécessite l'application d'architectures (matériel et logiciel) et de processus capables de prendre en charge ces objets connectés, leurs services et les flux de données qu'ils génèrent.

Actuellement, il n'y a pas un consensus unique sur l'architecture pour l'IoT, qui soit accepté par toute la communauté scientifique [22]. Différentes architectures ont été proposées par des chercheurs [23, 4, 24]. Dans le même contexte, de nombreux modèles d'architecture sont en cours de développement, comme le modèle NIST Smart Grid (*National Institute of Standards and Technology*), le modèle ITU-T (*International Telecommunication Union - Telecom*), le modèle M2M de l'ETSI (*European Telecommunications Standards Institute*) ou la référence architecturale du projet IoT-A de l'Union Européenne et les autres travaux liés à l'IETF, W3C etc.

La conception architecturale pour l'IoT standardisé par l'IETF, et communément admise, est composée de six couches [3] à savoir : application, transport, adaptation, MAC et physique. La figure 3.1 ci-dessous montre les différentes couches de l'architecture IoT ainsi que les différents protocoles utilisés. Les travaux de normalisation de l'IETF sur l'IoT ont également conduit à la mise en place des protocoles de communication légers adaptés aux objets connectés.

Ces protocoles sont entre autres, le CoAP (*Constrained Application Protocol*)³, le 6LoWPAN (*IPv6 over Low-Power Wireless Personal Area Network*)⁴, RPL (*IPv6 Routing Protocol for Low-Power and Lossy Networks*)⁵ ...

Modèle TCP/IP	Modèle IETF IoT	Couches
HTTP, FTP, DNS, SSH..	IETF CoAP	Application
TCP, UDP	UDP	Transport
IPv4, IPv6	IPv6, IETF RPL	Réseau
N/A	IETF 6 LoWPAN	Adaptation
Accès réseau	IEEE 802.15.4 MAC	Liaison
	IEEE 802.15.4 PHY	Physique

FIGURE 2.1 – Comparaison entre le modèle TCP/IP et l'architecture IoT de l'IETF[3].

L'architecture IoT la plus couramment utilisée pour les solutions SDN, et comme le montre la figure 3.2, comprend trois couches [25, 26, 27] : la couche de perception, la couche réseau et la couche application.

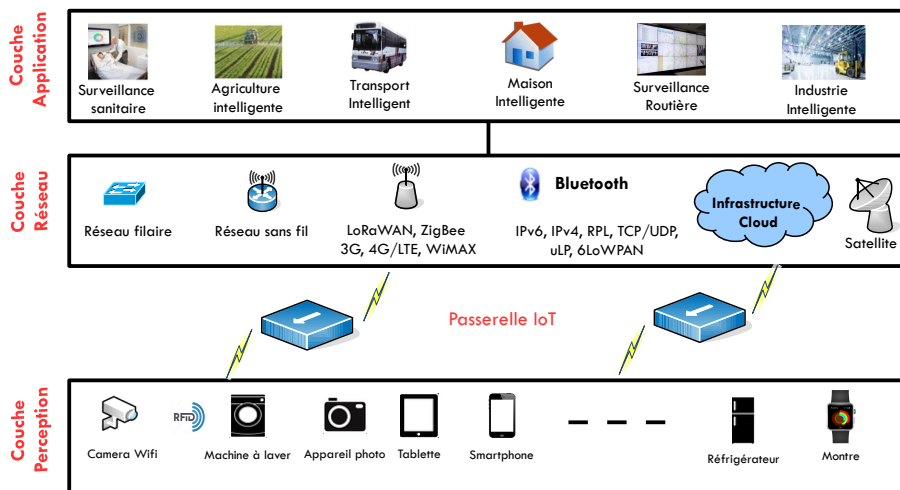


FIGURE 2.2 – Architecture pour l'Internet des Objets [4].

2.3.1 La couche perception

Cette couche est constituée d'objets physiques composés de capteurs, de dispositifs RFID, de mobiles etc. Le rôle principal de la couche perception est essentiellement d'identifier et de collecter les informations spécifiques aux objets [27]. Selon le type d'objets physiques, les informations peuvent être des informations de type localisation, humidité, température, qualité

3. <https://tools.ietf.org/pdf/rfc7252.pdf> (Online : accessed on 29/05/2019)

4. <https://tools.ietf.org/pdf/rfc6282.pdf> (Online : accessed on 29/05/2019)

5. <https://tools.ietf.org/pdf/rfc6550.pdf> (Online : accessed on 29/05/2019)

de l'air, etc. Les informations ainsi collectées sont ensuite transmises à la couche réseau pour leurs transmissions via des canaux sécurisés. Des mécanismes plug-and-play normalisés doivent être utilisés pour configurer les objets physiques hétérogènes [26, 28]. La grande quantité de données générées par l'IoT provient de cette couche. Le coût des objets connectés et leur autonomie sont les contraintes de cette couche, en dehors des données massives qu'ils génèrent.

2.3.2 La couche réseau

La couche réseau est responsable de la transmission des données vers la couche perception et vers la couche application [26]. Pour assurer le transfert de données, la couche réseau utilise différents types de technologies de mise en réseau et de protocoles. Ces solutions de connexions réseaux sont, d'une part, les technologies déjà largement déployées, capables de transporter de gros volumes d'informations, mais gourmandes en énergie, telles que Bluetooth, WiFi, 3G, 4G et LTE, et d'autre part, les technologies conçues spécifiquement pour le monde de l'IoT mais encore moins déployées comme les ZigBee, LoRa, Sigfox etc. Les réseaux de type SigFox et LoRa permettent de connecter des objets sur une dizaine de kilomètres mais avec des débits très faibles de l'ordre de quelques octets. La WiFi et le Bluetooth ont une portée limitée mais très utile dans un environnement restreint et pourvue de sources d'énergie (une usine par exemple). L'avantage de la 2G/3G/4G est sa disponibilité. Elle peut servir par exemple pour les véhicules connectés.

Les passerelles font partie de cette couche réseau et sont responsables de la transmission des données. Comme l'indique la figure 2.1, la passerelle IoT est l'un des composants le plus important de l'architecture IoT et sert de pont entre la couche de perception et la couche réseau [29]. Il doit être suffisamment flexible pour gérer les ressources disponibles et connecter le système IoT local au reste du système IoT global. La passerelle IoT se heurte à de nombreux problèmes tels que l'hétérogénéité des différents protocoles et composants de l'IoT. Les contraintes liées à cette couche réseau sont la couverture, le coût et la fiabilité.

2.3.3 La couche application

La couche application permet l'interaction directe avec les utilisateurs finaux. Les applications peuvent être déployées pour différents domaines de la vie quotidienne tels que la surveillance sanitaire, l'agriculture intelligente, le transport intelligent, les maisons intelligentes, la surveillance routière, l'industrie intelligente etc. [25, 30]. Elle comprend également des infrastructures serveur et cloud partageant du contenu et fournissant des services en temps réel. Le traitement des données et la fourniture de services sont deux des fonctions les plus importantes de cette couche. La couche application fournit une gestion globale du système IoT. Il reçoit des

informations de la couche réseau qui permet aux développeurs de créer diverses applications en utilisant une abstraction et des interfaces ouvertes et de haut niveau. Et c'est dans cette couche que toutes les décisions de contrôle, de sécurité et de gestion des applications sont prises.

2.4 Les protocoles de l'IoT

Les technologies de communication IoT connectent des objets hétérogènes pour fournir des services intelligents spécifiques. En règle générale, les noeuds IoT doivent fonctionner avec une faible consommation en présence de liaisons de communication avec perte et bruit. Ces liaisons fonctionnent avec des technologies, où les messages échangés entre différents dispositifs peuvent se faire grâce à des protocoles avec des interactions en temps réel. Par conséquent, une architecture de référence unique ne peut pas être utilisée comme modèle pour tous les cas d'usage. Si un modèle de référence est identifié, il est probable que plusieurs architectures coexisteront dans l'IoT. Tous les dispositifs IoT interconnectés utilisent différents protocoles pour permettre l'interconnexion entre l'environnement physique et numérique, ce qui permet de collecter et de traiter les données obtenues. Dans les environnements IoT plusieurs types de protocoles de communication peuvent être utilisés [23]. Les principaux protocoles qui permettent l'interconnexion des objets sur les réseaux IoT, que nous avons déjà évoqués dans la précédente section, sont détaillés sur le tableau ci-dessous :

Protocole	Standard	Fréquences	Portée	Débit
Wifi	802.11	2.4Ghz, 5Ghz	50-125m	150Mbps-1.3 Gbps
Bluetooth LE	4.2-5.0	2.4Ghz	50-240m	1-50Mbps
LoRaWAN	LoRaWAN	variées	2-15km	0.3-50kbps
ZigBee	IEEE802.15.4	2.4Ghz	10-100m	250kbps
Z-Wave	ITU-T G.9959	900MHz	30m	9.6/40/100kbit/s
Cellular	GSM/GPRS/ 2G/3G/4G /LTE- M/ Clean Slate	900/1800/ 1900/2100MHz	35km GSM - 200km HSPA	35-170kps (GPRS), 120-384kps (EDGE), 384kps-2Mbps(UMTS), 600kbps-10Mbps (HSPA), 1-10Mbps (LTE)
NFC	ISO/IEC 1800-3	13.56MHz	10cm	100-420kbps
Sigfox	Sigfox	900MHz	10-50km	10-1000bps
Neul	Neul	900MHz (ISM), 458MHz (UK), 470- 790MHz (White space)	10km	100kbps
LowPAN	RFC6282	2.4Ghz, ZigBee, low- power (sub-1GHz)	30-100m	20/40/250kbps.
DASH7	ISO 18000-7	433 MHz	1000m	200kbps
LTE-M			15km	15kbps-1Mbps
Clean Slate IoT			15km	15kbps-1Mbps

TABLE 2.1 – Technologies de Communications pour l'Internet des Objets [5]

2.5 Les architectures SDN pour l'IoT

Avec le développement exponentiel des objets connectés, assurer la gestion de ces dispositifs aux caractéristiques très variées est une tâche difficile pour les administrateurs des réseaux. C'est pourquoi, depuis ces dernières années, les chercheurs dans ce domaine particulier ont concentré leurs travaux sur les nouvelles approches d'architecture réseau, pour mieux organiser et gérer l'IoT. C'est ainsi qu'est née la technologie SDN qui a fait l'objet de notre précédent chapitre. Contrairement aux réseaux classiques, qui ne sont pas conçus pour supporter un passage à l'échelle avec un grand volume de trafic et une forte mobilité, les architectures de type SDN permettent de mieux gérer l'hétérogénéité des réseaux IoT dans le futur. L'émergence des architectures SDN pour IoT est aussi une grande opportunité pour gérer de manière simple la sécurité des objets connectés. Dans cette section, nous présentons un aperçu de certains de ces travaux sur les architectures de type SDN pour l'IoT.

Dans [31], les auteurs ont largement discuté des différentes opportunités telles que la simplification de déploiement et défis de sécurité du SDN dans le contexte de l'IoT et ont montré que le SDN aura un impact majeur sur la conception et la gestion des réseaux IoT afin de les rendre plus performant. Les auteurs ont également discuté des récents développements des réseaux sans fil et optiques adaptés au SDN et à l'IoT.

Martinez et al. ont proposé dans [32], une architecture basée sur le SDN pour l'IoT. Dans cette architecture les dispositifs IoT se connectent entre eux à l'aide du protocole IPv6. Elle permet également de simplifier les opérations de gestion et de contrôle des objets hétérogènes, par l'ajout d'un contrôleur IoT supplémentaire au contrôleur SDN. Par exemple, si un objet "A" veut communiquer avec un autre objet "B" se trouvant dans le même réseau ou dans un réseau différent, le contrôleur IoT récupère dans un premier temps les informations nécessaires à la définition des règles de communication à partir de l'agent de l'objet demandé qui se trouve à l'intérieur de cet objet et trouve l'objet demandé en l'occurrence l'objet "B" dans ce cas. Ensuite, un algorithme de routage est appliqué pour calculer le chemin d'accès à cet objet. Après avoir défini et établi les règles de transfert, le contrôleur IoT envoie ces règles au contrôleur SDN qui les transmet aux équipements réseaux pour acheminement. Ainsi, même si les objets "A" et "B" ont des protocoles incompatibles, les dispositifs de transfert les traduisent de manière à être compréhensible par le destinataire. Ce travail a permis à des objets hétérogènes d'un réseau de communiquer entre eux en utilisant le SDN et le protocole IPv6.

MINA (*Multinetwork Information Architecture*) est un framework SDN proposé dans [33] pour faciliter l'accès au réseau d'accès des dispositifs IoT, en utilisant un contrôleur pour traduire les exigences de service en exigences de réseau de niveau inférieur respectant certaines contraintes de qualité de service (QoS).

Dans [34], les auteurs ont développé un framework appelé UbiFlow utilisant des contrôleurs SDN distribués pour contrôler les flux et gérer la mobilité dans un réseau hétérogène. Cette solution basée sur le SDN divise le réseau IoT en plusieurs clusters. Chaque cluster représente une partition géographique donnée, gérée par son propre contrôleur. La gestion des flux entre les différents contrôleurs distribués s'effectue avec un processus de *handover*, dans le but de l'équilibrage de charge. Les contrôleurs répartis collaborent pour permettre également la gestion de la mobilité. Chaque dispositif IoT est attribué au contrôleur SDN à l'aide d'un algorithme de hachage distribué. Dans le même principe, un autre framework a été proposé dans [35] pour gérer les dispositifs IoT en exploitant les avantages du SDN. Le framework proposé par ces auteurs est principalement axé sur la gestion des messages M2M. Il est composé de quatre éléments : le contrôleur, les noeuds M2M, la passerelle et les noeuds généraux. Le contrôleur est responsable de l'apprentissage, de la construction et de la redéfinition du trafic réseau. Les noeuds M2M sont principalement des périphériques de support et des réseaux utilisant les communications M2M. Les périphériques qui ne supportent pas les communications M2M sont connectés à travers une passerelle.

Dans [36], un *framework* appelé SDIoT (*Software Defined IoT*) pour l'Internet des Objets a été proposé. Le SDIoT exploite le SDN afin de réduire la complexité des architectures IoT existantes et en même temps permet de stocker et de sécuriser les données produites à partir des objets connectés. Le *framework* est composé d'un contrôleur IoT, d'un contrôleur de stockage et d'un contrôleur de sécurité. Deux composantes, appelées respectivement *Sensor Network Cluster*(SNC) et *Database Pool Cluster* (DPC), s'occupent de l'organisation et de la collecte des données. Dans chaque cluster de réseau, une passerelle IoT permet d'assurer l'échange des données entre les applications et les dispositifs IoT. Les informations collectées sont stockées et classées par type dans le DPC. Le processus d'authentification et d'autorisation des dispositifs IoT est fait par le contrôleur de sécurité. Une fois le processus d'authentification terminé, le contrôleur IoT applique les règles de communication et sélectionne le dispositif cible. Le contrôleur IoT échange avec le contrôleur SDN pour calculer le chemin d'accès aux périphériques de destination et prend les décisions d'acheminement et de routage.

Un modèle de programmation pour les applications IoT sur les plates-formes *Cloud*, appelé PatRICIA (*P*rogramming *I*ntent-based *C*loud-scale *I*ot *A*pplications), a été proposé par Nastic et al dans [37]. Cette solution offre aux développeurs un moyen simple d'implémenter différentes applications IoT sur le *cloud* sans avoir besoin de connaître les informations détaillées relatives aux dispositifs IoT. Le problème de ce *framework* est qu'il n'intègre pas une solution programmable de monitoring et de gestion pour contrôler les dispositifs IoT.

Les dispositifs IoT sont hétérogènes et utilisent plusieurs technologies pour échanger entre eux à travers des middlewares, pour réduire la perte des messages échangés entre les appli-

cations et ces objets. Mais la question de l'interopérabilité reste un problème majeur à gérer afin d'améliorer les performances d'un réseau IoT. Pour traiter cette question, un système d'exploitation réseau peut jouer un rôle important dans la gestion de l'interopérabilité dans les systèmes hétérogènes complexes. Les noeuds de capteurs et les actionneurs sont considérés comme des éléments constitutifs d'un réseau IoT. Ces dispositifs minuscules ont des contraintes de ressources en énergie, de capacités de stockage, de puissances de traitement, de routages, etc. Cependant, les systèmes d'exploitation existants conçus pour les réseaux de capteurs ne sont pas capables de gérer l'interopérabilité à grande échelle. Pour cette raison, de nombreux systèmes d'exploitation, tels que Contiki OS [38], RIOT OS [39], Tiny OS [40], Lite OS [41], etc., ont été développés pour les réseaux de capteurs basés sur les réseaux IoT. Cependant, ces systèmes d'exploitation spécifiques à certaines applications, manquent de souplesse et de dynamisme.

Il n'y a pas de système d'exploitation standard pour gérer l'intégration de l'IoT basé sur le SDN à ce jour, même si des efforts ont été faits pour développer un système d'exploitation pour les objets connectés. Les systèmes d'exploitation disponibles sur le marché, tels que NOX, Maestro, OpenDaylight, etc, ne fonctionnent correctement que pour les réseaux SDN câblés. Ces contrôleurs, ou systèmes d'exploitation ne prennent pas en charge les caractéristiques des dispositifs IoT telles que les contraintes d'énergie, de traitement, d'agrégation de données, etc.

Un concept a été proposé dans [42] afin de faciliter la reprogrammation et la réattribution des tâches dans les réseaux de capteurs. Appelée SOF (*Sensor OpenFlow*), cette approche est basée sur une architecture à trois couches : une couche application, une couche de contrôle et une couche de plan de données. La couche application comprend toutes les applications nécessaires à la gestion des requêtes et au traitement des données, tandis que la couche de contrôle est constituée d'un module de ré-configuration des capteurs et d'un module de contrôle des requêtes. Le plan de données s'occupe de l'acheminement des flux de capteurs dans l'ordre défini par le contrôleur. Pour améliorer cette approche, dont la gestion des flux et la surcharge créée par le trafic de contrôle pourraient dégrader les performances, une solution complémentaire a été proposée pour surmonter ces limites. La solution proposée par Galluccio et al. dans ce cadre [43] est le SDN-WISE. SDN-WISE, un système d'exploitation réseau basé sur le SDN pour la gestion complète des réseaux de capteurs (agrégation des données, cycle de vie des capteurs...).

2.6 Conclusion

Dans ce chapitre, nous avons introduit l'Internet des Objets, abordé de manière succincte ses enjeux, son architecture et les technologies utilisées pour les échanges entre les dispositifs IoT pour mieux comprendre l'approche globale. Nous avons également fait un état de l'art sur

les architectures IoT basées sur le SDN. Ces techniques proposées dans la littérature permettent de faciliter la gestion des objets connectés en prenant en compte le caractère hétérogène de ses composants IoT.

CHAPITRE 3

La sécurité des réseaux avec le SDN

Résumé : Dans ce chapitre, nous allons aborder de manière succincte les techniques traditionnelles de sécurité, telles que firewall, IDS/IPS et NAC, pour rappeler les mécanismes existants de sécurisation des échanges dans un réseau. Ensuite, nous allons faire un état de l'art détaillé sur la sécurité des réseaux traditionnels et IoT avec le concept du SDN.

3.1 Introduction

Pendant longtemps, la sécurité des données impliquait uniquement le chiffrement des données pour les chercheurs, mais avec l'arrivée de l'Internet à la fin des années 80, la nécessité d'avoir des outils plus performants pour sécuriser les données s'est faite sentir. Pour résoudre ces problèmes, les firewalls et IDS/IPS ont été introduits pour la protection des données, services et applications dans les réseaux Internet actuels, dont les menaces et attaques sont devenues trop importantes. Malgré l'usage généralisé des firewalls et IDS/IPS, ces mécanismes de sécurisation des réseaux ont aussi leur limite, même si les récentes versions permettent d'avoir des fonctionnalités beaucoup plus évoluées.

L'arrivée des objets connectés a bouleversé la manière de sécuriser les réseaux, qui deviennent de plus en plus complexes à gérer avec les mécanismes traditionnels tels que firewall et IDS/IPS. De plus, l'augmentation du trafic induit par l'IoT impacte fortement les firewalls et IDS/IPS sur leurs capacités de traitement des menaces, d'où les limites aussi dans leurs fonctionnalités. De tout ce qui précède, il sera difficile de contrôler les réseaux IoT.

La sécurité des réseaux demande des efforts continus d'anticipation afin de s'adapter à la modification des tendances en matière de menace pour prévenir, minimiser ou empêcher de futures attaques. Dans n'importe quel type de réseau, la sécurité doit être cohérente, dynamique, évolutive et de bout en bout. Rendre la sécurité dynamique et intelligente permet de mieux identifier les menaces par des analyses prédictives et répondre en temps réel en cas de nécessité.

Le monde de la recherche accorde de plus en plus d'attention à la sécurité des réseaux, du fait de l'évolution exponentielle de l'Internet en général et de l'Internet des Objets en particulier. L'essor de ce dernier type d'Internet se confirme de jour en jour, entraînant dans son sillage des problèmes de sécurité encore plus complexes à gérer.

Les thématiques de recherche basées sur l'utilisation des techniques traditionnelles de sécurité sont nombreuses pour protéger les réseaux des différents types d'attaque, mais les techniques utilisant le paradigme du SDN pour améliorer la sécurité sont nouvelles, prometteuses et très largement discutées dans la littérature [44, 45].

Le principal objectif de ce chapitre est de voir un bref aperçu des techniques existantes de sécurité dans les réseaux traditionnels et ensuite faire un état de l'art sur la sécurité des réseaux IoT avec SDN.

3.2 Mécanismes de sécurité des réseaux traditionnels

Dans les réseaux traditionnels, la sécurité est principalement basée sur l'utilisation de firewall, de l'IDS/IPS, NAC ... Dans cette section, nous allons voir un aperçu de ces solutions de sécurisation des réseaux.

3.2.1 Protection par firewall

Un firewall est un système pour protéger un réseau ou un ordinateur des menaces provenant d'un réseau tier, comme l'indique la figure 3.1. Installé le plus souvent sur un équipement réseau et placé à l'entrée d'un réseau, le firewall assure le filtrage des paquets entrants et sortants d'un réseau, selon les règles prédéfinies par l'administrateur réseau. Les filtres permettent d'interdire ou d'accepter tous les paquets sauf ceux autorisés par la liste de l'administrateur [46]. Ils permettent également de reconnaître les caractéristiques de l'ensemble des éléments de l'entête d'un paquet IP, à savoir le numéro de port, l'adresse IP source, l'adresse IP destination etc. Pour le filtrage applicatif, la discrimination se fait avec le numéro de port. L'ensemble des numéros de port interdits ou acceptés sont répertoriés dans une table dans un firewall. L'inconvénient d'un réseau protégé par un firewall est qu'il n'est protégé que des menaces externes et sur la base des filtres installés par l'administrateur. Ce qui fait que la protection d'un réseau par un firewall n'est pas absolue et n'offre une meilleure efficacité que si l'ensemble des communications entrantes et sortantes passent par le firewall. L'introduction par exemple d'un support de stockage ou un modem avec accès Internet sur une machine du réseau peut rendre le firewall inefficace.

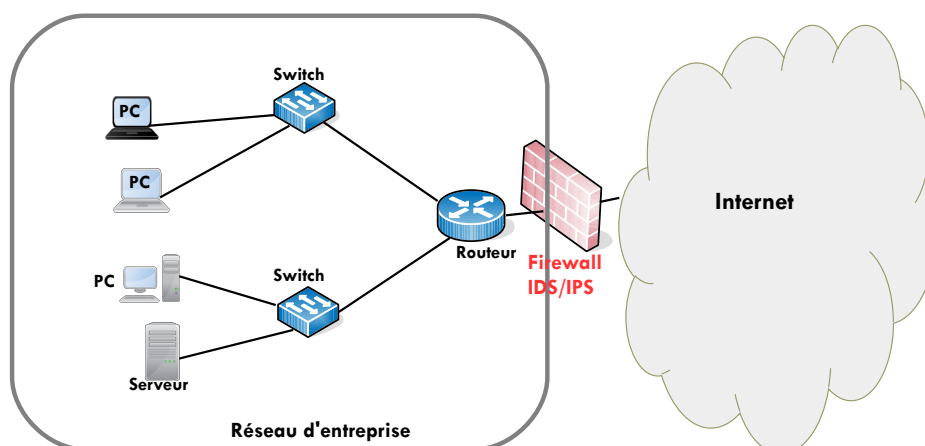


FIGURE 3.1 – Exemple d'un réseau protégé par un firewall

3.2.2 Protection par IDS/IPS

Les attaques par intrusion sur les réseaux sont nombreuses, complexes, intelligentes et de plus en plus sophistiquées. Elles se produisent sous plusieurs formes et dans toutes les couches du modèle TCP/IP. C'est pourquoi, il faut plusieurs mécanismes complémentaires au firewall pour sécuriser les réseaux et en temps réel. L'IDS et l'IPS sont des techniques permettant d'analyser et de détecter les intrusions et éventuellement de les prévenir [47, 48]. Détecter une intrusion consiste à mettre en oeuvre des moyens permettant de mettre en évidence d'éventuelles menaces. C'est à dire en surveillant l'activité du système (réseau, machine) afin d'en recueillir les événements qui se passent sur le réseau ou sur la machine et ensuite en analysant ces événements pour trouver des signes suspects s'il en existe. L'IDS a pour objectif d'automatiser les phases de collecte et d'analyse de l'information d'un système. Il est un sniffer couplé avec un moteur qui analyse le trafic selon des règles spécifiques qui décrivent le trafic à signaler. L'IDS est capable d'analyser le trafic de la couche réseau (IP, ICMP), de la couche transport (TCP, UDP) et de la couche application (http et ftp).

L'IDS examine d'une manière générale les actions qui violent la sécurité du réseau en le notifiant à l'administrateur en temps réel, mais il n'agit pas pour arrêter l'intrusion. Seul l'IPS, qui est une version évoluée de l'IDS, permet d'agir en plus de détecter l'intrusion pour prévenir d'une possible attaque. Un IDS est placé à l'entrée d'un réseau, soit en série, soit en sonde, pour analyser le trafic. L'architecture en série permet de vérifier le trafic en intégralité d'où la nécessité d'avoir aussi des ressources suffisantes en stockage et en traitement des flux.

Il y a deux méthodes majeures de détection d'intrusion [49, 50, 51] : la méthode de détection des anomalies par rapport à un profil de trafic habituel, basée sur une comparaison du trafic et la méthode à base de signature d'attaques connues en vérifiant les entêtes et la charge utile

des paquets.

Détection par anomalie : cette méthode est basée sur la détection des anomalies par rapport à un profil de trafic habituel. Pour la mise en oeuvre, les IDS vont découvrir le fonctionnement normal du système à surveiller avec une phase d'apprentissage. À partir de ces comportements, le système peut déclencher une alerte lorsque des événements hors profil se produisent. L'avantage de cette méthode est qu'elle détecte les nouvelles attaques sans qu'il n'y ait eu une mise à jour du système. Le problème est qu'elle génère beaucoup de fausses alertes si la forme du trafic évolue.

Détection par signature : elle est basée sur une comparaison entre le trafic et une base de signature, en regardant les entêtes et la charge utile des paquets. L'avantage de cette méthode est qu'elle génère beaucoup moins de fausses alertes. Le problème de cette méthode est qu'elle ne détecte pas de nouvelles menaces sans une mise à jour régulière de sa base de signature.

L'IDS est appelé *Network IDS* s'il est exécuté sur un firewall, sur un équipement spécial placé à l'entrée d'un réseau en série ou en sonde. Le principe est de contrôler l'activité réseau (contenu, volume etc.) en analysant et en interprétant les paquets qui circulent sur le réseau. L'équipement doit avoir une grande capacité de traitement des paquets car il inspecte forcément tous les paquets en temps réel s'il est placé en série. On appelle *Host IDS* s'il est installé sur un serveur pour contrôler l'activité du système (logs, fichiers, processus etc.). *Network IDS* et *Host IDS* sont complémentaires. Un IDS réseau est beaucoup plus large en terme de champs d'action que le *Host IDS*. Une *Application IDS* est une application particulière sur une machine.

Les IDS permettent de mettre en évidence une éventuelle intrusion dans un réseau. Après détection d'une menace l'IDS doit recueillir le maximum d'informations (cible, source, procédé...), puis archiver et tracer la menace. Il doit aussi préparer la réponse à une attaque en la mettant en quarantaine par exemple.

Il y a des IDS propriétaires (Cisco, Symantec...) et libres comme Snort¹. Il faut noter aussi que les multiples technologies de détection et de prévention d'intrusion sont complémentaires et fonctionnent pour différents types d'environnements, certaines donnant de meilleurs résultats que d'autres.

3.2.3 Protection par Network Access Control (NAC)

Le NAC est une technique utilisée par Cisco et autres équipementiers, permettant de soumettre l'accès à un réseau en passant par un protocole d'identification de l'utilisateur [52]. Le dispositif est généralement composé d'un serveur pour la gestion de la politique de sécurité, d'un protocole de contrôle sur les points d'accès (wifi, routeur, switch), d'une application sur

1. <https://www.snort.org/> (Online : accessed on 10/10/2018)

les postes des utilisateurs et d'un système d'administration qui fournit les outils de contrôle et de reporting. L'objectif est de maîtriser le comportement d'une machine dans un réseau en surveillant l'état de son antivirus, les mises à jour de son système d'exploitation et les droits d'accès pour minimiser l'impact des programmes malveillants. Le champ d'action du NAC est la sécurisation du périmètre intérieur d'un réseau d'entreprise [53]. À titre d'exemple, le fonctionnement d'un NAC se fait de la manière suivante : lorsqu'une machine tente de se connecter au réseau, dans un premier temps, le point d'accès interroge l'application installée sur la machine et transmet la demande d'accès avec les éléments sur son état de sécurité au serveur de politique de sécurité, si la politique est conforme, la machine reçoit son adresse et accède au réseau, dans le cas contraire, la machine sera mise en quarantaine. En résumé, nous pouvons dire que le NAC permet à un réseau d'identifier et de prévenir les menaces par un système intégré au réseau lui-même. Le NAC est une solution qui a apporté un coup de pouce à la sécurité interne des réseaux, mais le manque de standard a fait qu'il n'existe pas d'interopérabilité, rendant les entreprises pessimistes à son large déploiement.

3.3 Avantages du SDN par rapport aux réseaux traditionnels

Dans les réseaux traditionnels, l'administration du réseau est complexe et sujette aux erreurs du fait que le plan de contrôle et le plan de données sont ensemble sur l'équipement réseau (commutateur, routeur...). Cela rend très difficile l'introduction des nouvelles fonctionnalités de gestion et de sécurité, telles que firewall, IDS/IPS et NAC. De plus, le déploiement des nouveaux protocoles est long et fastidieux du fait du caractère propriétaire des équipements réseaux traditionnels. L'arrivée du concept de SDN a simplifié l'administration et la sécurisation des réseaux, en séparant le plan de contrôle du plan de données. Avec cette approche, le plan de contrôle est externalisé et mis sur un équipement externe appelé contrôleur SDN. Ce dernier est un contrôleur logiciel responsable d'exécuter les fonctionnalités du plan de contrôle. Le plan de données est composé des équipements physiques, tels que commutateurs et routeurs, qui s'occupent de l'acheminement des paquets. Le contrôleur centralisé du SDN permet d'introduire des nouvelles fonctionnalités sans trop de peine. Un autre grand avantage du SDN est qu'il offre la possibilité de tester les fonctionnalités réseaux avant de les intégrer, sans affecter le fonctionnement du réseau. Ce qui permet de minimiser les risques de perturbation du réseau. Pour concrétiser le SDN, le protocole OpenFlow a été introduit pour gérer les échanges entre le contrôleur et les équipements réseaux et aussi gérer le trafic dans les équipements réseaux. SDN a la capacité de simplifier la gestion des grands réseaux avec cohérence et évolutivité ce

qui lui a donné un crédit particulier dans le monde industriel.

3.4 Défis de sécurité des architectures SDN

En général, les menaces de sécurité sur les architectures de type SDN sont similaires aux réseaux traditionnels. Cependant, le profil de ces menaces change avec le SDN. Par exemple, une attaque de déni de service sur un contrôleur centralisé qui gère un grand réseau constitué de plusieurs équipements réseaux (routeurs, commutateurs...) est plus destructrice qu'une attaque ciblée contre un routeur. Un contrôleur SDN usurpé pourrait permettre à un hacker de prendre le contrôle de tout un réseau, tandis qu'un routeur usurpé ne pourrait nuire qu'au bon fonctionnement du trafic acheminé à travers ce routeur.

Le SDN est confronté à ces nouveaux challenges de sécurité, notamment sur la question de comment sécuriser l'architecture SDN elle-même. C'est pourquoi Kreutz et al. ont évoqué dans [54] la nécessité d'intégrer la sécurité et la fiabilité dans la conception même du réseau SDN avant toute utilisation. Étant basée sur une architecture en trois couches et des interfaces de programmation, la sécurité du SDN doit être assurée à tous ces niveaux, ce qui constitue plusieurs défis à relever. La liste des défis de sécurité du SDN devrait s'allonger avec le déploiement progressif de cette technologie.

3.5 Historique de la sécurité des réseaux programmables

La sécurité est une tâche difficile, dû à la complexité des équipements réseaux par leurs caractères propriétaires, ainsi que les périmètres à sécuriser de plus en plus larges. L'architecture des réseaux actuels, qui utilisent ces équipements pour leur fonctionnement, rencontre des problèmes de sécurité à cause du manque de facilité d'innovation. C'est pourquoi, par le passé, plusieurs initiatives tendant à concevoir des nouveaux types d'architectures, afin de rendre moins complexe et évolutive la gestion de la sécurité, ont émergé. Ces initiatives ont été concrétisées par la proposition des architectures de sécurité programmables :

Active Networking : Active Networking est une approche d'architecture réseau apparue dans les années 94-95 à l'initiative de la *Defense Advanced Research Projects Agency* (DARPA) pour faciliter le déploiement des services réseaux. Elle a été proposée pour permettre la programmabilité d'un équipement réseau, tels que des commutateurs et routeurs [55]. Active Networking proposait d'inclure sur les équipements réseaux une API afin de programmer le plan de données. L'un des objectifs d'Active Networking est aussi la sécurisation des échanges dans un réseau, c'est pourquoi plusieurs travaux en lien avec la sécurité ont suivi et ont été proposés notamment dans [56] et [57].

Secure Architecture for the Networked Enterprise (SANE) : SANE est une architecture pour sécuriser les réseaux d'entreprise avec, pour idée principale, de simplifier la complexité des systèmes de sécurité utilisés dans les réseaux d'entreprise. Il a pour objectif de faciliter l'intégration des politiques de sécurité, l'indépendance de la topologie du réseau des équipements, la protection des services réseaux des accès non autorisés et la centralisation et la définition des règles de sécurité [58]. L'architecture du SANE est composée d'un *Domaine Controller* (DC) qui réalise principalement trois fonctions : (1) l'authentification des utilisateurs, des devices et des commutateurs, et le maintien d'une clé symétrique pour chaque élément afin de sécuriser les communications. (2) Annoncer et contrôler les accès pour les services disponibles. (3) Contrôler toutes les connexions avec le réseau SANE. Globalement, SANE permet de sécuriser un réseau d'entreprise en simplifiant l'intégration des règles de sécurité sur le réseau de manière centralisée. Ainsi, la complexité de la gestion du réseau est simplifiée et le nombre de composants à configurer est diminué.

Ethane : Ethane est une extension de SANE, basée sur le principe de déploiement progressif des services dans un réseau de manière centralisée. Le projet Ethane [59] avait pour objectif de centraliser le contrôle d'un réseau par le découplage de l'infrastructure d'un réseau de contrôle, c'est-à-dire que les équipements d'un réseau d'entreprise agissent selon les instructions d'un contrôleur centralisé. L'architecture Ethane est composée d'un contrôleur centralisé qui permet d'avoir une vue globale du réseau, d'un commutateur vierge avec des tables de flux et d'un canal sécurisé pour les échanges entre le commutateur et le contrôleur.

Ethane est basée sur des règles de gestion de réseau implémentées au niveau applicatif. Ensuite, ces règles déterminent le chemin suivi par les paquets, facilitant ainsi le déploiement des nouveaux services réseaux, notamment de sécurité. Ces règles permettent, par exemple, de rediriger certains types de trafics vers un firewall ou de l'ignorer.

Étant une évolution de SANE, Ethane prend en compte dans son architecture le principe de la gestion de la sécurité à travers le contrôle d'accès basé sur l'identité. Avec Ethane, la sécurité est considérée comme un sous-ensemble de la gestion du réseau et offre l'avantage de déploiement graduel des services selon l'évolution du réseau.

3.6 Analyse des menaces de sécurité du SDN

Le SDN a beaucoup d'avantages par rapport aux réseaux traditionnels, dû à la séparation entre le plan de contrôle et le plan de données, rendant les réseaux programmables, flexibles et évolutifs. Cependant, cela a induit au SDN d'autres problématiques de sécurité qui lui sont propres. Même si l'idée semble avoir convaincu un grand nombre de fournisseurs de services cloud et réseaux pour ses nombreux avantages, elle ne peut pas être largement adoptée si ses

défis de sécurité ne sont pas relevés. Ces défis doivent être mis en évidence afin que des mesures de sécurité appropriées puissent être prises de manière proactive. L'analyse de la sécurité par plusieurs auteurs [2, 60, 61, 62, 63] dans la littérature a montré que le SDN souffre de nombreuses menaces de sécurité. Comme l'indique la figure 3.2, il y a plusieurs vulnérabilités sur les différentes couches et interfaces de l'architecture des réseaux SDN.

Dans cette section, nous allons énumérer les vulnérabilités de toutes les couches et interfaces du SDN. Nous allons également étudier les approches proposées dans la littérature pour sécuriser les réseaux SDN.

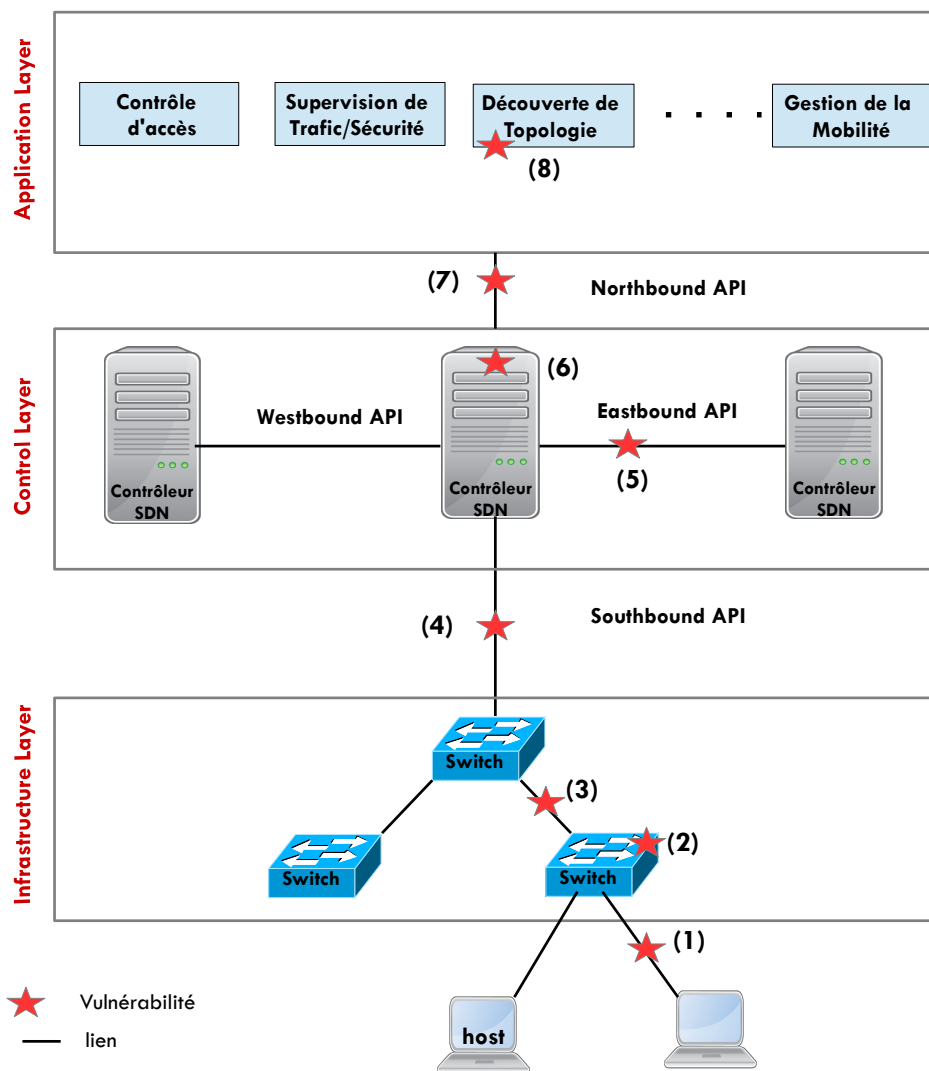


FIGURE 3.2 – Analyse des vulnérabilités du SDN

1. Sur le lien entre l'équipement réseau et le host : il existe différentes manières d'exploiter de manière malveillante les périphériques réseaux dans le plan de données du SDN.

Une machine hôte malveillante peut attaquer n'importe quel commutateur ou contrôleur du SDN en générant des paquets réseaux falsifiés. Autrement dit, un hacker peut injecter un volume important de données en faisant passer une machine malveillante pour une machine autorisée du réseau, pour lancer une attaque de déni de service (DoS) [64] par exemple. Il peut aussi modifier divers champs de paquets comme le champ IP ou MAC pour masquer son identité.

2. Sur l'équipement réseau : dans les réseaux SDN, c'est le contrôleur qui installe les règles de flux dans les tables de flux d'un commutateur de manière proactive ou réactive. Cependant, un commutateur dispose d'un nombre limité de tables de flux dans lesquelles les règles de flux sont installées. C'est pourquoi, depuis que la capacité de prise de décision des commutateurs a été externalisée dans le concept du SDN, par la séparation du plan contrôle du plan de données, le principal défi des commutateurs OpenFlow en matière de sécurité est de savoir comment distinguer les règles de flux authentiques des fausses. Un autre défi est aussi le nombre d'entrées de flux qu'un commutateur peut gérer. Avec OpenFlow, un commutateur OpenFlow doit mettre en mémoire tampon les flux de données jusqu'à l'obtention d'une règle de flux du contrôleur SDN. Ce qui fait qu'un commutateur OpenFlow est vulnérable aux attaques par saturation (par exemple), car ses ressources sont limitées pour la mise en tampon des flux en grande quantité non sollicités, qu'ils soient UDP ou TCP.

Un commutateur SDN peut être attaqué par un hacker en manipulant son comportement pour perturber les échanges dans le réseau. Cela peut se traduire par l'introduction d'un faux trafic ou règle de flux sur le commutateur dont les conséquences peuvent être désastreuses (vol de données, déviation du trafic, surcharge du contrôleur et des commutateurs voisins, suppression des paquets). Par exemple, la suppression d'une règle de flux peut interrompre la communication entre les machines hôtes. Cela peut également entraîner l'abandon des paquets réseaux, ou la génération de nombreux messages à destination du contrôleur SDN, pouvant provoquer un déni de service [65].

Par ailleurs, il faut noter que dans le concept du SDN, le contrôleur SDN a une implication directe sur les commutateurs SDN, car si le contrôleur est hors service, c'est l'ensemble du réseau qui est compromis [64].

3. Sur le lien entre les équipements réseaux : dans l'architecture SDN, les échanges sur les liens entre les commutateurs SDN ne sont pas chiffrés. Ce défaut offre la possibilité aux hackers d'intercepter les informations échangées sur ces liens. Ce qui peut compromettre la sécurité du réseau.

4. Sur le lien entre le contrôleur et l'équipement réseau : avec SDN, si un commutateur ne reçoit pas d'ordre d'acheminement de flux du contrôleur, soit en raison d'une défaillance du contrôleur SDN ou d'une déconnexion, le plan de données devient hors service [66]. Par conséquent, le lien entre le commutateur et le contrôleur doit être protégé car il peut

potentiellement devenir une cible privilégiée des hackers pour attaquer le réseau.

Par ailleurs, il faut savoir que le protocole OpenFlow dans sa version 1.3 n'intègre pas par défaut le chiffrement des données avec le protocole SSL ou TLS sur lien entre le contrôleur et l'équipement réseau. La configuration du SSL ou TLS est complexe, c'est pourquoi de nombreux fournisseurs ont ignoré la prise en charge de SSL ou TLS dans leurs commutateurs OpenFlow [63]. L'absence de chiffrement sur cette interface et le manque d'authentification des commutateurs au niveau du contrôleur rendent tout le réseau SDN vulnérable. Un tel défaut de sécurité peut être exploité pour obtenir un accès non autorisé aux données. Des attaques de type *Man-in-The-Middle*, l'écoute clandestine ou l'accès malveillant aux règles de flux peuvent aussi se produire.

5. Sur le lien entre les contrôleurs : les liens entre les contrôleurs sont sujets à plusieurs menaces. Pour assurer la redondance entre eux, les contrôleurs échangent des informations de mise à jour. Pendant ces échanges, un hacker peut exploiter le défaut de manque de chiffrement sur ce lien pour accéder et partager les informations de mise à jour [67].

6. Sur le contrôleur : les menaces spécifiques au SDN sont dues à la séparation entre le plan de contrôle et le plan de données, et l'introduction d'un contrôleur logiquement centralisé. Le contrôleur est le composant principal d'un réseau SDN qui assure le contrôle du réseau. Par conséquent, il peut être très ciblé pour compromettre le réseau en raison de son rôle pivot.

Les applications implémentées au-dessus du contrôleur dans l'architecture SDN peuvent être des menaces. Le contrôleur SDN doit donc faire face à ce défi pour authentifier et autoriser les applications avant toute utilisation. Il est également nécessaire de séparer les différentes applications en fonction de leurs implications en matière de sécurité, avant l'accès aux informations réseaux.

Une machine compromise connectée dans le même réseau qu'un contrôleur SDN peut permettre à un hacker de lancer facilement une attaque sur le contrôleur du SDN et prendre le contrôle entier du réseau [68]. De plus, le fait que le contrôleur soit utilisé pour traduire les commandes du réseau, peut être exploité pour introduire une attaque agrégée sur le réseau. Pour gérer tous les flux réseaux entrants, le contrôleur SDN peut également devenir un goulot d'étranglement.

Un autre grand défi pour l'implémentation de contrôleurs actuellement disponibles consiste à spécifier le nombre d'équipements réseaux à gérer par un contrôleur afin de faire face aux contraintes de latence. Si le nombre de flux sur le contrôleur augmente, il est très probable que la latence augmente, ce qui implique qu'il faut plus de puissance de traitement au contrôleur. Une limitation des capacités de traitement du contrôleur peut provoquer donc une défaillance totale du réseau. Pour éviter qu'un contrôleur ne devienne un point critique, il est possible d'utiliser plusieurs contrôleurs. Cependant, il est démontré dans [69] que l'utilisation de plu-

sieurs contrôleurs n'est pas une solution pour éviter que le contrôleur ne constitue un point critique. La raison avancée dans cet article est que si les contrôleurs supportant la charge du contrôleur défaillant dépassent leurs capacités de traitement de flux, alors cela peut provoquer une défaillance en cascade des contrôleurs.

Les attaques de type DoS et DoS distribuées sur le contrôleur SDN sont parmi les défis de sécurité les plus importants à relever, car ce genre de menaces sont très dangereuses et peuvent rendre une ressource réseau indisponible aux utilisateurs légitimes. Pour preuve, une attaque DoS sur un contrôleur SDN est démontrée dans [70].

7. Sur le lien entre les applications et le contrôleur : contrairement au *Southbound API*, le *Northbound API* n'est pas normalisé et souffre des vulnérabilités qui lui sont propres. Un manque de mécanisme pour assurer la confiance sur l'interface *Northbound API* utilisé pour la communication entre les applications [71] et le contrôleur, peut introduire des menaces. Autrement dit, une faible méthode d'authentification entre les applications et le contrôleur et une autorisation inappropriée permettent à un hacker de lancer, par exemple, une attaque d'usurpation d'identité ou un accès non autorisé aux applications. Ainsi, le hacker peut utiliser une application malveillante pour demander au contrôleur, par exemple, de déconnecter une application créant un problème de modification de flux. Il peut aussi envoyer une multitude de requêtes vers le contrôleur pour créer un goulot d'étranglement et surcharger le processeur.

Divers API pour l'interface *Northbound* peuvent augmenter les menaces de sécurité en raison des vulnérabilités intégrées. Cela diminue la fiabilité entre le contrôleur et diverses applications.

8. Sur les applications : les applications sont généralement utilisées par les hackers pour prendre la main sur le contrôleur. Cela est facilité par le fait que les applications et le logiciel contrôleur sont le plus souvent situés sur la même machine physique. Ainsi, lors des échanges entre les applications et le contrôleur, un code malveillant peut être introduit sur le logiciel de contrôle via le *Northbound API* qui est aussi vulnérable. Avec des applications non fiables, associées aux vulnérabilités intrinsèques aux API, un hacker peut prendre le contrôle du réseau en introduisant ses propres règles sur le contrôleur.

La gestion centralisée d'un réseau à travers un contrôleur est fondamentale pour les réseaux du futur, notamment l'Internet des Objets. Cependant, cela ouvre également de nouveaux défis en matière de sécurité. Du risque de point critique du contrôleur SDN aux vulnérabilités des interfaces, en passant par le besoin d'authentification et d'autorisation des applications réseau, les défis sont nombreux. De plus, la liste de ces défis de sécurité devrait augmenter avec le déploiement progressif du SDN. Le tableau 3.1 ci-dessous présente un résumé des principaux problèmes de sécurité existant dans les trois couches du SDN.

Couche SDN	Type de menace	Description
Couche Application	Manque d'authentification et d'autorisation	Pas de mécanisme d'authentification et d'autorisation convaincant pour les applications.
	Insertion frauduleuse de règles de flux	Des applications malveillantes ou compromises peuvent générer de fausses règles de flux et il est très difficile de vérifier si cela provient d'une application compromise.
	Manque de contrôle d'accès et de traçabilité	Difficulté à mettre en oeuvre le contrôle d'accès et la traçabilité sur les applications tierces.
Couche Contrôle	Attaques de déni de service (DoS)	La centralisation de l'intelligence du réseau sur un contrôleur et les ressources de traitement limitées du contrôleur sont les principales raisons qui attirent les attaques DoS.
	Accès non autorisé au contrôleur	Aucun mécanisme convaincant pour mettre en oeuvre le contrôle d'accès sur une application
	Évolutivité et disponibilité	La centralisation de l'intelligence dans une seule entité peut poser des problèmes d'évolutivité et de disponibilité.
Couche Infrastructure	Règles de flux frauduleuses	Le plan de données est vulnérable aux règles de flux frauduleuses.
	Attaques par inondation	Les tables de flux des commutateurs OpenFlow peuvent stocker un nombre limité de règles de flux.
	Piratage ou compromission du contrôleur	Le plan de données dépend du plan de contrôle qui assure sa sécurité. La sécurité du plan de données dépend de la sécurité du contrôleur.
	Attaques de niveau TCP	TLS est sensible aux attaques de niveau TCP.
	Attaque de type Man-in-the-middle	Dû à l'usage optionnel et la complexité de configuration du TLS.

TABLE 3.1 – Quelques types de menaces de sécurité du SDN [6]

3.7 Solutions de sécurité pour les différentes couches du SDN

Actuellement, une grande partie de la communauté des chercheurs se base sur le SDN pour superviser et implémenter des politiques de sécurité, afin de détecter et d'isoler un flux malveillant dans un réseau. L'architecture SDN a introduit une séparation du plan de données, des plans de contrôle et applicatifs qui doivent être protégés des menaces de sécurité de plus en plus sophistiquées, tels que le déni de service distribué (DDoS).

3.7.1 Solutions de sécurité pour la couche infrastructure

Les équipements réseaux, responsables de l'acheminement des données dans le concept du SDN, doivent être sécurisés des applications malveillantes pouvant installer ou modifier des règles de flux sur ces éléments du réseau. Par conséquent, des solutions de sécurité pour la couche infrastructure ont été proposées dans la littérature pour faire face à ces défis.

Ahmad et al ont proposé dans [72] un outil appelé VeriFlow, permettant de rechercher les règles de flux malveillants insérées par les applications SDN sur un commutateur OpenFlow. Il permet également d'empêcher de manière dynamique les règles frauduleuses d'atteindre les commutateurs OpenFlow, afin de maintenir l'intégrité des règles de flux. Les auteurs ont expérimenté cet outil sur Mininet, un environnement de simulation de réseau compatible OpenFlow, et les résultats montrent que VeriFlow peut, grâce au suivi des données de routage, réaliser

la détection d'une nouvelle règle de transfert en quelques millisecondes.

Dans [73], les auteurs ont proposé une plateforme du nom de FortNox pour permettre à un contrôleur NOX de vérifier en temps réel les incohérences des règles de flux, afin d'autoriser ou non les applications OpenFlow avant qu'elles ne puissent modifier les règles de flux. Cette plateforme fournit un mécanisme d'autorisation basé sur des signatures numériques et de contrainte de sécurité grâce à une extension logicielle sur un contrôleur de type NOX. À l'aide d'un système dynamique de détection de conflit de règles, FortNOX traite toutes les demandes d'insertion de règles OpenFlow à l'aide d'un algorithme d'analyse des conflits de règles. Ainsi, lorsqu'une règle de flux est insérée par une application de sécurité authentifiée, la plateforme FortNOX empêche d'autres applications d'insérer des règles de flux contradictoires sur un même réseau SDN.

VAVE [74] pour *Virtual source Address Validation Edge* est un programme de protection préventif expérimenté sur un contrôleur OpenFlow de type NOX, permettant d'atténuer les attaques DoS causées par une usurpation d'adresses IP. Quand un nouveau paquet qui ne correspond à aucune règle d'une table de flux sur un commutateur arrive, il sera envoyé au contrôleur pour la validation de l'adresse source où une usurpation d'adresse IP peut être détectée. Si le contrôleur détecte une usurpation d'adresse IP alors, il crée une règle dans la table de flux pour arrêter le flux spécifique venant de cette adresse source.

Lorsque le plan de contrôle est découplé du plan de données, les mécanismes de résilience en cas de défaillance et de récupération changent. Il est aussi prouvé dans [66] que le chemin entre un contrôleur SDN et les commutateurs est proportionnel à la perte de connectivité. C'est pourquoi dans [66] ces mêmes auteurs ont proposé une approche pour raccourcir le chemin entre les commutateurs et le contrôleur, dans le but d'améliorer la disponibilité de contenu des applications de sécurité, de permettre une restauration rapide et d'analyser la sécurité.

FlowChecker [75] est un outil de vérification de configuration utilisé pour identifier les incohérence des règles OpenFlow dans un ou plusieurs commutateurs. Autrement dit, FlowChecker, vérifie la cohérence des différents commutateurs et valide l'exactitude de la table de flux de la configuration déployée par les nouveaux services et protocoles. Il peut être utilisé comme un contrôleur OpenFlow centralisé pour recevoir des requêtes des applications OpenFlow afin d'analyser, de vérifier ou de déboguer la configuration.

SDNsec [76] est une extension de sécurité pour assurer la traçabilité des règles d'acheminement des paquets au niveau du plan de données du SDN. Autrement dit, s'assurer que les règles de flux sont correctement appliquées sur la couche infrastructure. Les auteurs ont proposé un mécanisme qui permet de veiller à ce que les commutateurs transmettent les paquets en fonction des instructions du contrôleur d'une part, et d'autre part, de valider le chemin des paquets afin de permettre au contrôleur de vérifier de manière réactive que le plan de données a

suivi les règles spécifiées. Cette solution garantit la cohérence de mise à jour des règles de flux, de sorte que le comportement du plan de données soit bien défini lors des reconfigurations.

Avant-Guard [64] est un système de défense contre les attaques par saturation sur le lien entre le plan de données et le plan de contrôle. Il crée une extension matérielle des commutateurs OpenFlow avec un proxy TCP pour atténuer les attaques basées sur TCP. Le proxy répond par un SYN-ACK et transfère le paquet SYN pour vérifier l'existence de la source et de la destination. Ce système considérera une connexion comme légale uniquement lorsque la source et la destination existent. Le problème est qu'Avant-Guard ne peut gérer que les attaques basées sur TCP et introduit de la latence pour les paquets SYN légaux. C'est pourquoi FloodGuard [77] a été proposé pour atténuer les attaques basées sur UDP et ICMP.

3.7.2 Solutions de sécurité pour la couche de contrôle

La programmabilité des contrôleurs SDN facilite énormément la tâche aux administrateurs réseaux, car ils peuvent installer des applications de sécurité sur les équipements réseaux, à travers les contrôleurs, grâce aux *Northbound API*. Ainsi, les applications informent les contrôleurs de faire fonctionner les équipements réseaux qu'ils contrôlent en tant qu'équipements de mise en exécution de la politique de sécurité. Les solutions de sécurité du plan de contrôle peuvent être des applications ou des approches permettant de protéger le plan de contrôle des menaces de type DoS, DDoS, applications malveillantes...

Pour faire face à ces types d'attaques, notamment de déni de service sur un contrôleur SDN, il faut analyser les caractéristiques du trafic des flux stockés dans les commutateurs OpenFlow.

FloodGuard [77] est une solution de sécurité évolutive et légère pour les réseaux SDN. Elle est destinée à protéger les réseaux SDN contre les attaques de déni de service. FloodGuard est composée de deux modules : *Proactive Flow Analyzer* et *Packet Migration*. *Proactive Flow Analyzer* effectue une analyse dynamique basée sur la logique d'exécution en temps réel du contrôleur, afin de détecter les flux de trafic causés par des attaques de déni de service. *Packet Migration* est responsable de la mise en mémoire tampon des paquets reçus et de leur soumission au contrôleur SDN, pour traitement à un débit limité via un algorithme de planification de rotation, ce qui empêche le contrôleur de consommer trop de ressources de calcul. Lorsqu'une attaque DoS est découverte, *Packet Migration* redirigera un message vers le commutateur. En parallèle, et au même moment, *Proactive Flow Analyzer* surveille le flux réseau pour déterminer divers paramètres ou variables sensibles, grâce auxquels le contrôleur SDN peut générer des règles de flux en aval et les installer de manière proactive sur le commutateur.

Security Enhanced-Floodlight (SE-Floodlight) [78] est une extension du contrôleur SDN Floodlight qui vise à créer une couche de contrôle sécurisée idéale pour le SDN. Le contrôleur

SE-Floodlight permet de fournir des mécanismes de séparation des privilèges, en ajoutant au contrôleur une *Northbound API* sécurisée jouant le rôle de médiateur entre les applications et le plan de données. L'introduction d'un nouveau sous-système d'audit OpenFlow sur SE-Floodlight permet de suivre tous les événements de sécurité survenus au niveau de la couche de contrôle.

DISCO est un contrôleur multi-domaine proposé par Kevin et al. dans [79] afin de fournir des fonctions de sécurité à la couche de contrôle pour les réseaux hétérogènes distribués. Il est composé d'un module de contrôle inter-domaine et d'un module de contrôle intra-domaine. Le module de contrôle inter-domaine est responsable de la supervision et de la gestion des priorités des données qui circulent entre les domaines, de manière à ce que les chemins de flux avec des priorités différentes puissent être calculés et transférés. Ainsi, le module de contrôle inter-domaine peut rediriger de manière dynamique ou arrêter le flux pour faire face aux attaques sur le plan de contrôle. Le module de contrôle intra-domaine assure la gestion des échanges entre les contrôleurs. Il dispose d'un émetteur-récepteur de messages et des agents qui lui permettent d'assurer la recherche des contrôleurs voisins et de fournir un canal de contrôle entre les contrôleurs. Les agents échangent les informations du réseau grâce au canal de communication fourni par le module émetteur-récepteur.

Dans [80], Tootoonchian et al. décrivent, HyperFlow, une plateforme de contrôle évolutive logiquement centralisée mais physiquement distribuée, basée sur la propagation d'événements. HyperFlow permet aux opérateurs réseaux de déployer plusieurs contrôleurs, capables de prendre des décisions localement afin de maximiser l'évolutivité du contrôleur SDN et de minimiser le temps d'installation de flux.

Afin d'améliorer la capacité de traitement des contrôleurs, McNettle a été proposé dans [81]. McNettle est un contrôleur SDN multiple coeurs de processeur mis au point pour les grands réseaux supportant les algorithmes de contrôle demandant beaucoup de ressources. Il est efficace en terme de performance et évolutif car il peut gérer jusqu'à 46 coeurs de processeur. Avec un langage de programmation de haut niveau, les administrateurs réseaux peuvent programmer l'extension de McNettle selon les besoins.

Les attaques de type DoS distribué peuvent rendre indisponibles différents services d'une organisation sur Internet, en les inondant par des requêtes malveillantes au détriment des demandes des utilisateurs légaux. Cela peut induire des pertes économiques ou nuire à l'image des entreprises ou organisations. C'est pourquoi, dans [82], S. Hoyos et al. présentent le *Support Vector Machine* (SVM), un modèle d'apprentissage supervisé qui analyse le trafic de données, filtre les en-têtes et normalise les données en fonction de variables opérationnelles. SVM permet d'éviter les attaques DDoS en utilisant des techniques de largage de paquets et de limitation de débit.

3.7.3 Solutions de sécurité pour la couche application

La couche de contrôle du SDN s'interface entre les équipements réseaux et les applications pour réduire la complexité du réseau à partir des applications. Ainsi, le SDN facilite le déploiement des applications qui peuvent extraire les données sur les statistiques du réseau et les caractéristiques des paquets via le contrôleur, pour implémenter des nouveaux services de sécurité. Pour sécuriser la couche application, plusieurs outils et langages de programmation réseau ont été proposés dans la littérature, tels que ProMeta, Frenetic et NetCore [83]. Pour déboguer et automatiser les tests des programmes OpenFlow, OFTesting [84], une application OpenFlow basée sur Python est utilisée par cette couche applicative.

PermOF [85] est un système de permission qui permet de fournir un contrôle de privilège aux contrôleurs OpenFlow et aux applications s'exécutant au-dessus du plan de contrôle du SDN. PermOF est basé sur un ensemble de 18 autorisations devant être appliquées par les API du contrôleur. Il propose également un *framework* d'isolation personnalisé qui maintient diverses priorités pour les applications et isole le trafic de contrôle du trafic de données, afin d'obtenir une isolation complète des ressources et un contrôle d'accès.

OpenFlow est un standard innovant et ouvert permettant la programmation dynamique des règles de contrôle de flux dans un réseau, mais peu d'attention est accordée par la communauté de chercheurs à la mise au point de méthodes permettant de vérifier que les règles de flux dynamiques insérées dans un réseau OpenFlow n'enfreignent pas la politique de sécurité du réseau. C'est pourquoi dans [86], les auteurs ont introduit *Flow Verification* (FloVer), un outil de vérification pour OpenFlow capable de vérifier que l'ensemble des règles de flux ne violent pas les politiques de sécurité du réseau. Implémenté en tant qu'application OpenFlow, qui s'exécute sur un contrôleur de type NOX, FloVer permet de vérifier que les nouvelles règles de flux créées par le contrôleur sont cohérentes avec un ensemble de propriétés spécifiées. C'est-à-dire que lorsqu'un contrôleur OpenFlow fournit les règles de flux nouvellement créées à un commutateur OpenFlow, FloVer vérifie un ensemble de propriétés spécifiées non contournées par rapport au jeu de règles de flux mis à jour.

Dans [87], les auteurs ont proposé FRESKO, un *framework* de sécurité pour OpenFlow capable de surveiller et de détecter une menace de sécurité dans un réseau SDN. Il est composé d'une couche application FRESKO et d'un noyau d'application de la sécurité. La couche d'application de FRESKO fournit un environnement de développement et un contrôleur de ressources grâce auxquels les développeurs peuvent utiliser le langage de script FRESKO pour composer des fonctions de sécurité, tandis que le noyau d'application de la sécurité de FRESKO est intégré au contrôleur OpenFlow pour garantir la conformité des règles et éviter les conflits. En intégrant FRESKO sur un contrôleur SDN de type NOX, les auteurs ont démontré que FRESKO n'introduit pas de surcharge et permet de créer assez rapidement des fonctions de

sécurité avec moins de lignes de code.

3.7.4 Conclusion

Nous avons présenté dans ce chapitre un aperçu des travaux de recherche existants sur la sécurité des réseaux traditionnels et SDN. Il est important de noter que le concept du SDN a considérablement changé la manière d'administrer et de sécuriser les réseaux, mais a aussi introduit de nouvelles vulnérabilités propres au SDN. À titre d'exemple, l'introduction d'un nouveau protocole ou d'une nouvelle API dans le SDN peut exposer le réseau à des menaces de sécurité particulières et donc nécessite des contre-mesures spécifiques.

Deuxième partie

Nouvelles architectures de sécurité pour les réseaux du futur

CHAPITRE 4

Gestion intelligente de la sécurité dans un réseau avec le SDN

Résumé : Après avoir présenté dans nos précédents chapitres le SDN et ses protocoles, l'Internet des objets et l'état de l'art sur la sécurité des réseaux SDN, dans ce chapitre nous présentons notre approche de gestion intelligente de la sécurité dans un réseau avec le concept du SDN. Plusieurs travaux dans la littérature ont proposé différentes approches de sécurité des réseaux avec le concept SDN ; notre approche se différencie de ces solutions par sa simplicité de mise en oeuvre, avec l'usage des protocoles OpenFlow et OpFlex. Le contenu de ce chapitre porte en partie sur notre publication dans [88].

4.1 Introduction

L'augmentation du nombre d'équipements connectés et le volume de données qu'ils génèrent a considérablement augmenté le besoin de sécurité de ces données. Actuellement, la sécurisation des échanges dans un réseau est en grande partie réalisée par des efforts d'anticipation des menaces et en utilisant les mécanismes traditionnels, tels que firewall, IDS/IPS... Mais ces approches doivent être améliorées pour assurer la sécurité de ces objets connectés car ils sont hétérogènes et mobiles. Autrement dit, les réseaux sont devenus trop complexes et sans frontières. C'est pourquoi, dans ce chapitre nous proposons une nouvelle approche globale de sécurité avec le SDN, permettant d'éviter les implémentations incohérentes de sécurité pouvant être source de complexité et nécessitant des compétences techniques supplémentaires, rendant la gestion des réseaux coûteuse. Dans ce travail, nous proposons et mettons en oeuvre un firewall intelligent basé sur le principe de fonctionnement du SDN, pour distribuer les politiques de sécurité dans un réseau. Avec cette solution de sécurité basée sur le SDN, la sécurisation des échanges dans un réseau devient cohérente, dynamique et intelligente, afin de mieux iden-

tifier les menaces de sécurité et offrir une vue globale de l'ensemble des connexions et éléments du réseau. Il s'agit de mettre en oeuvre une solution de sécurité de bout en bout grâce aux nombreux avantages du SDN.

Plusieurs travaux dans la littérature proposent des solutions de sécurité en utilisant l'architecture SDN. Certains ont développé et implémenté des applications de firewalling dans [89, 90, 91, 92, 93, 94] sur le contrôleur du SDN, d'autres ont installé des règles de sécurité directement sur le commutateur OpenFlow, pour assurer la sécurité dans un réseau. La contribution de notre approche est sa simplicité de mise en oeuvre par l'usage des protocoles OpenFlow et aussi la possibilité de distribuer les règles de sécurité sur les équipements de manière déclarative avec OpFlex, qui est un protocole proposé par Cisco pour le SDN, permettant de garder l'intelligence des équipements réseaux.

4.2 Approches existantes de sécurité avec le SDN

Pour sécuriser un réseau, nous distinguons deux directions principales : la première utilise la solution traditionnelle basée sur des composants de sécurité standards (comme les firewalls, IPS/IDS, etc.) et la seconde utilise le concept du SDN. Comme indiqué précédemment, le SDN simplifie la gestion en la centralisant et permet de programmer la sécurité grâce aux API fournies avec les contrôleurs. C'est pour ces raisons qu'il est aujourd'hui très étudié dans les travaux récents.

Dans [92], les auteurs ont développé un pare-feu centralisé de niveau 2, basé sur le filtrage des adresses MAC et sur un contrôleur SDN de type POX. Comme les attaques sont de plus en plus évoluées, ce type de filtrage est obsolète. Les auteurs de [95] ont ainsi développé une application de pare-feu basée sur les couches 2 à 4. Elle est capable d'analyser le trafic réseau et de comparer les en-têtes des paquets reçus en fonction des règles prédéfinies. Si l'application trouve une correspondance, le paquet est supprimé, sinon il est transféré à sa destination. Dans ces deux travaux, le contrôleur SDN possède une place centrale dans la détection. Or, cela induit un point critique : si le contrôleur est attaqué et qu'une personne malveillante en prend le contrôle, c'est toute la sécurité du réseau qui est compromise. De plus, si le trafic est trop important, la charge du contrôleur risque d'être trop grande. Là encore, si le contrôleur n'est plus en mesure d'effectuer les analyses, la sécurité n'est plus assurée.

Aussi, dans [91], les auteurs ont mis en place une infrastructure SDN avec un contrôleur unique. Ils ont étudié l'impact du traitement des flux liés à OpenFlow sur la latence des communications. En simulant du trafic ICMP à l'aide du simulateur Mininet, ils ont pu vérifier le bon fonctionnement des règles de filtrage mises en place au niveau de leur firewall logiciel. Même si la latence constatée n'est pas si importante, malgré le goulot d'étranglement de l'unique

contrôleur, il faut cependant remarquer que l'utilisation de Mininet ne reflète qu'une partie du trafic réel d'un réseau d'entreprise (qui impliquerait aussi un nombre de règles plus important) et que l'unique contrôleur est toujours un point critique de l'architecture.

Pour améliorer cela, dans [96], Flauzac et al. ont proposé une solution basée sur plusieurs contrôleurs avec la possibilité de répartir les contrôleurs par domaine, permettant ainsi d'avoir une solution d'équilibrage de charge du système de contrôle du réseau. La mise en place d'une telle architecture est ainsi décrite dans [80] : les auteurs proposent d'utiliser plusieurs contrôleurs NOX en utilisant HyperFlow. HyperFlow est une application permettant de propager les événements détectés par un contrôleur aux autres contrôleurs, mais aussi de palier la défaillance d'un contrôleur. Même si ce type d'architecture permet de régler le problème du goulot d'étranglement, il reste la surcharge de chaque contrôleur : le traitement nécessitant de traiter tous les flux, l'augmentation de la charge nécessite de dimensionner correctement les contrôleurs au risque de les voir tous hors service. Même si HyperFlow permet de relayer alors le flux vers un autre contrôleur, la saturation d'un contrôleur pourrait risquer de se propager aux autres.

Ainsi, une solution consiste à séparer une partie du traitement et donc d'alléger le travail du contrôleur, en exploitant un élément externe au contrôleur : un IDS. Ainsi, plusieurs solutions ont été proposées dans [97, 98, 99]. À partir d'une architecture et de l'utilisation du simulateur Mininet, les auteurs ont montré la faisabilité de ce type d'architecture. Cependant, un IDS fonctionne sur le principe de règles : si celles-ci ne sont pas décrites correctement, des attaques peuvent ne pas être détectées.

Pour proposer des solutions plus dynamiques et permettre une détection plus poussée, les auteurs de [100] et [101] ont proposé d'exploiter la *machine learning/deep learning* couplée à l'IDS. En entraînant le réseau de neurones sur des jeux de données bien connus, ils ont montré que ce type de solutions pouvait ainsi avoir de bonnes performances selon les cas. Par contre, aucune architecture globale, en liant la détection des alertes au contrôleur SDN, n'a été proposée à notre connaissance.

4.3 Gestion intelligente de la sécurité avec le SDN

Dans les réseaux classiques, la sécurité est principalement assurée par des firewalls pour se protéger des attaques. Avec l'arrivée des objets connectés, ces techniques ne suffisent pas pour se protéger des attaques de plus en plus sophistiquées et sur des périmètres encore plus vastes avec la mobilité. C'est pourquoi nous proposons une nouvelle approche qui repose sur un firewall intelligent et dynamique, utilisant les fonctionnalités d'un contrôleur SDN, que nous appellerons *Smart Firewall*. Cette solution est inspirée de l'approche sur la grille de sécurité proposée

par Flauzac et al. dans [102]. Ce concept est un middleware décentralisé pour implémenter des politiques de gestion et de sécurité des périphériques réseaux via un firewall. Tous les équipements du réseau sont situés dans un domaine appelé zone de confiance.

4.3.1 Gestion impérative des politiques de sécurité avec OpenFlow

Les premières approches de la gestion de sécurité avec le SDN utilisent des applications de type firewall et fonctionnent sur la base d'un contrôleur centralisé sur des réseaux de taille intermédiaire. Les politiques de sécurité sont définies sous forme de règles OpenFlow. Ces règles OpenFlow utilisent les *Northbound* API pour envoyer les instructions au contrôleur. Ce dernier récupère les informations reçues de l'application firewall, traduit les requêtes abstraites en commandes et les envoie aux équipements réseaux (commutateurs, routeurs...). Les équipements réseaux, qu'ils soient physiques ou virtuels, appliquent la demande des changements reçus du contrôleur pour sécuriser les échanges selon les attentes du firewall. La communication des équipements réseaux avec le contrôleur SDN est faite en utilisant les *Southbound* API, en l'occurrence OpenFlow. Il existe plusieurs versions d'OpenFlow depuis son introduction, mais la version la plus utilisée pour la mise en oeuvre du SDN est sa version 1.3. Dans cette version, OpenFlow permet la communication entre le contrôleur et les équipements réseaux via un canal sécurisé. Cependant, les échanges ne sont pas chiffrés entre le contrôleur et les équipements réseaux par défaut par les techniques de chiffrement TLS et les échanges se font en clair sur cette interface. Cette méthode permet certes de centraliser la gestion de la sécurité à travers un contrôleur SDN, mais fait face à d'autres problèmes de sécurité tels que le point critique du contrôleur s'il n'est pas distribué physiquement et sa capacité de traitement des échanges. Cette dépendance du contrôleur expose aussi le réseau à des attaques de type DoS et DDoS. Il est aussi important d'ajouter que le firewall doit non seulement contrôler les paquets de type IP, TCP et UDP mais aussi les communications de niveau applicatif (http,ftp...). De plus, lorsqu'on monte à l'échelle sur le réseau, le contrôleur devient un goulot d'étranglement avec un fort impact sur la performance et la latence du réseau. C'est pour répondre à ces manquements que l'approche de gestion déclarative des politiques, inspirée du modèle de Cisco ACI, a été introduite pour le SDN.

4.3.2 Gestion déclarative des politiques de sécurité avec OpFlex

Ce modèle est un modèle orienté objet basé sur la théorie de la promesse. Il s'appuie sur un contrôle évolutif des objets intelligents contrairement aux modèles impératifs. Le modèle déclaratif permet d'obtenir des performances élevées à l'échelle avec résilience, en déplaçant la complexité du réseau vers les périphériques. Il permet également de préciser les politiques

en termes abstraits et d'être hautement interopérables. Plusieurs fournisseurs peuvent avoir la même politique sans avoir à offrir un matériel, une configuration ou une version du logiciel identique. Ce modèle déclaratif, qui peut fonctionner en environnement multi-fournisseurs, permet de traduire et d'appliquer la définition des politiques dans les équipements réseaux. À notre connaissance, il n'existe à ce jour aucun protocole standard SDN pour réaliser cela sur un commutateur ou un routeur, pour assurer les services réseaux des niveaux 4 à 7. C'est ce vide qui a conduit au développement du protocole OpFlex.

Basé sur l'approche SDN, OpFlex permet d'envoyer une politique de sécurité sur un contrôleur SDN grâce à une API. Le contrôleur SDN va ensuite informer les équipements réseaux sur les attentes de sécurité grâce au protocole OpFlex. Il permet d'installer des agents sur les équipements réseaux. Selon le principe de fonctionnement de OpFlex, un agent appelé *Policy Element* communique avec un *Policy Repository*, qui définit les politiques de sécurité pour implémenter des règles de sécurité sur les équipements réseaux. Même si le SDN a introduit un contrôleur externe, permettant de programmer les flux grâce au protocole OpenFlow mais avec OpFlex, on peut encore mieux implémenter les politiques de sécurité sur le contrôleur et ensuite pousser sur les équipements réseaux. Ce qui permet, non seulement de mieux distribuer les politiques, mais aussi de réduire la charge du contrôleur, puisque les décisions de gestion de flux seront prises par les équipements réseaux et en local, contrairement à l'approche initiale du SDN où tout doit passer par le contrôleur. Le protocole OpFlex est destiné à maintenir le contrôle intelligent dans l'infrastructure de réseau elle-même, au lieu de le centraliser dans un contrôleur distinct (ce qui est l'essence même du modèle de découplage OpenFlow entre le contrôle et la transmission). En proposant OpFlex, Cisco compte bien garder le matériel réseau comme l'élément fondamental de la gestion programmable de ce réseau, plutôt que d'en faire un serviteur du contrôleur centralisé SDN.

4.3.3 Notre solution de firewall Intelligent

Dans le but de relever les défis liés aux limites des architectures traditionnelles pour la gestion de la sécurité, dans cette section, nous décrivons la manière dont les règles de sécurité sont gérées avec le SDN. Comme nous l'avons souligné dans le chapitre sur le SDN au début de ce document, ce concept a introduit une couche d'abstraction donnant la possibilité aux administrateurs réseaux de prendre le contrôle d'un réseau et de gérer le comportement des paquets circulant dans un réseau de bout en bout. Avec tous ces avantages offerts par le SDN, il est possible de programmer le comportement des flux à travers une application pour sécuriser un réseau.

Pour le développement de notre firewall basé sur le SDN, le protocole OpenFlow définit

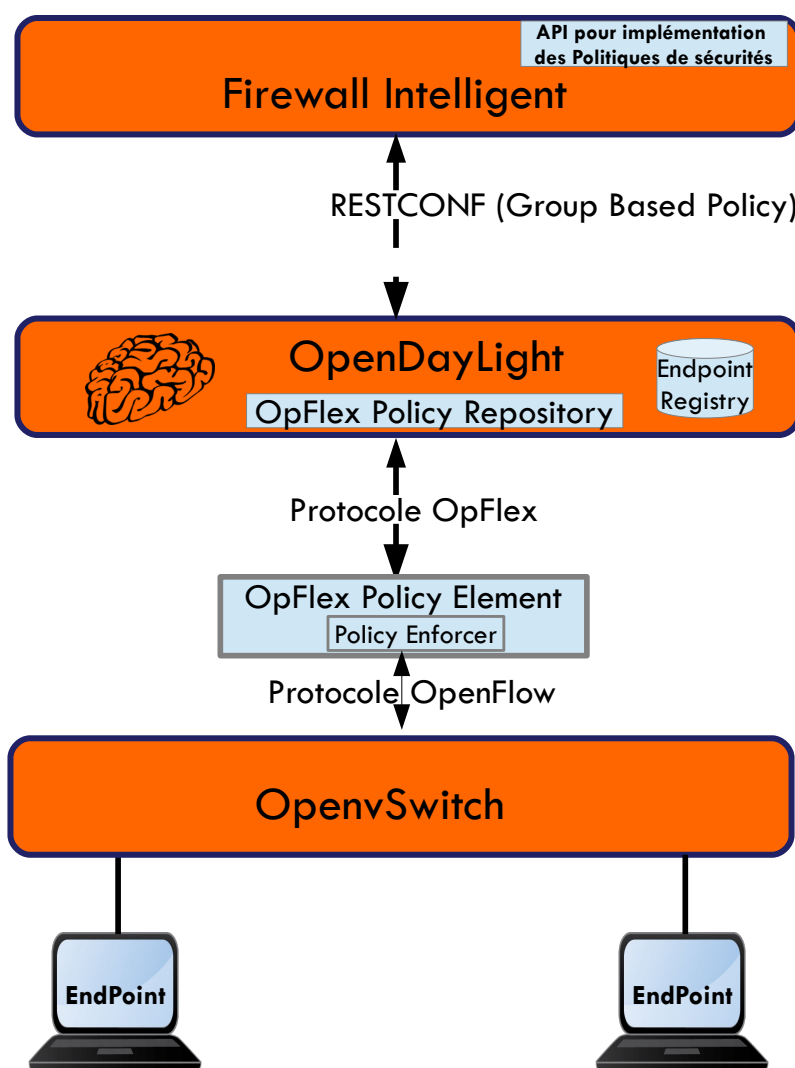


FIGURE 4.1 – Sécurité réseau avec SDN utilisant le protocole OpFlex

les messages de contrôle qui permettent au contrôleur SDN d'établir une connexion sécurisée avec les équipements réseaux, de lire leur état et d'installer les règles de flux pour la manière dont les échanges doivent se faire. Avec OpenFlow, les équipements réseaux sont capables de gérer l'acheminement des flux selon des règles prédéfinies par l'administrateur. En revanche, si un device est corrompu par un virus, cette menace peut se propager sur le réseau sans aucun contrôle. De plus, n'importe qui peut connecter son appareil sur le réseau. C'est pourquoi dans la conception de notre Firewall Intelligent nous proposons d'utiliser OpFlex en complément d'OpenFlow, pour distribuer les règles de sécurité sur les devices. Ainsi, nous pouvons contrôler les échanges de type applicatif tels que http, ftp ou ssh, ce qui n'est pas possible avec la spécification actuelle du protocole OpenFlow avec ses 14 champs de correspondance. À chaque échange, l'équipement réseau enverra les informations du device au contrôleur. Si le contrôleur

détecte une menace, alors il peut désinstaller le flux précédemment installé.

Nous avons dans un premier temps mis en oeuvre un réseau OpenFlow avec un contrôleur SDN centralisé de type OpenDaylight, comme l'indique la figure 4.2, pour comprendre le fonctionnement du SDN que nous allons détailler dans le chapitre implémentation à la fin de ce document.

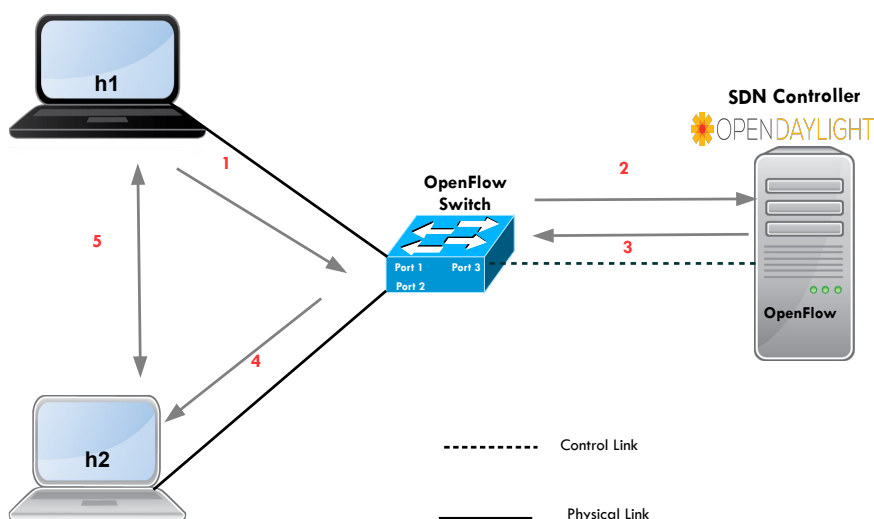


FIGURE 4.2 – Mécanisme d'échanges de flux dans un réseau OpenFlow

Le diagramme de la figure 4.3 montre la séquence d'échange des différents types de message entre deux machines dans un réseau OpenFlow. Nous avons constaté que l'usage d'un contrôleur unique expose non seulement le réseau aux risques d'attaque de type DoS ou DDoS, mais aussi un défaut matériel peut mettre le réseau hors service. De plus, le réseau pourrait faire face au problème de capacité de traitement en cas de montée en charge. C'est pour répondre à cette problématique que nous avons proposé le *Smart Firewall*, pour sécuriser les échanges dans un réseau. La figure 4.1 illustre l'idée principale de notre approche.

Notre firewall intelligent est basé sur la distribution des politiques de sécurité sur un ou plusieurs commutateurs OpenFlow à partir d'un contrôleur SDN, en utilisant le protocole OpFlex. Il faut comprendre au passage que ce protocole ne supprime pas le besoin d'OpenFlow car ce dernier est nécessaire pour décomposer une politique de sécurité abstraite en un langage compréhensible par les périphériques. OpFlex est compatible OpenDaylight et OpenVswitch (OVS), il est basé sur deux éléments importants : *Le Policy Repository* et le *Policy Element*.

Contrairement à OpenFlow, où toutes les fonctions de contrôle du réseau sont centralisées sur un contrôleur SDN, OpFlex se concentre principalement sur les politiques de sécurité qui doivent être définies sur le *Policy Repository* et envoyées au contrôleur qui les enverra sous forme de règles de sécurité au *Policy Element* qui seront installées sur le commutateur OpenFlow. Les

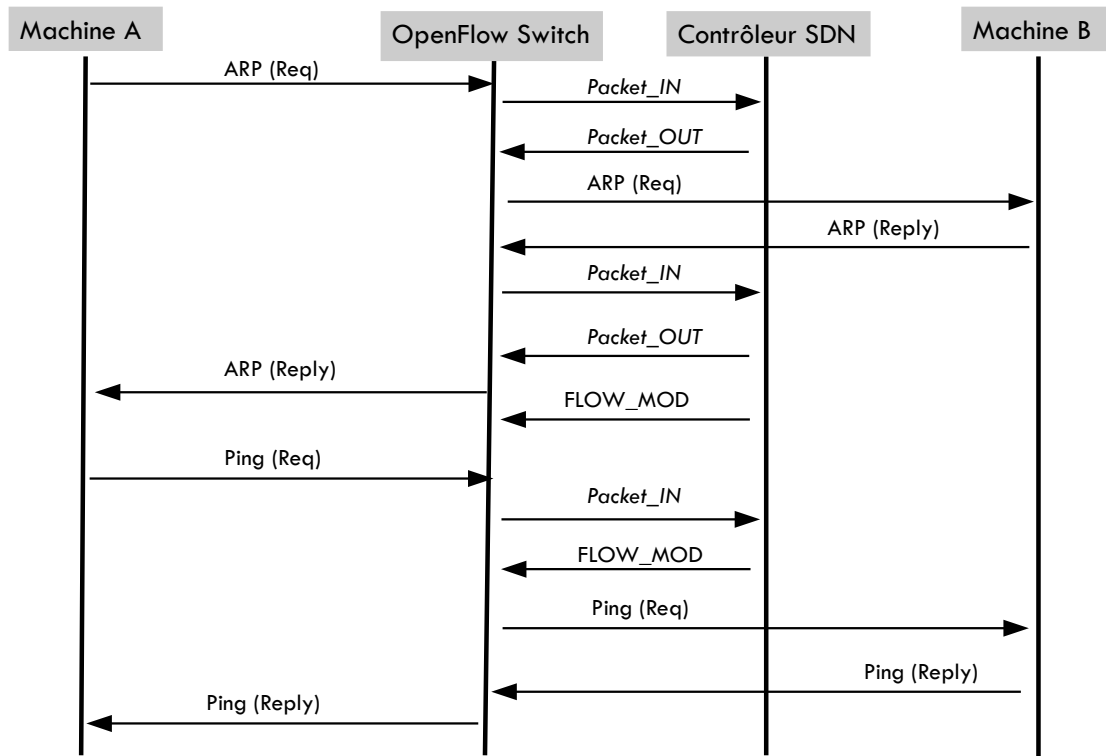


FIGURE 4.3 – Diagramme d'échanges OpenFlow

politiques peuvent être des filtres entrants et sortants mais peuvent aussi être des paramètres de qualité de service.

Cette manière de gérer les politiques de sécurité évitera que le contrôleur ne devienne un goulot d'étranglement du réseau. De plus, une indisponibilité temporaire du contrôleur SDN n'empêchera pas le réseau de fonctionner, car les équipements réseaux disposent d'une intelligence, et grâce à l'agent installé sur le commutateur, le réseau peut continuer à fonctionner d'où une plus grande résilience. En revanche, il n'est pas possible de mettre à jour les politiques de sécurité si le contrôleur est hors service. Dans ce modèle déclaratif basé sur OpFlex, l'intelligence est distribuée sur le réseau et la gestion des politiques est centralisée.

Pour intégrer OpFlex sur le contrôleur OpenDaylight, nous avons installé : *OpFlex Protocole Library*, *OpFlex SB plugin* et *OpFlex Policy Element*. L'échange des messages entre l'OVS et le contrôleur se fait avec le protocole OpFlex, supporté par le logiciel contrôleur OpenDaylight. Un agent appelé *Policy Element* intégré sur l'OVS est chargé de rendre les règles abstraites reçues du *Policy Repository* en concrètes et ensuite les exécuter sur l'OVS.

Le device ou l'équipement terminal peut être connecté à l'OVS via le *Policy Element*, capable d'appliquer les règles de sécurité localement. Le *Endpoint Registry* qui réside sur le *Policy*

Repository est une composante logique qui enregistre les informations sur l'état opérationnel (identité, localisation...) des *Endpoint* (devices) dans le système. Les statistiques et les défauts sont remontés via l'entité logique *Observer* comme le montre la figure 4.4 ci-dessous.

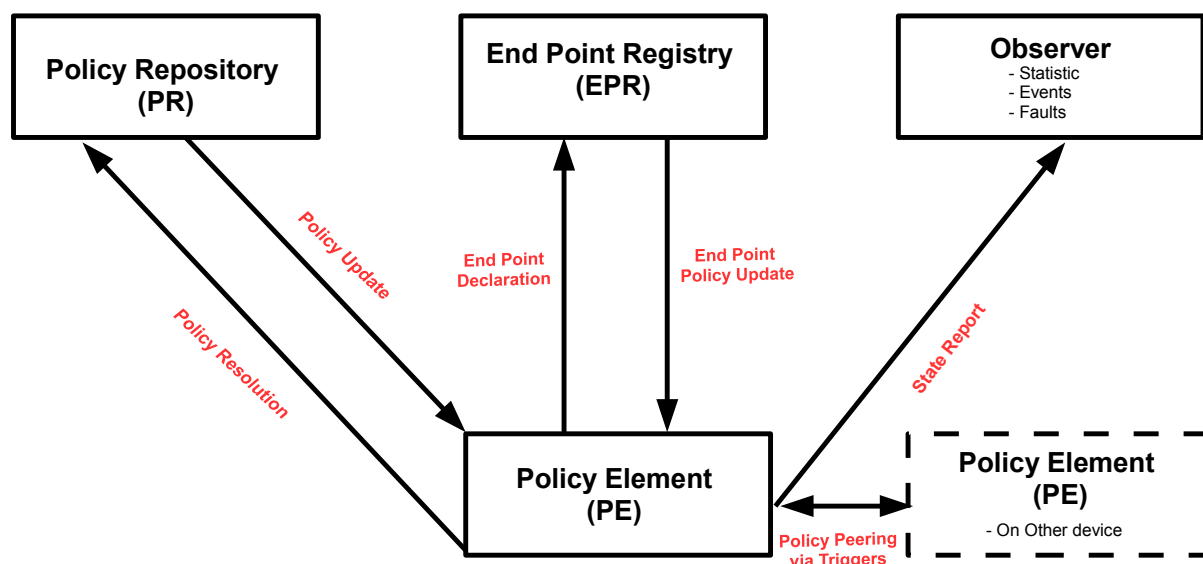


FIGURE 4.4 – Principe de fonctionnement du protocole OpFlex

Le *Policy Element* est un agent qui fonctionne avec l'OVS pour appliquer des politiques de sécurité sur des groupes avec des machines virtuelles ou des conteneurs rattachés localement sur l'OVS. Il est aussi conçu pour fonctionner correctement avec des outils d'orchestration, tels que OpenDaylight et *OpenStack*. Le diagramme de la figure 4.5 montre les détails des différentes séquences d'échanges lors de l'exécution de notre firewall.

OpenDaylight fournit une *Northbound API* pour le *Group Based Policy*^{1 2} et une *Southbound API* pour le protocole OpFlex.

Le *Group Based Policy* (GBP) définit un modèle de politique centré sur l'application pour OpenDaylight, qui permet de séparer les informations sur les exigences de connectivité des applications. Ci-dessous quelques termes clés :

- **Endpoint** : peut être n'importe quel périphérique comme une machine virtuelle, une interface, etc.

1. https://wiki.opendaylight.org/view/Group_Policy:Main(Online : accessed on 23/10/2018)

2. <https://wiki.openstack.org/wiki/GroupBasedPolicy>(Online : accessed on 23/10/2018)

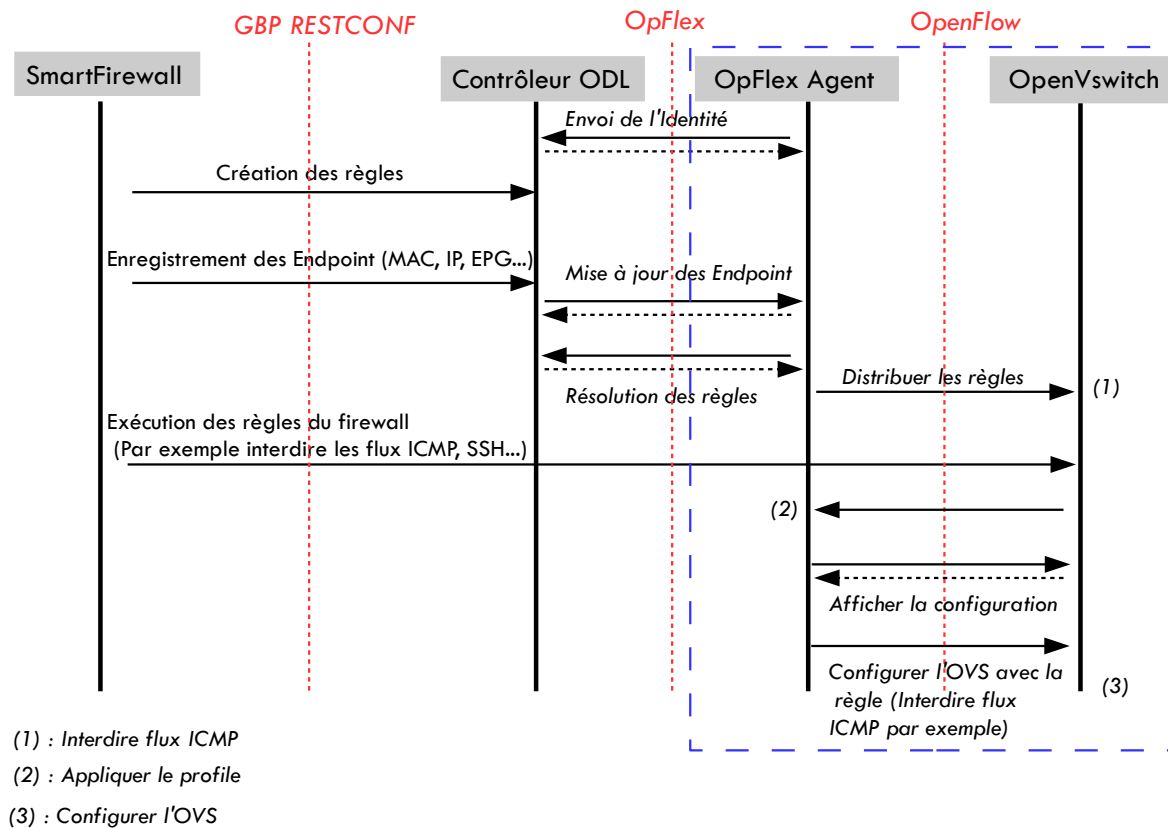


FIGURE 4.5 – Diagramme d'échanges lors de l'exécution du firewall

- **End Point Group (EPG)** : ensemble des *Endpoint* partageant la même politique.
- **Contrat** : définit comment les EPG devraient communiquer entre eux.
- **Clause, Subject** : spécifie les détails du contrat.

Le GBP regroupe les *Endpoint* en EPG c'est-à-dire que chaque *Endpoint* doit être affecté à un EPG. Ensuite, des contrats sont appliqués pour définir le type de trafic autorisé à être échangé sur le réseau. Les contrats déterminent la manière dont les *Policy Element* peuvent communiquer entre eux. Un contrat fournit un sujet qui spécifie un contrat en détail en définissant sous forme de règles pour gérer le trafic du réseau et des clauses pour déterminer comment un contrat peut être appliqué aux *Endpoint* et aux EPG. Dans notre cas, nous avons juste un EPG et deux *Endpoint* comme l'indique la figure 4.1. Ci-dessous sur la figure 4.6 un exemple d'enregistrement d'un *Endpoint* que nous avons appelé host1 (h1).

Cette solution de sécurité peut être renforcée par l'installation sur le contrôleur des applications d'authentification, de contrôle d'accès au réseau (NAC) ou d'autres applications selon le besoin.

```
{
  "policy-space-name": "test",
  "endpoint-group-name": "group1",
  "interface-name": "s1-eth1",
  "ip": [
    "10.10.10.1"
  ],
  "mac": "00:00:00:00:00:01",
  "uuid": "83f18f0b-80f7-46e2-b06c-4d9487b0c754"
}
```

FIGURE 4.6 – Exemple d'enregistrement d'un *Endpoint*

4.3.4 Conclusion

Dans ce chapitre nous avons décrit notre approche de gestion intelligente de la sécurité dans un réseau avec le SDN. La proposition par Cisco du protocole OpFlex, un protocole complémentaire à OpenFlow, a permis d'améliorer de manière significative la distribution des politiques de sécurité sur un commutateur OpenFlow. Cette gestion déclarative des politiques de sécurité dans un réseau permet de rendre le réseau plus résilient et évolutif. Ce qui fait que, même en cas de panne d'un contrôleur SDN, le réseau reste en service en attendant la maintenance car les équipements réseaux conservent leur intelligence, contrairement à l'idée initiale du SDN où les équipements réseaux ne disposent pas d'intelligence.

CHAPITRE 5

Gestion automatisée du réseau en fonction des vulnérabilités détectées

Résumé : Dans le précédent chapitre, nous avons notre solution de firewall basée sur le SDN. L'implémentation de cette solution nous a permis d'identifier les menaces internes d'un réseau utilisant le protocole OpenFlow et OpFlex. C'est pourquoi dans ce chapitre nous présentons notre solution de sécurisation des échanges dans un réseau en fonction des événements détectés avec du SDN. De plus, nous allons proposer une architecture étendue aux objets connectés. Le contenu de ce chapitre correspond à notre publication dans [103].

5.1 Introduction

Dans le précédent chapitre, nous avons proposé une approche intelligente de la sécurité basée sur un firewall. Notre travail était basé sur deux protocoles complémentaires, à savoir OpenFlow et OpFlex, avec une facilité de distribution des politiques de sécurité dans un réseau. Pour améliorer la sécurité plusieurs solutions ont été proposées dans la littérature pour la sécurité des réseaux SDN avec un IDS, afin que tout le trafic dans un réseau soit analysé.

Dans [104], les auteurs ont proposé un IPS comprenant un module IDS qui analyse et détecte les menaces dans un réseau compatible SDN. À la détection des menaces l'IDS met à jour l'IPS avec la liste des menaces potentielles. Ainsi l'IPS prend les mesures pour prévenir et atténuer les menaces dans le réseau.

L'approche proposée dans [105] est un système de sécurité réseau avec du SDN pour protéger un réseau contre les attaques par déni de service distribué (DDoS). La solution proposée comprend un contrôleur SDN, un serveur IDS responsable des décisions à prendre et un IDS host. Dans un premier temps, l'IDS host analyse et détecte un flux malveillant et informe l'IDS serveur. Ensuite le serveur IDS envoie au contrôleur SDN une alerte pour traiter la menace.

Selon les auteurs, cette manière de faire permet de réduire la charge de travail d'analyse du contrôleur SDN. En revanche, cette approche ne précise pas si le contrôleur SDN peut envoyer une règle au commutateur pour supprimer le flux malveillant.

Dans [106], une plate-forme logiciel de sécurité composée d'une couche application comprenant les services réseaux, d'une couche de management composée d'un contrôleur SDN et d'une couche de données incluant un firewall et un IDS pour la détection de trafic malveillant, a été proposée. Dans leur approche, le contrôleur SDN a été utilisé pour mettre à jour les politiques de sécurité afin de bloquer les accès non autorisés. Cependant, aucun mécanisme d'analyse, de détection et de suppression de flux malveillants n'a été abordé dans cet article.

Dans ce chapitre nous allons décrire notre approche de la sécurité en fonction des événements détectés et de manière automatisée. Cette solution étant basée uniquement sur OpenFlow, nous avons introduit un IDS permettant de collecter, analyser, détecter et isoler un flux malveillant de manière automatisée. Pour éviter que le contrôleur ne devienne un point critique, nous allons également proposer dans ce chapitre une architecture distribuée et étendue aux objets connectés.

5.2 Sécurité IoT et SDN

Dans [29], les auteurs ont proposé une architecture SDN-IoT avec une implémentation NFV pour faciliter la gestion des objets connectés. Cette architecture se propose, par l'usage du SDN, de faciliter et d'accélérer l'innovation dans la gestion des objets connectés. Pour cela, les auteurs proposent un framework SDN-IoT.

Bull et al [107] proposent une méthode pour détecter et atténuer le comportement anormal d'un objet connecté à partir d'une passerelle IoT, basée sur le SDN. Les entrées de flux de la passerelle IoT sont préalablement configurées pour permettre d'analyser les flux, afin de détecter tout comportement anormal et associer une action. Trois types d'actions ont été définies à savoir : bloquer, transférer ou appliquer les règles de qualité de service. Le problème de cette solution est la configuration de manière statique des règles sur la passerelle IoT, même si l'usage de plusieurs contrôleurs permet d'assurer la haute disponibilité du réseau.

Dans [108], les auteurs proposent une architecture distribuée de sécurité pour l'IoT avec du SDN. Leur solution permet d'assurer la surveillance et la haute disponibilité du réseau par l'usage de plusieurs contrôleurs SDN synchronisés et répartis par domaine. Ils proposent également un protocole de routage multi-domaine en environnement SDN pour assurer les échanges entre les contrôleurs des différents domaines. Même si cette solution a l'avantage d'être testée en environnement virtuel, elle ne dispose pas de mécanisme de détection d'intrusion d'où des menaces provenant de l'intérieur du réseau pourraient compromettre la sécurité du

réseau.

Dans [109] l'auteur présente une vue d'ensemble de l'état actuel de l'IoT ainsi que les défis de sécurité associés tels que l'identification des objets, la confidentialité et l'intégrité, l'authentification et l'autorisation, et les logiciels malveillants dans IoT. Il propose aussi une architecture de sécurité avec du SDN, basée sur un mécanisme de sécurité où un contrôleur IoT échange avec des agents IoT. Le contrôleur IoT est responsable des décisions à prendre en fonction des informations reçues des agents IoT et puis répercute sur le réseau via le contrôleur SDN. À la réception de la demande de connexion de son agent, le contrôleur IoT établira les règles de transfert en fonction des protocoles de réseau utilisés et communiquera ces règles au contrôleur SDN.

5.3 Architecture de gestion distribuée et collaborative de sécurité dans un réseau

Comme nous avons vu dans le précédent chapitre, les solutions centralisées proposées avec un seul contrôleur, ou plusieurs mais qui sont redondés et non distribués, n'assurent pas suffisamment de sécurité car un déni de service, ou une prise de contrôle à distance du contrôleur par un pirate, peut rendre le réseau inopérant. Cela peut être évité par l'usage de plusieurs contrôleurs distribués. Dans la pratique, la fonctionnalité MD-SAL (Model-Driven Service Abstraction Layer) fournit par le contrôleur OpenDaylight permet de mettre en place la redondance et la haute disponibilité avec plusieurs contrôleurs SDN. Aussi, les menaces provenant des utilisateurs par l'usage des clés USB, des modems d'accès Internet ou des objets connectés, doivent être détectées et isolées. C'est pourquoi nous proposons une nouvelle architecture de sécurisation des échanges dans un réseau étendue à l'IoT.

Notre solution est inspirée du concept de grille de sécurité [102] et de notre approche de firewall intelligent [88] pour améliorer la sécurité dans un réseau classique et l'étendre à l'IoT. Dans cette approche, nous proposons une solution collaborative de sécurité avec une architecture distribuée des contrôleurs, couplée avec des IDS. Nous avons opté pour une architecture distribuée SDN car l'architecture centralisée avec un seul contrôleur implique un fort risque d'indisponibilité du réseau en cas d'attaque de déni de service (DoS). À titre d'exemple, si la menace est uniquement sur une machine, cela n'est pas critique et isoler la machine peut être une solution, mais si c'est le contrôleur unique qui est compromis, c'est tout le réseau qui est menacé. L'usage de plusieurs contrôleurs permet donc de créer de la redondance, assurer une haute disponibilité et réduire la latence du réseau.

Comme l'indique la figure 5.1, notre solution consiste à un ou plusieurs clusters. Chaque

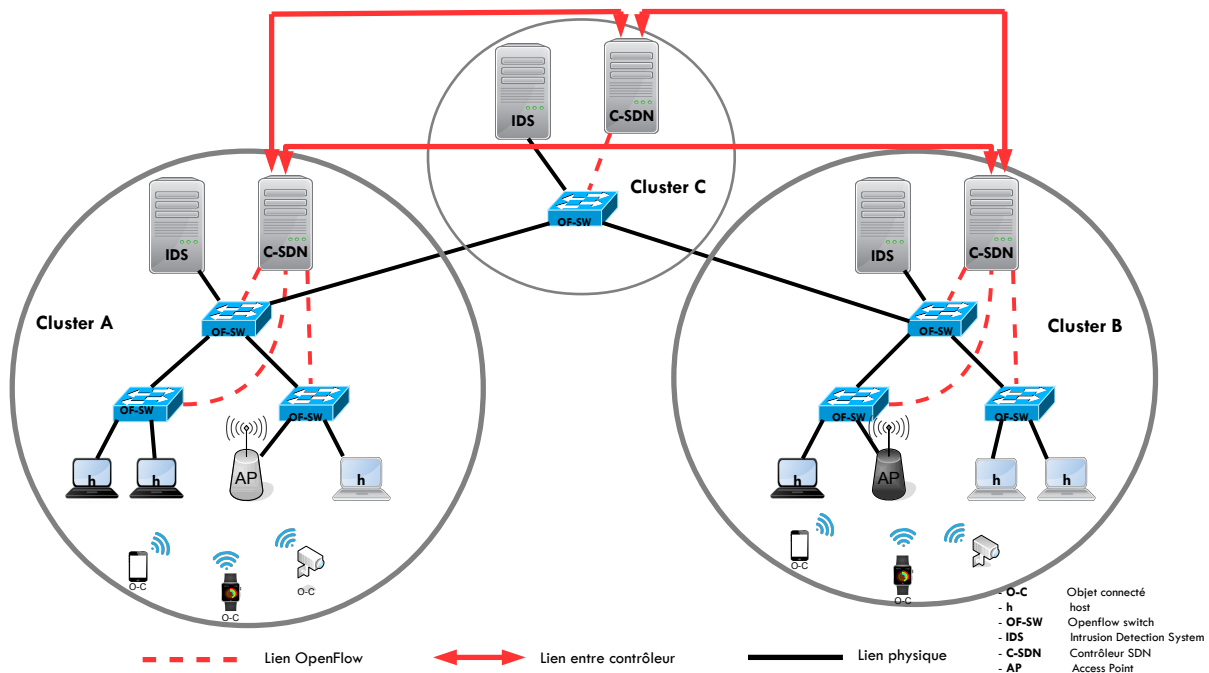


FIGURE 5.1 – Architecture distribuée et collaborative de sécurisation d'un réseau avec du SDN

cluster est composé d'un ou de plusieurs équipements réseaux qui sont responsables de l'interconnexion des devices dont des objets connectés. Au sein de chaque cluster, un contrôleur SDN gère le réseau OpenFlow. Chaque contrôleur SDN est couplé à un IDS. L'IDS est responsable de la détection des intrusions dans le périmètre réseau de chaque cluster. Autrement dit, un cluster constitue un domaine du SDN dans lequel nous utilisons un réseau OpenFlow avec un contrôleur SDN et un IDS pour gérer le domaine de sécurité, que nous appelons zone de confiance. Pour former une zone de confiance, l'ensemble des équipements et devices de cette zone doit être totalement sécurisé. Ce travail de sécurité est assuré par le couple contrôleur et IDS. Le contrôleur SDN sert de firewall intelligent pour la zone de confiance et dispose des règles de sécurité spécifiques aux besoins de sécurité du cluster. Ces règles sont programmées par un administrateur. Elles peuvent être distribuées sur les autres zones de confiance si le besoin de sécurité est le même à travers l'API East-West.

Notre approche permet, non seulement de gérer la sécurité de manière totalement décentralisée à travers une gestion locale de la sécurité par le couple contrôleur SDN/IDS, mais aussi que les contrôleurs échangent les informations sur les menaces détectées dans leurs clusters respectifs. Le contrôleur SDN est l'élément central pour la gestion de la sécurité dans chaque zone de confiance. Il a une vue globale du réseau, gère le trafic et distribue les politiques de sécurité sur les équipements réseaux de son propre cluster.

Avant d'isoler la menace par le contrôleur SDN dans chaque cluster, il faut la détecter. C'est pourquoi nous utilisons un IDS pour résoudre ce problème. Dans la pratique, cela peut être

réalisé par la mise en place d'un IDS de type snort ou autre.

Dans le fonctionnement du SDN, une fois les flux installés sur le commutateur SDN, les échanges ne passent plus par le contrôleur, ce qui exposerait le réseau aux menaces en cas de compromission d'une des machines du réseau. Pour résoudre ce problème dans chaque zone de confiance nous procédons comme l'indique la figure 5.2 en trois phases :

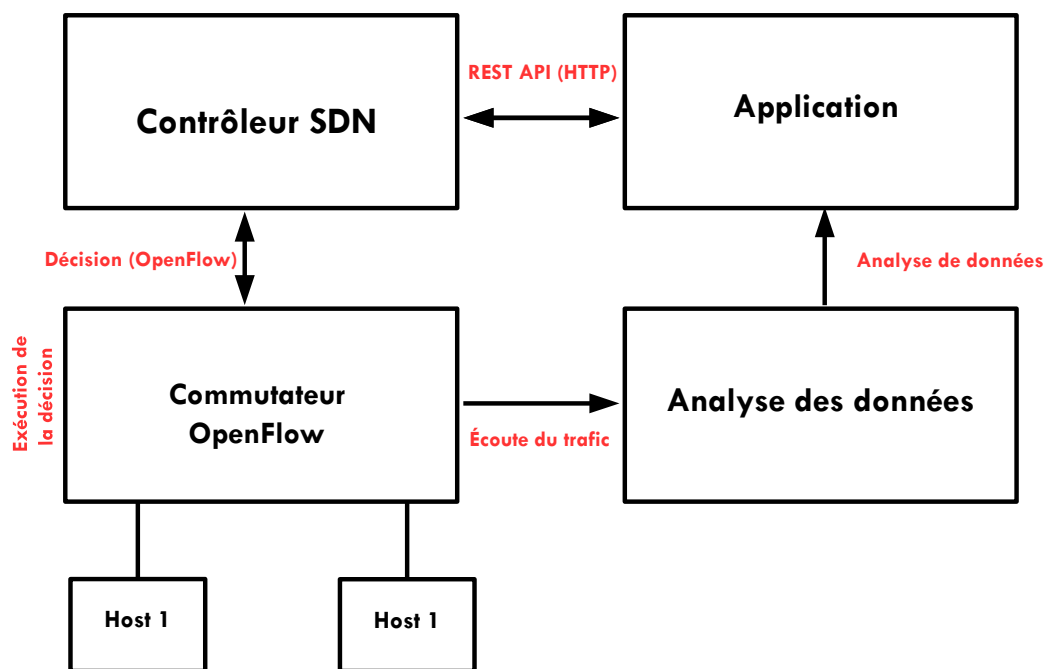


FIGURE 5.2 – Approche théorique de la solution

5.3.1 Collecte des données du réseau

Pendant cette première phase nous collectons les données à analyser avec un mécanisme spécifique pour qu'aucun flux n'échappe au contrôle. Pour cela, nous avons fait appel à la technique de port mirroring pour remonter l'ensemble du trafic qui circule sur le réseau vers un port particulier, afin de l'analyser constamment par la suite.

Dans un environnement SDN le port mirroring se fait en trois phases comme l'indique la figure 5.3, pour isoler un flux malveillant : (1) implémentation de la fonction de port mirroring par le contrôleur SDN, (2) exécution de la fonction de port mirroring par le commutateur OpenFlow pour envoyer tout le trafic vers l'IDS, (3) blocage ou acheminement du flux par le commutateur OpenFlow selon les instructions du contrôleur.

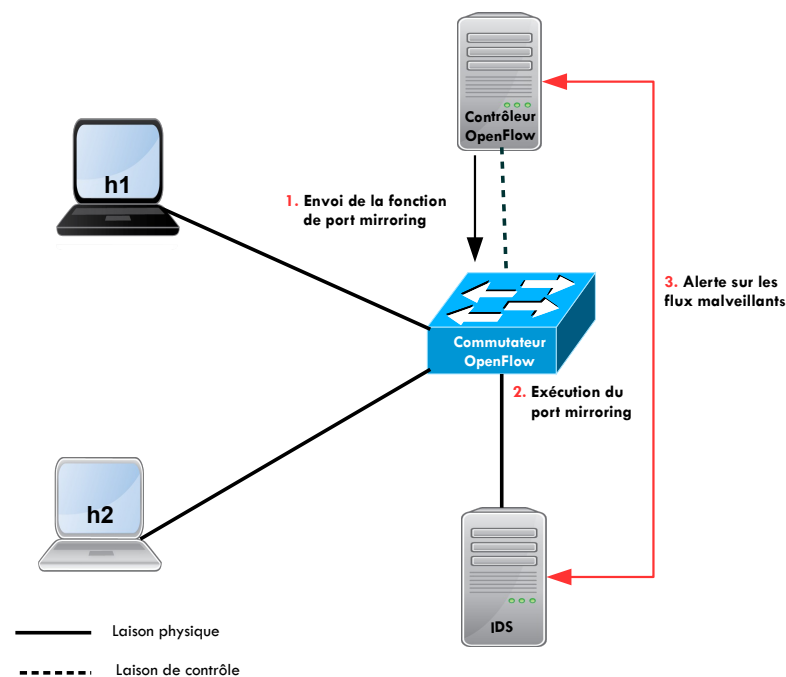


FIGURE 5.3 – Intégration d'un port mirroring dans le contexte SDN

5.3.2 Analyse, détection et génération des alertes

La seconde étape est l'analyse et la détection de menaces puis la génération d'une alerte. Pour réaliser cela, nous avons utilisé un IDS pour écouter l'ensemble du trafic du réseau, analyser et détecter les flux malveillants.

L'IDS analyse les données réseaux et déceles des anomalies ou des patterns d'attaque prédéfinis par l'administrateur réseau. Cette détection porte principalement sur l'analyse des entêtes des paquets des couches réseaux et transports mais également sur le contenu des paquets.

Pour détecter un flux malveillant, l'IDS utilise principalement deux méthodes d'analyse, à savoir la méthode de détection à base de signatures permettant de déceler des patterns connus dans les données analysées, ou la méthode de détection comportementale détectant les déviations d'un comportement vis à vis d'un profil normal. Dans les deux cas, l'IDS compare les données analysées à une référence décrite, soit par une signature, soit par un profil normal.

Une fois les données analysées, l'IDS peut générer une alerte sous forme de fichier log en cas de flux malveillants.

5.3.3 Suppression des flux malveillants

En dernière étape, nous avons développé une application qui permet d'extraire, d'analyser les logs puis d'envoyer une instruction au contrôleur SDN pour supprimer dynamiquement les

flux malveillants via l'API REST.

En suivant cette procédure, dans chaque zone de confiance, l'IDS analyse le trafic et envoie une alerte en cas de flux malveillants au contrôleur, qui prendra une décision en envoyant une règle de flux au commutateur OpenFlow via le protocole OpenFlow, pour interdire le flux en question. Chaque contrôleur a ses propres politiques de sécurité implémentées en fonction des retours de l'IDS.

La technique de port mirroring est utilisée pour collecter les flux sur un port particulier afin d'analyser les données constamment par l'IDS. L'IDS analyse le trafic et génère un fichier logs. Ce fichier est ensuite analysé par un script développé à cet effet pour analyser le log et envoyer une instruction pour supprimer le flux sur l'équipement réseau via l'API RESTful.

Pour prévenir et arrêter les futurs menaces dans les autres clusters, les contrôleurs échangent des informations sur les menaces de sécurité dans leurs domaines respectifs, afin qu'ils ne se propagent pas dans les autres clusters via leur API *East-Westbound*.

La notion de zone confiance nous permet d'avoir des domaines de sécurité réseau et qui ont la possibilité de partager les informations sur les menaces avec d'autres clusters.

5.4 Conclusion

Les techniques traditionnelles de sécurité des réseaux basées sur des équipements indépendants tels que firewall, IPS/IDS et NAC ne suffisent plus aux besoins de sécurité des réseaux du futur, notamment des objets connectés. C'est pourquoi, nous proposons une nouvelle solution de sécurité basée sur l'analyse, la détection et la suppression des menaces de manière automatisée grâce au nouveau concept SDN. Cette solution basée sur un contrôleur SDN et un système de détection des menaces, offre la possibilité de gérer la sécurité dans un réseau en fonction des événements détectés. Si notre solution est testée aujourd'hui pour sécuriser un réseau classique, elle peut être étendue à la sécurisation des objets connectés dans le futur.

En perspective, nous comptons monter à l'échelle notre réseau pour analyser l'impact de notre solution sur le temps de réaction.

CHAPITRE 6

Implémentation

Résumé : Dans ce chapitre, nous présentons notre maquette d'expérimentation qui nous a permis de tester notre approche de sécurisation des échanges dans un réseau, basée sur le concept du SDN, présentée dans le précédent chapitre, avec une description des outils utilisés dans nos différentes phases de mise en oeuvre. Concrètement, nous avons implémenté un réseau géré par un contrôleur de type OpenDaylight, un IDS snort [110] et la mise en place de la communication entre snort et OpenDaylight.

6.1 Introduction

Pour la mise en place de notre solution basée sur le SDN, nous avons uniquement travaillé en environnement virtuel. La virtualisation est une technique en informatique qui consiste à faire fonctionner plusieurs systèmes d'exploitation (invités) sur un ou plusieurs hôtes physiques. Elle permet de partager du matériel et des ressources. Il est possible de virtualiser un système complet (serveur ou poste client), une application, un service spécifique, un réseau (connexion entre machines virtuelles), les éléments du réseau (routeur, commutateur, etc.) et la sécurité.

Pour notre expérimentation, nous avons mis en place un réseau virtuel hébergé sur une plateforme de virtualisation de type VMware¹. La plateforme VMware permet à plusieurs machines virtuelles de fonctionner simultanément sur une machine physique et dispose d'une interface web donnant la possibilité à un administrateur de modifier, par exemple, la rapidité du processeur, la capacité mémoire, le stockage sur chaque machine virtuelle... L'accès à notre réseau virtuel se fait avec *vSphere* en accès sécurisé à distance par *Virtual Private Network* (VPN) avec authentification.

Notre expérimentation consiste en la mise en pratique de notre solution de sécurisation des échanges via un contrôleur centralisé, illustrée sur la figure 6.9. Cette architecture est composée

1. <https://www.vmware.com/> (Online : accessed on 14/12/2018)

d'un commutateur OpenFlow, d'un contrôleur OpenFlow, d'un IDS et de deux machines hôtes. Une machine hôte représente un ordinateur disposant d'une adresse logique et physique. Le réseau est à l'état initial, les tables de flux du commutateur OpenFlow sont vides. Les objectifs de ce réseau sont de permettre une communication entre deux machines hôtes à travers un commutateur OpenFlow et d'installer des politiques de sécurité sur cet équipement réseau via un contrôleur OpenDaylight.

6.2 Les outils utilisés

Pour mettre en oeuvre notre expérimentation en environnement virtuel, nous avons fait appel à plusieurs outils à savoir : un contrôleur OpenDaylight, un OVS, un IDS de type Snort, un Nmap, un qemu system et alpine linux. Nous avons utilisé des outils d'exportation graphique comme mRemoteNG, un outil de gestion de connexions à distance des machines virtuelles open source multi-onglets et multi-protocoles, et *Virtual Network Computing* (VNC) pour accéder plus facilement à notre réseau et à nos machines à distance via un OpenVPN permettant d'assurer la sécurité de la connexion.

6.2.1 Le contrôleur OpenDaylight

OpenDaylight est développé par ONF en collaboration avec plusieurs acteurs des réseaux comme Cisco, RedHat etc. Il nous a permis de programmer les fonctions de contrôle et de sécurité sur nos équipements réseaux via son *Southbound API* (Protocole OpenFlow).

La version stable Beryllium-SR4 d'OpenDaylight est utilisée dans nos travaux même si la dernière version, au moment où nous écrivons ces lignes, est *Fluorine-SR1*. Son architecture simplifiée est présentée dans la figure 6.1 ci-dessous. OpenDaylight a des API REST qui sont utilisées pour interagir avec les applications externes. Les applications existantes dans OpenDaylight utilisent la couche d'abstraction de service de l'anglais *Service Abstraction Layer* (SAL) pour communiquer avec différents types de périphériques, en utilisant les protocoles de communication tels que OpenFlow, Netconf, etc., pris en charge par OpenDaylight Beryllium-SR4. Le SAL expose la couche infrastructure aux applications situées au-dessus de celle-ci et détermine comment remplir les services demandés indépendamment du protocole sous-jacent utilisé et les périphériques réseaux. Il y a deux approches dans la programmation des applications OpenDaylight : *API-Driven SAL* (AD-SAL) et *Model-Driven SAL* (MD-SAL).

Le projet OpenDaylight héberge un nombre important de sous-projets (par exemple *open-flowplugin*, *controller*, *dlux*, *l2switch*, *yangtools*...) qui, lorsqu'ils sont combinés, fournissent une vaste gamme de fonctionnalités dans un seul package. Ces sous-projets sont disponibles sur

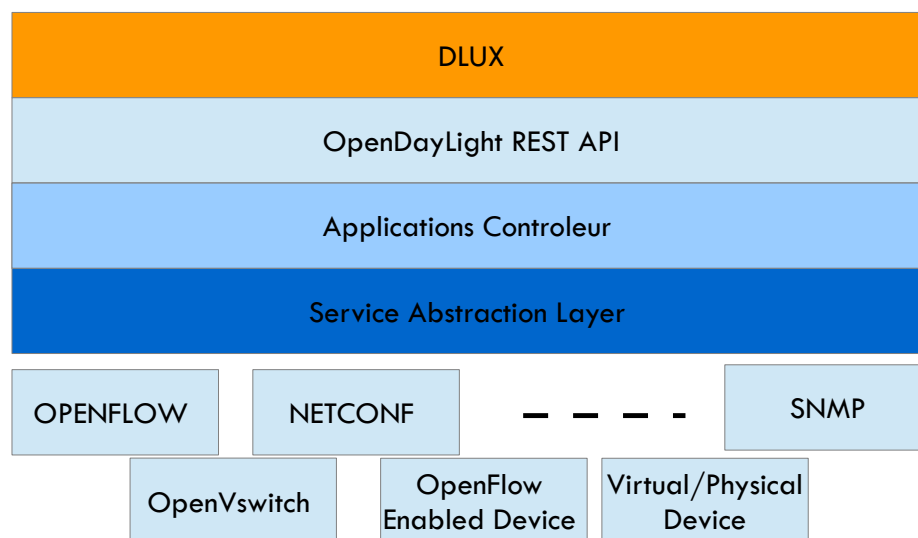


FIGURE 6.1 – Architecture simplifiée d'une plateforme OpenDaylight

GitHub², une plateforme d'hébergement pour les projets de développement logiciel.

La division des fonctionnalités en différents projets contribue à simplifier la conception du contrôleur. Si, par exemple, la spécification OpenFlow est mise à jour pour fournir des types d'actions plus récentes, le projet *openflowplugin* seul peut être modifié pour effectuer ce changement. Cette séparation des fonctionnalités est possible grâce à l'utilisation du *Open Service Gateway Interface* (OSGi).

Le contrôleur OpenDaylight est strictement implémenté dans le logiciel et est contenu dans sa propre machine virtuelle Java appelé *Java Virtual Machine* (JVM). Il peut être déployé sur n'importe quel matériel avec n'importe quel système d'exploitation prenant en charge Java. OpenDaylight s'appuie sur les technologies suivantes :

Maven : outil de gestion de projets qui simplifie et automatise les dépendances entre différents projets et qui aide les développeurs à gérer tous les plugins et dépendances requis pour une application. Il est hébergé par *Apache Software Foundation* et utilisé principalement pour les projets Java.

Java : langage de programmation utilisé pour développer des applications et des fonctionnalités dans OpenDaylight. Le développement en Java offre une sécurité et un moyen simple d'implémentation des services.

OSGi : plateforme de service modulaire utilisée pour les projets Java afin d'assurer la modularité et la réduction de la complexité du code et en favorisant la réutilisation du code. Dans OSGi, et comme l'indique la figure 6.2, les applications sont exécutées dans un conteneur qui fournit un framework de base pour la communication inter-applications, la gestion et l'exécution

2. <https://github.com/opendaylight>(Online : accessed on 14/12/2018)

des besoins modulaires.

Karaf : il existe plusieurs type de conteneurs OSGi tels que Equinox, Concierge,... mais OpenDaylight utilise Apache Karaf³, un sous-projet de Apache Felix, en tant que conteneur OSGi. Karaf fournit une exécution de commande similaire à celle de Linux. Un exemple de console karaf est présenté à la figure 6.10 dont les applications peuvent être installées et démarrées à partir de la ligne de commande présentée par Karaf.

YANG : utilisé par les développeurs pour modéliser une fonctionnalité d'application et générer des API à partir des modèles définis, ensuite utilisées pour fournir ses implémentations. YANG prend en charge la modélisation des données opérationnelles et de configuration, ainsi que des informations *Remote Procedure Call* (RPC) et des notifications. RPC est un protocole réseau permettant de faire des appels de procédures sur un ordinateur distant à l'aide d'un serveur d'applications.

MD-SAL : définit les fonctionnalités des messages et de stockage pour les données, les notifications et les RPC modélisés par les développeurs d'applications. Autrement dit, MD-SAL est le noyau du contrôleur OpenDaylight et gère les contrats et les échanges d'états entre chaque application. Il fait cette adaptation en gérant l'état centralisé. MD-SAL utilise YANG comme langage de modélisation.

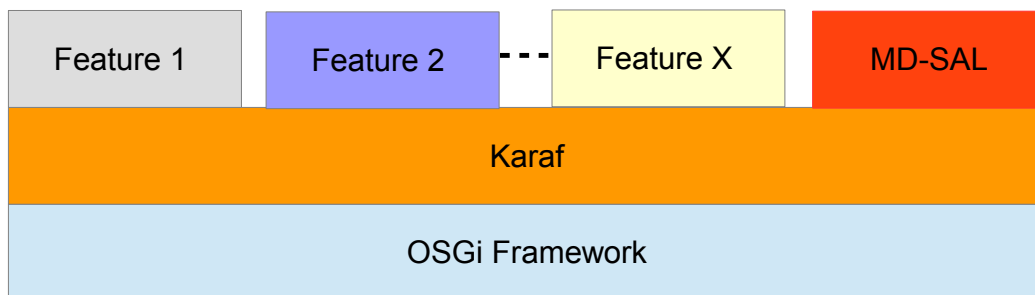


FIGURE 6.2 – Schéma théorique d'une plateforme OpenDaylight pour la programmation réseau

Pour utiliser OpenDaylight, il faut installer les fonctionnalités appelées *feature*, comme l'indique la figure 6.2, car l'installation de base ne permet d'effectuer aucune action. Par exemple, pour avoir accès à l'interface graphique permettant d'avoir un aperçu de l'architecture du réseau, il faut installer la fonctionnalité appelée *dlux*.

6.2.2 Open vSwitch (OVS)

L'OVS est une implémentation logicielle open source d'un switch Ethernet avec la particularité d'être multicouche et distribuée. Elle est conçue pour fonctionner comme un commutateur

3. <https://karaf.apache.org/> (Online : accessed on 20/12/2018)

dans les environnements de machines virtuelles. Son objectif est de fournir une fonction de routage niveau 2 et 3 du modèle OSI pour les environnements virtualisés supportant différents protocoles et standards. L'OVS supporte le protocole OpenFlow et fait intégralement partie du SDN. Dans nos travaux, elle nous a permis de faire communiquer des machines virtuelles Alpines Linux entre elles.

6.2.3 Création des machines virtuelles

Afin de créer nos machines virtuelles, nous avons utilisé les outils ci-dessous :

Qemu

Qemu (*Quick Emulator*)⁴ est un logiciel open source qui permet la virtualisation des machines et l'exécution d'instances de systèmes d'exploitations. Il s'exécute sur les architectures (x86, x64, ARM, PowerPC, SPARC, MIPS) et sur les systèmes d'exploitation (Linux, Mac OS X, Unix, Windows). Combiné au pilote *Kernel-based Virtual Machine* (KVM), une technologie de virtualisation Open Source intégrée aux noyaux Linux, il permet aussi de réaliser de l'accélération Hardware (HVM). L'outil de base `qemu-img` permet de créer et de gérer des images disques. En format local, les images disques peuvent se trouver en format binaire brut appelé raw (les images à ce format utilisent seulement l'espace nécessaire par rapport à la taille allouée) et en format `qcow2` dit "*copy-on-write*" qui intègre des fonctionnalités plus évoluées comme la compression, le chiffrement et les snapshots.

Alpine Linux

C'est une distribution ultra légère et sécurisée de Linux utilisée dans les environnements virtualisés. Elle est basée sur *musl libc*, une bibliothèque standard C destinée aux systèmes d'exploitations basés sur le noyau Linux, et *BusyBox*, un logiciel qui implémente un grand nombre des commandes standard sous Unix. Avec Alpine Linux un container n'a pas besoin de plus de 8 Mo de RAM et une installation minimale occupe environ 130 Mo d'espace disque.

6.2.4 Surveillance de trafic et gestion de flux malveillants

Wireshark, Nmap et Snort, en détail ci-dessous, nous ont servi respectivement dans les travaux d'analyses, de simulations des attaques et de détections d'un flux malveillant.

4. <https://www.qemu.org/>(Online : accessed on 18/12/2018)

Wireshark

Wireshark⁵ est un analyseur de paquets libres utilisé dans l'analyse et le dépannage de réseaux informatiques. Dans notre contexte, nous l'avons utilisé pour faire de l'investigation sur les messages OpenFlow échangés entre le commutateur OpenFlow et le contrôleur OpenDaylight.

Nmap

Nmap (*Network Mapper*)⁶ est un outil libre d'exploration de réseau, destiné principalement à l'audit de sécurité. Initialement conçu pour rapidement scanner de grands réseaux, il peut être aussi utilisé pour fonctionner sur une cible unique. Nmap est généralement utilisé pour les audits de sécurité mais de nombreux gestionnaires de systèmes et de réseaux l'apprécient pour des tâches de routine, comme les inventaires de réseau, la gestion des mises à jour planifiées ou la surveillance des hôtes et des services actifs.

Snort

Snort⁷ est un logiciel de détection et de prévention d'intrusion réseau open source. Il permet d'analyser le trafic réseau de type IP en temps réel et de détecter une grande variété d'attaques (scan de port, ...), grâce à sa capacité d'analyses de protocole et de recherche de correspondances de contenu. Actuellement maintenu par Sourcefire (racheté par Cisco en 2013), Snort a été à l'origine développé par Martin Roesch [110] en 1998. Très populaire et exploité par des millions d'utilisateurs de par le monde, Snort est considéré de facto comme un standard de fait. Snort est compatible avec plusieurs systèmes d'exploitation tels que Linux, Mac OS, FreeBSD, Open BSD, UNIX et Windows [111] et il est supporté par une grande communauté open source.

L'architecture Snort est composée de quatre principaux composants comme l'indique la figure 6.3 :

Le sniffer : ce composant permet d'écouter le trafic réseau. Le trafic le plus souvent écouté est le trafic IP mais il peut également s'agir d'autres protocoles de transfert.

Le preprocessor : à ce stade, les paquets capturés sont analysés et triés statistiquement en recherchant un certain type de comportement. Quand le comportement est déterminé, un paquet est envoyé au moteur de détection. À ce niveau, de nombreux et divers plugins peuvent être utilisés pour obtenir de meilleures performances.

La detection engine : une fois les paquets traités par le *preprocessor*, la *detection engine*

5. <https://www.wireshark.org/>(Online : accessed on 18/12/2018)

6. <https://nmap.org/>(Online : accessed on 18/12/2018)

7. www.snort.org(Online : accessed on 18/12/2018)

effectue l'action principale qui est de vérifier les données du paquet par rapport aux règles prédéfinies. S'il existe une correspondance entre les données et la règle, le paquet est envoyé au processeur d'alertes. Ce composant est la partie la plus importante de Snort.

Le output : dans le cas d'une correspondance au moteur de détection, une alerte est générée. Il peut être stocké dans des fichiers journaux / bases de données, envoyé par courrier électronique... Cela dépend des configurations de sortie activées.

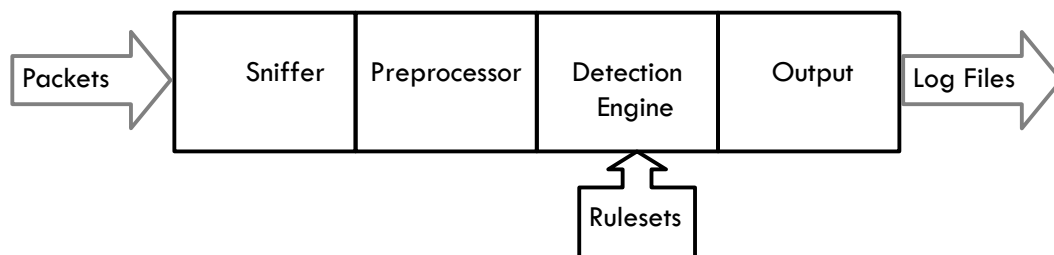


FIGURE 6.3 – Schéma de l'architecture Snort

Snort peut être configuré pour fonctionner principalement en trois modes opérationnels :

Le mode NIDS : dans ce mode, Snort agit comme un détecteur d'intrusion réseau en analysant le trafic réseau. Ensuite, il compare ce trafic à des règles déjà définies par l'administrateur réseau et établit des actions à exécuter ;

Le mode sniffer : c'est le mode qui permet à Snort de lire les paquets circulant sur le réseau et les afficher en continu sur un écran ou les enregistrer dans un fichier spécifique ;

Le mode packet logger : dans ce mode Snort journalise le trafic réseau dans des répertoires sur le disque. Dans nos travaux, le mode NIDS est le plus approprié puisque que nous allons utiliser Snort pour surveiller plusieurs interfaces réseaux. Ce qui nous permettra d'analyser le trafic sur l'ensemble des éléments du réseau, selon des règles spécifiques préalablement définies. Dans Snort, les règles sont définies comme des chaînes de texte brut composées de différentes parties (par exemple : action, protocole, adresse IP source et destination, ports source et de destination et options de messages et de règles pour les administrateurs). Cette flexibilité est l'une des principales raisons pour lesquelles nous avons choisi Snort, car il nous donne la possibilité d'écrire facilement des règles pour tout protocole si nous comprenons correctement son architecture et les vulnérabilités.

6.3 Mécanisme d'échanges de flux dans un réseau

Dans cette section, nous allons voir un bref aperçu du fonctionnement des échanges dans un réseau traditionnel et dans un réseau SDN. Ensuite, nous allons expliquer comment se fait la découverte de topologie dans un réseau SDN.

6.3.1 Mécanisme d'échanges dans un réseau traditionnel

Dans les réseaux traditionnels, l'acheminement de l'information est fait par les équipements réseaux (commutateurs et routeurs). Un commutateur est un équipement réseau qui permet de transférer des paquets dans un réseau. Il dispose de deux plans : le plan de contrôle et le plan de données. Le plan de données est responsable de l'acheminement des paquets de la source à la destination. Pour effectuer cette tâche, qui consiste à récupérer les paquets au niveau de l'interface d'entrée, le commutateur consulte alors sa table de routage pour déterminer l'interface de sortie. Le plan de contrôle est responsable d'associer une décision de routage aux paquets. Ce travail est soit fait par un protocole de routage, tels que *Border Gateway Protocol* (BGP), *Open Shortest Path First* (OSPF)...., soit par un administrateur réseau. Dans l'approche traditionnelle de gestion des réseaux, le plan de contrôle et le plan de données sont co-localisés sur le commutateur.

Comme l'indique la figure 6.4, dans un réseau classique IPv4 (Figure 4), le processus d'échanges de flux commence par une requête *ARP Request* de la machine h1 vers la machine h2, sachant que la machine h1 a une adresse MAC et une adresse IP et connaît l'adresse IP de la machine h2. En réponse à la machine h1, la machine h2 envoie une réponse *ARP Reply* pour déclarer à la machine h1 son adresse MAC. Après une mise à jour de la table de commutation, la machine h1 envoie le paquet à la machine h2.

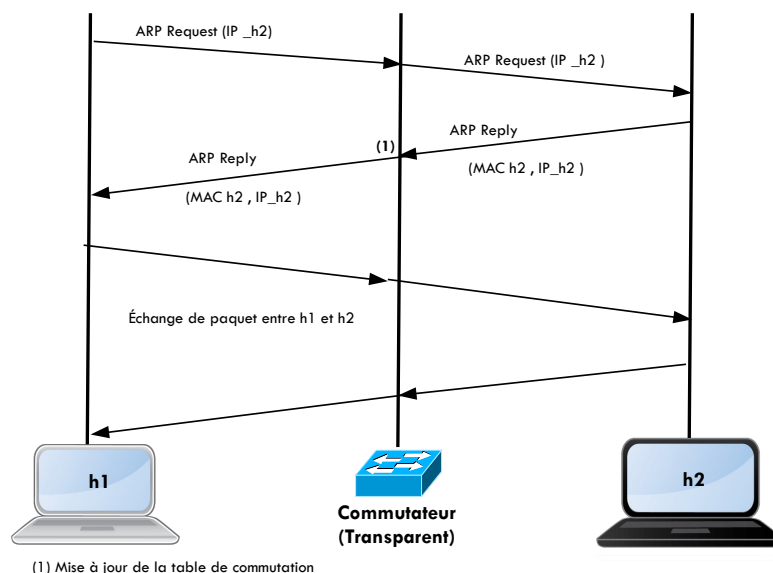


FIGURE 6.4 – Échanges de flux dans un réseau en IPv4

Cette manière de gérer les réseaux est certes résiliante et largement adoptée, mais très complexe à gérer et à faire évoluer.

6.3.2 Mécanisme d'échanges dans un réseau SDN

Afin d'expliquer plus précisément le fonctionnement d'un réseau SDN, nous présentons ci-dessous un exemple d'échanges de paquets entre deux machines. Nous avons utilisé seulement deux machines et un commutateur pour cette explication mais notre plateforme offre la possibilité de lancer plusieurs machines avec un ou plusieurs commutateurs. Lors d'un échange ICMP dans un réseau SDN avec un commutateur OpenFlow appelé Open vSwitch (OVS), deux machines h1 et h2 et un contrôleur OpenFlow de type OpenDaylight, comme l'indique la figure 6.5, les différentes phases d'échanges entre h1 et h2 sont :

1. Envoi de paquet de h1 vers l'OVS ;
2. L'OVS va recevoir le paquet ICMP de h1 et le transmet au contrôleur ;
3. Le contrôleur analyse le paquet reçu et installe le flux sur l'OVS ;
4. h2 reçoit le paquet ICMP de h1 ;
5. Échange de flux entre h1 et h2.

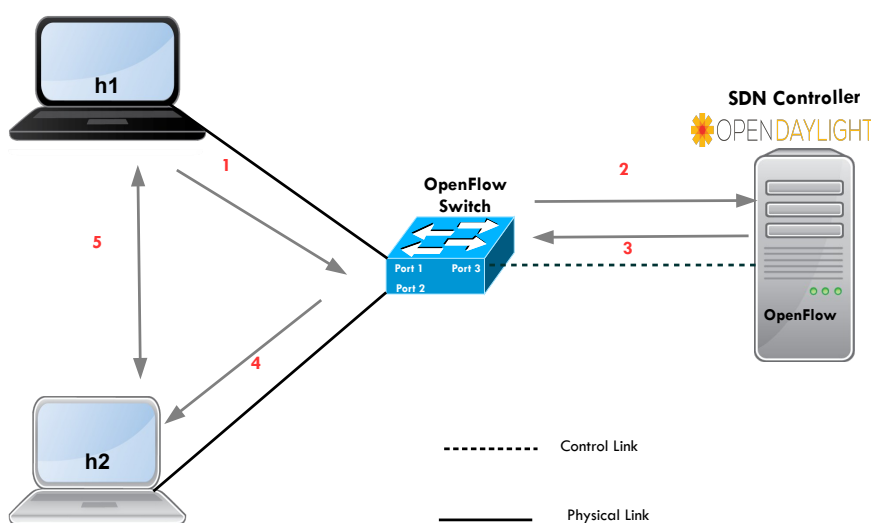


FIGURE 6.5 – Mécanisme d'échanges de flux dans un réseau OpenFlow

La figure 6.6 ci-dessous montre les types de messages OpenFlow échangés entre le contrôleur SDN et l'OVS. Ces messages sont capturés avec Wireshark, un logiciel d'analyse des paquets réseaux.

Si h1 ou h2 se retrouve compromise après l'installation du flux, ce défaut de sécurité risque de se propager sur la totalité du réseau. C'est pour cette raison que nous avons proposé dans le précédent chapitre une combinaison du SDN avec un IDS, pour améliorer la sécurité dans un réseau en fonction des événements détectés de manière automatisée.

No.	Time	Source	Destination	Protocol	Length	Info
32	64.000751	192.168.1.9	192.168.1.12	OpenFlow	82	Type: OFPT_HELLO
53	112.000803	192.168.1.9	192.168.1.12	OpenFlow	82	Type: OFPT_HELLO
90	162.093014	192.168.1.9	192.168.1.12	OpenFlow	98	Type: OFPT_FEATURES_REPLY
92	162.160260	192.168.1.12	192.168.1.9	OpenFlow	90	Type: OFPT_ROLE_REQUEST
93	162.160514	192.168.1.9	192.168.1.12	OpenFlow	90	Type: OFPT_ROLE_REPLY
95	162.165128	192.168.1.12	192.168.1.9	OpenFlow	74	Type: OFPT_BARRIER_REQUEST
96	162.165305	192.168.1.9	192.168.1.12	OpenFlow	74	Type: OFPT_BARRIER_REPLY
97	162.167858	192.168.1.12	192.168.1.9	OpenFlow	98	Type: OFPT_BARRIER_REQUEST
98	162.168039	192.168.1.9	192.168.1.12	OpenFlow	90	Type: OFPT_ROLE_REPLY
99	162.168116	192.168.1.9	192.168.1.12	OpenFlow	74	Type: OFPT_BARRIER_REPLY
101	162.177513	192.168.1.12	192.168.1.9	OpenFlow	82	Type: OFPT_MULTIPART_REQUEST, OFPMP_TABLE_FEATURES
102	162.177759	192.168.1.9	192.168.1.12	OpenFlow	94	Type: OFPT_ERROR
103	162.182588	192.168.1.12	192.168.1.9	OpenFlow	98	Type: OFPT_MULTIPART_REQUEST, OFPMP_DESC
104	162.182788	192.168.1.9	192.168.1.12	OpenFlow	274	Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_DESC
105	162.182974	192.168.1.9	192.168.1.12	OpenFlow	1138	Type: OFPT_MULTIPART_REPLY, OFPMP_DESC
107	162.261281	192.168.1.12	192.168.1.9	OpenFlow	82	Type: OFPT_MULTIPART_REQUEST, OFPMP_GROUP_FEATURES
108	162.261519	192.168.1.9	192.168.1.12	OpenFlow	94	Type: OFPT_ERROR
109	162.273315	192.168.1.12	192.168.1.9	OpenFlow	82	Type: OFPT_MULTIPART_REQUEST, OFPMP_METER_FEATURES
110	162.273522	192.168.1.9	192.168.1.12	OpenFlow	98	Type: OFPT_MULTIPART_REPLY, OFPMP_METER_FEATURES
112	162.322471	192.168.1.12	192.168.1.9	OpenFlow	194	Type: OFPT_FLOW_MOD
113	162.334773	192.168.1.12	192.168.1.9	OpenFlow	242	Type: OFPT_FLOW_MOD
115	162.700065	192.168.1.12	192.168.1.9	OpenFlow	78	Type: OFPT_SET_CONFIG
117	162.844049	192.168.1.12	192.168.1.9	OpenFlow	219	Type: OFPT_PACKET_OUT

FIGURE 6.6 – Messages OpenFlow échangés entre le contrôleur SDN et l'OVS

6.3.3 Découverte de topologie dans un réseau SDN

Le contrôleur tient une vue globale du réseau à travers son service de découverte de topologie. Cette fonctionnalité est importante pour que le SDN soit capable de gérer efficacement et apporter des services comme la commutation, le routage, le firewalling, ... Le réseau a aussi besoin d'une mise à jour efficace des informations sur la topologie. En réalité, un contrôleur n'a pas besoin de découvrir un noeud réseau puisque c'est le noeud qui initie la connexion avec le contrôleur et ainsi annonce son existence. En d'autres termes, il s'agit de découverte des liens. Un commutateur OpenFlow ne supporte pas toutes les fonctionnalités dédiées à la découverte des liens. Ce travail est réalisé par le contrôleur dans le cadre des réseaux SDN. Pour la découverte des liens, un contrôleur SDN s'appuie sur le standard *OpenFlow Discovery Protocol* (OFDP)[112, 113]. OFDP⁸ s'appuie sur le *Link Layer Discovery Protocol*(LLDP)⁹, et utilise le format de paquet LLDP. LLDP est un protocole standard (IEEE 802.1AB) de niveau 2 du modèle OSI de découverte de topologie implémentée sur les commutateurs Ethernet. Chaque trame LLDP est composée d'un en-tête *LLDP Data Unit* (LLDPDU). Un LLDPDU est composé des *Type-Length-Value* (TLV) servant à contenir les informations. Il est composé des TLV obligatoires et optionnels comme le montre la figure 6.7.

Pour la découverte des liens entre les commutateurs, OFDP crée des paquets LLDP pour

8. <http://groups.geni.net/geni/wiki/OpenFlowDiscoveryProtocol>(Online : accessed on 18/12/2018)

9. <https://www.ciscomadesimple.be/2014/07/30/lldp-link-layer-discovery-protocol/>(Online : accessed on 18/12/2018)

Preamble	Dst MAC	Src MAC	Ether-type : 0x88CC	CHASSIS ID TLV	PORT ID TLV	Time To Live TLV	Option TLVs	END TLV	Frame Check Seq.
----------	------------	------------	------------------------	----------------------	-------------------	------------------------	----------------	------------	---------------------

FIGURE 6.7 – Structure d'un LLDPDU

chaque port de chaque commutateur. Chaque paquet LLDP a un numéro de châssis (Châssis ID) et un numéro de port (*Port ID*).

Comme l'indique la figure 6.8, nous avons expérimenté un réseau avec deux commutateurs OpenFlow gérés par un contrôleur SDN de type OpenDaylight, pour observer la découverte des liens en SDN. Nous avons quatre phases :

1. Le contrôleur envoie un message Packet-OUT LLDP séparé à chaque port de l'OVS-1 avec les instructions correspondantes ;
2. L'OVS-1 envoie la trame LLDP à l'OVS-2
3. L'OVS-2 re-transmet le paquet LLDP au contrôleur via un message LLDP Packet-IN ;
4. Le message LLDP Packet-IN reçu par le contrôleur contient les métadonnées comme l'ID de l'OVS-2 et le port d'entrée où le paquet est reçu (Port 3) ;

En se référant aux données contenues dans la trame LLDP reçue, le contrôleur déduit l'existence d'une liaison entre l'OVS-1 et l'OVS-2 et met à jour sa topologie. Ce cycle se répète toutes les 5 secondes.

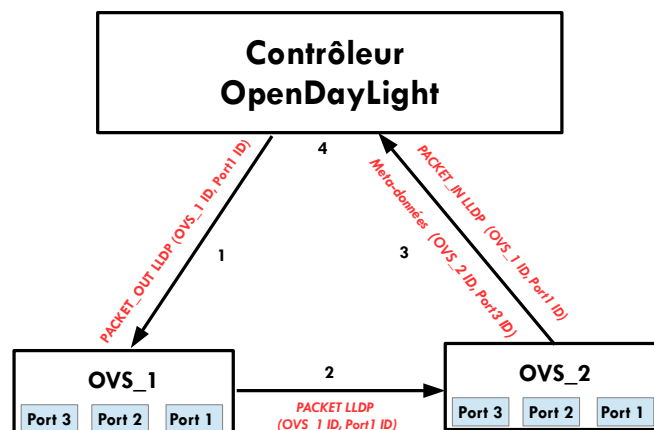


FIGURE 6.8 – Découverte des liens avec OFDP

6.4 Implémentation

Comme l'indique la figure 6.9 ci-dessous, notre maquette d'implémentation est réalisée entièrement en environnement virtuel avec des outils open source.

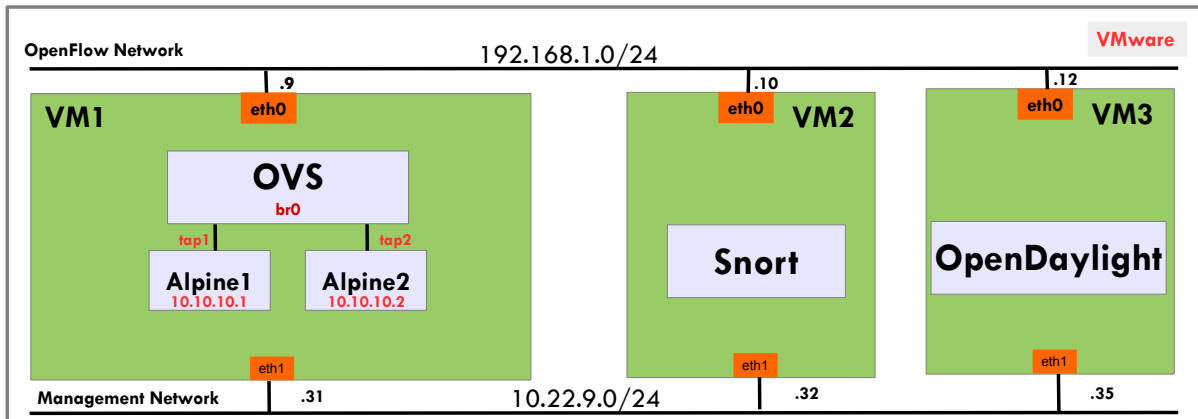


FIGURE 6.9 – Maquette de mise en oeuvre de sécurisation d'un réseau avec du SDN

6.4.1 Installation d'OpenDaylight

Le contrôleur OpenDaylight est un logiciel de système d'exploitation réseau open source développé en Java et géré par la fondation Linux. Il est basé sur une architecture modulaire et échange avec les applications du SDN en utilisant les *Northbound* API. OpenDaylight communique avec les équipements réseaux avec sa *Southbound* API. La *Southbound* API la plus souvent utilisée dans le SDN est OpenFlow.

Pour expérimenter notre solution, nous avons créé une machine virtuelle de 2 CPU et 16GB de RAM avec un système d'exploitation Ubuntu 16.04 sur la plateforme VMware. Ensuite nous avons installé sur cette machine un contrôleur SDN de type OpenDaylight version Beryllium-SR4.

Pour l'installer, nous avons téléchargé OpenDaylight version Beryllium-SR4 à partir du lien <https://www.opendaylight.org/downloads>. Avec des privilèges administrateur, OpenDaylight peut être initialisé en exécutant le script *karaf.sh* à partir de la ligne de commande, comme l'indique la figure 6.10.

Toutes les distributions OpenDaylight sont livrées sans aucune fonctionnalité activée par défaut. En revanche, toutes les fonctionnalités sont disponibles pour l'installation.

Cette version d'OpenDaylight, basée sur la plate-forme Java, nécessite au minimum *Java 7 JDK* installé sur la machine contrôleur. Pour définir la variable d'environnement JAVA-

HOME, il est nécessaire d'ajouter la ligne suivante dans le fichier de base : *export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdkamd64*.

Les fonctionnalités disponibles peuvent être installées avec la commande *feature :install jfeaturej*.



```

root@ubuntu-LTS:~/distribution-karaf-0.4.4-Beryllium-SR4/bin# ./karaf
OpenJDK 64-Bit Server VM warning: ignoring option MaxPermSize=512m; support was removed in 8.0

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown OpenDaylight.

opendaylight-user@root>

```

FIGURE 6.10 – Ecran de lancement OpenDaylight

Le répertoire OpenDaylight Beryllium-SR4 est constitué de plusieurs répertoires qui sont :

1. configuration ;
2. lib contenant les librairies java ;
3. plugins-ODL OGSi plugins directory ;
4. karaf.bat pour démarrer ODL sous Windows ;
5. karaf.sh pour démarrer ODL sous Linux ;
6. version.properties pour spécifier le numéro de la version.

Une fois OpenDaylight installé, nous avons ajouté les fonctionnalités telles que *odl-l2switch-switch*, *odl-dlux-all* et *odl-restconf* pour supporter les commutateurs de niveau 2/3, l'interface web et communiquer avec les applications via l'API REST. Il est aussi important d'activer la version OpenFlow 1.3 en ajoutant l'option *-of13* sur le fichier du script de lancement, car la version 1.0 de OpenFlow est par défaut implémentée sur le contrôleur OpenDaylight. OpenDaylight fournit plusieurs types de fonctionnalités à utiliser selon le besoin.

Pour prendre une décision d'acheminement de niveau 2/3 du modèle OSI, le contrôleur OpenDaylight connaît la topologie du réseau, ainsi que les équipements qui sont connectés avec leurs identifiants (adresses IP et adresses MAC). En utilisant OpenFlow 1.3, le contrôleur OpenDaylight configure un commutateur OVS, gère et met à jour le réseau OpenFlow.

Le lien *http ://@IP du serveur contrôleur :8181/index.html* permet d'accéder à OpenDaylight via une interface de navigation Web. Les accès par défaut (identifiant et mot de passe) de

l'interface graphique du contrôleur sont *admin/admin*. Ces identifiants peuvent être personnalisés à partir de cette même interface. La figure 6.11 ci-dessous montre l'environnement graphique d'OpenDaylight version Beryllium-SR4.

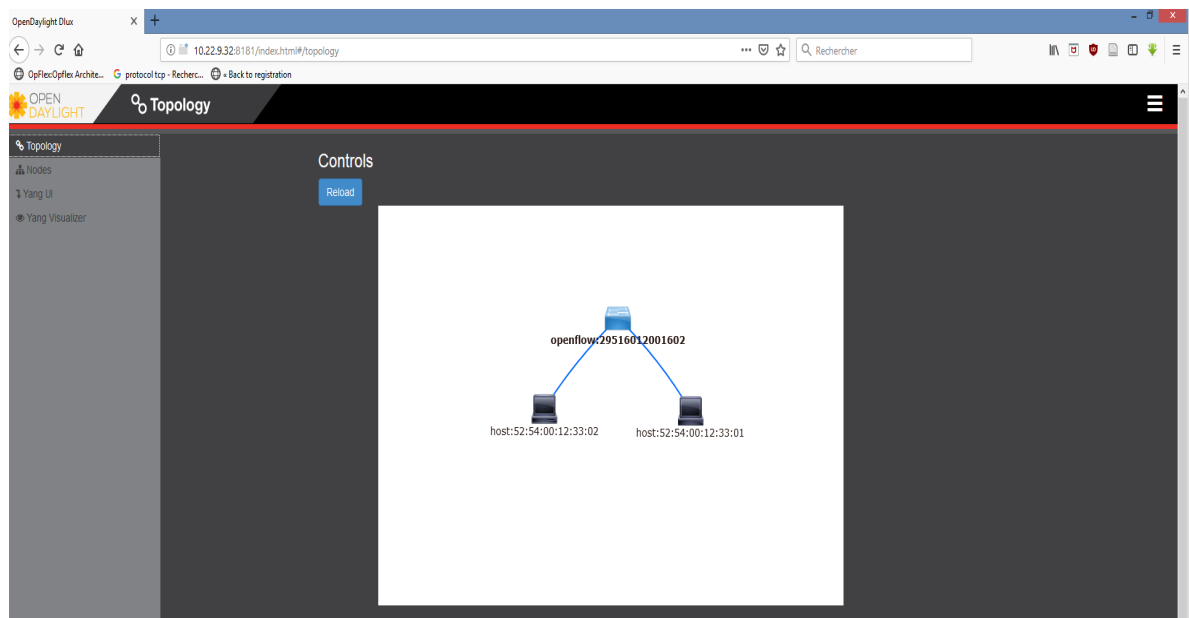


FIGURE 6.11 – Interface graphique d’OpenDaylight Beryllium-SR4

Il est aussi possible de visualiser l’ensemble des interfaces avec leurs identifiants (numéro port, nom, adresse mac, ...), comme l’indique la figure 6.12 ci-dessous.

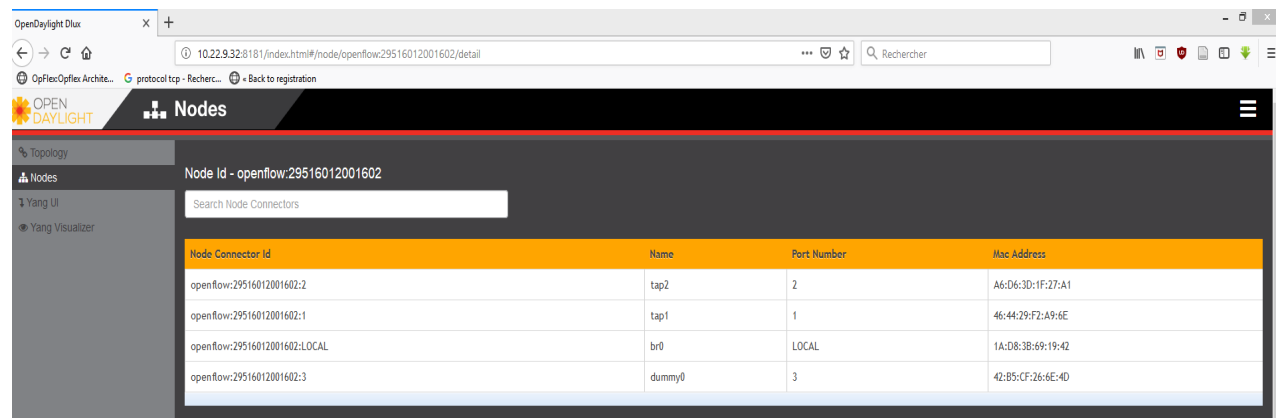


FIGURE 6.12 – Inventaire des noeuds sur l’interface graphique d’OpenDaylight Beryllium-SR4

Après cette étape d’installation du logiciel contrôleur, nous allons, dans le paragraphe suivant, simuler le réseau OpenFlow à gérer par OpenDaylight.

6.4.2 Réalisation de l'architecture

La plupart des travaux dans la littérature utilisent le simulateur de réseau mininet pour expérimenter le réseau SDN. Nous avons fait le choix d'utiliser des machines virtuelles dans un environnement de production sur plateforme VMvare, afin d'être dans un cas réel d'usage.

Pour réaliser notre architecture de réseau virtuel, nous avons créé une deuxième machine virtuelle avec un système d'exploitation Ubuntu 16.04, 2 CPU virtuel et 16GB de RAM sur une plateforme VMvare. Sur cette machine, nous avons installé un commutateur virtuel compatible OpenFlow 1.3 (OVS version 2.6.0) et Qemu, un émulateur open source de machine virtuelle sur architecture x86.

L'OVS est une implémentation logicielle open source d'un commutateur Ethernet avec la particularité d'être multicouche et distribué. Il est conçu pour fonctionner comme un commutateur de niveau 2/3 du modèle OSI dans les environnements des machines virtuelles supportant différents protocoles et standards, dont le protocole OpenFlow. Dans nos travaux, il nous a permis de faire communiquer les machines virtuelles clientes entre elles. Qemu est utilisé pour émuler nos machines clientes avec un système d'exploitation Alpine Linux, une distribution ultra légère de Linux avec 48MB de RAM. Nous avons utilisé l'outil de base qemu-img pour créer et gérer les images disque. Le format qcow2 est utilisé dans ce travail car il intègre des fonctionnalités plus évoluées comme la compression et le chiffrement.

Ensuite, nous avons écrit un script bash pour lancer plusieurs machines virtuelles clientes qemu avec la possibilité de les gérer à distance. Le même script permet de lancer l'OVS pour interconnecter les machines virtuelles Alpine Linux entre elles et créer la liaison entre le commutateur OpenFlow et le contrôleur OpenDaylight, pour permettre à ce dernier de contrôler le réseau via le protocole OpenFlow 1.3. Une attribution dynamique des adresses IPv4 en DHCP des machines virtuelles clients du réseau est faite par le même code. C'est ainsi que nous avons mis en place notre réseau OpenFlow avec la possibilité de monter à l'échelle en variant juste le nombre de machines virtuelles et d'OVS souhaités.

6.4.3 Sécurisation du lien entre le contrôleur OpenDaylight et l'OVS

Comme nous l'avons évoqué précédemment dans la partie état de l'art, le canal de communication entre l'OVS et le contrôleur OpenDaylight n'est pas chiffré par défaut, ce qui veut dire que le cryptage des messages d'échanges OpenFlow entre ces deux éléments du réseau SDN ne s'exécute pas automatiquement. De plus, certains contrôleurs ne supportent même pas le support du TLS pour le chiffrement des communications entre le commutateur SDN et le contrôleur. Un hacker peut exploiter ce manque de sécurité sur le canal OpenFlow pour attaquer le réseau et mener des actions malveillantes. Ce qui est extrêmement dangereux si

le hacker obtient un accès au contrôleur qui lui permettrait d'avoir le contrôle de l'ensemble du réseau. Avec une prise en main du contrôleur, le hacker peut supprimer les commutateurs OpenFlow, modifier les règles OpenFlow dans le commutateur, capturer le trafic sensible et surveiller la manière dont le contrôleur gère les paquets OpenFlow. C'est pourquoi, il faut chiffrer en SSL/TLS les échanges des messages OpenFlow sur le canal entre l'OVS et l'OpenDaylight. Le chiffrement des messages OpenFlow entre l'OVS et l'OpenDaylight est crypté à l'aide d'une connexion SSL/TLS, basée sur le modèle d'infrastructure à clé publique appelée *Public Key infrastructure* (PKI).

En utilisant la boîte à outils de chiffrement OpenSSL, nous avons généré un *keyStore*, un fichier contenant les clés privées et publiques du contrôleur. Ensuite, le fichier de clés est importé dans un fichier de clés au format JKS, adapté pour être configuré sur le fichier de configuration OpenFlow du contrôleur OpenDaylight comme l'indique la figure 6.13.

```
<module>
  <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:openflow:switch:connection:provider:impl">prefix:openflow-switch-connection-provider-impl</type>
  <name>openflow-switch-connection-provider-default-impl</name>
  <port>6633</port>
  <switch-idle-timeout>15000</switch-idle-timeout>
  <transport-protocol>TLS</transport-protocol>
  <tls>
    <keystore>configuration/ssl/ctl.jks</keystore>
    <keystore-type>JKS</keystore-type>
    <keystore-path-type>PATH</keystore-path-type>
    <keystore-password>opendaylight</keystore-password>
    <truststore>configuration/ssl/truststore.jks</truststore>
    <truststore-type>JKS</truststore-type>
    <truststore-path-type>PATH</truststore-path-type>
    <truststore-password>opendaylight</truststore-password>
    <certificate-password>opendaylight</certificate-password>
    <cipher-suites>TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384</cipher-suites>
    <cipher-suites>TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384</cipher-suites>
    <cipher-suites>TLS_DHE_DSS_WITH_AES_256_GCM_SHA384</cipher-suites>
    <cipher-suites>TLS_DHE_RSA_WITH_AES_256_GCM_SHA384</cipher-suites>
  </tls>
</module>
<module>
  <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:openflow:switch:connection:provider:impl">prefix:openflow-switch-connection-provider-impl</type>
  <name>openflow-switch-connection-provider-legacy-impl</name>
  <port>6653</port>
  <switch-idle-timeout>15000</switch-idle-timeout>
  <transport-protocol>TLS</transport-protocol>
  <tls>
    <keystore>configuration/ssl/ctl.jks</keystore>
    <keystore-type>JKS</keystore-type>
    <keystore-path-type>PATH</keystore-path-type>
    <keystore-password>opendaylight</keystore-password>
    <truststore>configuration/ssl/truststore.jks</truststore>
    <truststore-type>JKS</truststore-type>
    <truststore-path-type>PATH</truststore-path-type>
    <truststore-password>opendaylight</truststore-password>
    <certificate-password>opendaylight</certificate-password>
    <cipher-suites>TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384</cipher-suites>
    <cipher-suites>TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384</cipher-suites>
    <cipher-suites>TLS_DHE_DSS_WITH_AES_256_GCM_SHA384</cipher-suites>
    <cipher-suites>TLS_DHE_RSA_WITH_AES_256_GCM_SHA384</cipher-suites>
  </tls>
</module>
```

FIGURE 6.13 – Fichier de configuration OpenFlow pour supporter TLS

Pour créer la clé publique, la clé privée et le certificat pour l'OVS, la commande *ovs-pki req + signe sc* permet de créer les fichiers de l'autorité de certification (CA).

L'utilisation de la commande *ovs-vsctl set-ssl* pour pointer sur le répertoire des clés publiques et privées générées précédemment, permet de configurer l'OVS avec cryptage SSL. En outre, il est nécessaire d'ajouter le certificat de contrôleur *cacert.pem* qui autorise la connexion au contrôleur ODL en vérifiant la signature de ce certificat (CA). Ainsi, le chiffrement en SSL/TLS de la connexion entre OVS et le contrôleur OpenDaylight fonctionne dans les deux sens. D'où la sécurité du canal entre les deux éléments du réseau SDN. En revanche, les communications SSL/TLS sont susceptibles d'être interceptées et modifiées via une attaque de type man-in-the-middle en cas de fuite de la clé privée. C'est pourquoi, il est important de conserver toutes les clés privées dans un espace sécurisé, car en disposant de la clé privée correspondante, une personne malintentionnée et ayant les compétences techniques peut déchiffrer un message crypté avec la clé privée.

6.4.4 Mise en place de Snort

Afin de détecter les menaces, nous avons utilisé un IDS de type Snort. C'est un logiciel de détection d'intrusion réseau open source qui permet d'analyser le trafic réseau de type IP en temps réel et de détecter une grande variété d'attaques (scan de port...), grâce à sa capacité d'analyse de protocole et de recherche de correspondance de contenu.

Dans nos travaux, nous avons utilisé Snort en mode NIDS, approprié pour surveiller plusieurs interfaces réseau. Dans ce mode, Snort agit comme un détecteur d'intrusion réseau en analysant le trafic réseau et compare ce trafic à des règles déjà définies par l'administrateur réseau, pour établir des actions à exécuter. Les règles sont définies comme des chaînes de texte brut composé de différentes parties (par exemple : action, protocole, adresse IP source et destination, ports source et de destination, et options de messages et de règles pour les administrateurs).

Pour déployer Snort, nous avons opté pour une architecture centralisée avec un seul Snort qui surveille le trafic du réseau sur un port particulier et ensuite nous avons mis en place un port mirroring de tous les ports de notre réseau vers le port particulier qui sera analysé constamment par Snort. Le choix de cette architecture centralisée avec un Snort, est motivé par sa simplicité de mise en oeuvre, mais qui a l'inconvénient d'avoir des goulets d'étranglement en cas de montée en charge du réseau.

C'est pourquoi nous avons prévu de mettre en place une architecture avec plusieurs instances de Snort dans nos futurs travaux, que nous avons évoqué dans le précédent chapitre, avec une ébauche d'architecture.

Pour intégrer Snort dans le réseau, nous avons créé une troisième machine virtuelle avec un système d'exploitation Ubuntu 16.04, 2 CPU virtuel et 16 GB de RAM sur laquelle nous avons installé le logiciel Snort version 2.9.11. La figure 6.9 montre l'intégration de Snort dans notre

réseau SDN.

Après l'installation de Snort, nous avons défini quelques règles non exhaustives pour générer des logs pour toute tentative de requête ICMP de type echo request et echo reply, de scan de ports et d'usurpation d'adresse IP source ou MAC. Les informations sur les alertes générées par Snort sont enregistrées sous forme de fichier logs dans un répertoire sur le serveur hébergeant Snort.

6.4.5 Liaison de Snort avec OpenDaylight

Après avoir mis en place un réseau géré par un contrôleur OpenDaylight puis intégré Snort pour surveiller le réseau, nous avons écrit une application qui permet d'extraire et d'analyser les informations sur les logs générés par Snort et d'envoyer une règle de sécurité au contrôleur OpenDaylight, pour désinstaller un flux malveillant via l'API REST en temps réel. L'API REST permet l'échange d'informations d'une manière simple entre un client et un serveur basé sur le protocole HTTP. Elle est aujourd'hui utilisée par la plupart des contrôleurs SDN pour échanger des informations réseaux avec les applications. Pour prendre en charge l'API REST, nous avons ajouté la fonctionnalité *odl-restconf* lors de l'installation de OpenDaylight.

Ainsi, notre application échange avec OpenDaylight à travers l'API REST pour isoler la machine à l'origine de la menace, par un shutdown du port par exemple. Les opérations CRUD (*Create, Read, Update, Delete*) ont été mises en place grâce aux méthodes POST, GET, PUT et DELETE de l'API REST pour créer, récupérer, modifier et supprimer un flux spécifique sur le contrôleur OpenDaylight. Par exemple, ce lien suivant sur le navigateur web : *GET http://10.22.9.32:8181/restconf/operational/.opendaylight-inventory:nodes* permet de lister les noeuds.

Pour modifier la configuration sur OpenDaylight, PUT et DELETE nous ont permis de modifier ou de supprimer un flux spécifique sur le contrôleur. La figure 6.9 montre la maquette d'intégration de Snort et d'OpenDaylight sur un réseau SDN.

6.4.6 Analyse, détection et suppression de flux

Nous implémentons notre solution en environnement virtuel avec un contrôleur SDN OpenDaylight version Beryllium-SR4 utilisant le protocole OpenFlow 1.3, un commutateur virtuel compatible OpenFlow (OVS) version 2.6.0, deux machines Alpine Linux avec 48 Mo de mémoire RAM et un IDS open source Snort version 2.9.11.

Pour prendre une décision d'acheminement de niveau 2 du modèle OSI, le contrôleur OpenDaylight connaît la topologie du réseau, ainsi que les équipements qui sont connectés avec leurs identifiants (adresses IP et adresses MAC). En utilisant OpenFlow 1.3, le OpenDaylight pro-

Algorithme 1 Lancement du réseau OpenFlow

```

1: img = Alpine_img
2: path_img = /path/to/img
3: mem = {48 MB}
4: N = {Nombre de VMs à démarrer}
5: OVSi = OVSnom
6: y = 0
7: while  $1 + y \leq N$  do
8:   tap = tap(1+y)
9:   VMname = Alpine_(1+y)
10:  new_img = /path/to/save/img
11:  mac = 00 :00 :00 :00 :00 :(1+y)
12:  port = (5900+y)
13:  qemu crée une image au format qcow2 dans new_img
14:  qemu démarre les VM avec {mem, new_img, VMname, mac, port}
15:  if OVSi = NULL then
16:    ajout bridge{OVS}
17:    set OF version to bridge{OVS, protocols=OpenFlow13}
18:    ajout port{OVS, tap}
19:  else
20:    ajout port{OVS, tap}
21:  end if
22:  y = y + 1
23: end while

```

gramme le commutateur OVS, gère et met à jour le réseau OpenFlow. Le commutateur OVS agit comme un commutateur de niveau 2.

Le contrôleur OpenDaylight tourne sur une machine virtuelle Vmware avec un système d'exploitation Ubuntu 14.04, 2 CPU virtuels et 16GB de RAM, sur le même réseau que les deux autres machines ayant les mêmes caractéristiques qui hébergent l'IDS Snort et les machines virtuelles Alpine Linux.

Pour collecter, analyser, détecter et supprimer un flux malveillant, nous avons écrit un script avec les commandes de l'OVS et de Snort. Le code exécute dans un premier temps une commande OVS qui permet de collecter tout le trafic échangé entre les machines Alpine Linux vers un port particulier. Ensuite, ce port est scanné par l'IDS Snort qui enverra un message d'alerte en cas de flux malveillants au contrôleur OpenDaylight, dont le rôle est de surveiller et de sécuriser efficacement le réseau contre les menaces internes et externes. Ce dernier utilise des messages OpenFlow 1.3 pour envoyer une règle de flux à l'OVS pour interdire le flux malveillant. Tout cela se fait de manière automatisée. L'OVS gérera tout autre flux de même type sans passer par OpenDaylight jusqu'à la prochaine règle ou mise à jour du contrôleur.

D'une manière générale on peut définir plusieurs types d'alerte sur le Snort. Dans notre

expérimentation, nous avons défini sur Snort une alerte pour toute tentative de requête ICMP.

L'écran de la figure 6.15 montre le résultat de l'exécution de notre code pour désinstaller un flux sur l'OVS.

La commande `ovs-ofctl dump-flows` nous a permis d'afficher la table de flux sur l'OVS, comme l'indique la figure 6.14, avant de lancer notre script. Sur la partie encadrée en rouge sur le premier écran, nous constatons que le flux pour interconnecter nos deux machines est bien installé. Ce flux a été ajouté par le contrôleur OpenDaylight lors du lancement de notre requête ICMP.

```

1 Ecran 1
2 root@root:~# sudo ovs-ofctl -OOpenFlow13 dump-flows br0
3
4 OFFST_FLOW reply (OF1.3) (xid=0x2):
5
6 cookie=0x2a000000000000001, duration=8.001s, table=0, n_packets=0, n_bytes=0, priority=100,d_type=0x88cc actions=CONTROLLER:65535
7
8 cookie=0x2a000000000000000, duration=8.001s, table=0, n_packets=8, n_bytes=784, idle_timeout=300, hard_timeout=600, priority=10,d_src=52:54:00:12:33:01,d_dst=52:54:00:12:33:02 actions=output 2
9
10 cookie=0x2a000000000000000, duration=8.001s, table=0, n_packets=8, n_bytes=784, idle_timeout=300, hard_timeout=600, priority=10,d_src=52:54:00:12:33:02,d_dst=52:54:00:12:33:01 actions=output 1
11
12 cookie=0x2b000000000000001, duration=8.893s, table=0, n_packets=2, n_bytes=140, priority=2,in_port=2 actions=output:3,output:1,CONTROLLER:65535
13
14 cookie=0x2b000000000000000, duration=8.893s, table=0, n_packets=0, n_bytes=0, priority=2,in_port=3 actions=output:2,output:1,CONTROLLER:65535
15
16 cookie=0x2b000000000000000, duration=8.893s, table=0, n_packets=3, n_bytes=238, priority=2,in_port=1 actions=output:3,output:2,CONTROLLER:65535
17
18 cookie=0x2b000000000000001, duration=11.896s, table=0, n_packets=3, n_bytes=294, priority=0 actions=drop

```

FIGURE 6.14 – Ecran d'affichage de flux installés sur l'OVS

En lançant notre code, nous constatons sur le deuxième écran de la figure 6.15 que notre système de sécurisation des échanges dans un réseau en fonction des événements détectés a bien été déclenché avec la désinstallation du flux malveillant.

```
Ecran 2
```

```
1 root@root:~# script.sh
2 Malicious flow detection !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
3 Malicious flow removal !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
4 In_port=1
5 dl_src=52:54:00:12:33:01,dl_dst=52:54:00:12:33:02
6 OpenP flow info to OVS !!!!!!
7 REFID_FLOW reply (OF1.3) (xid=0x2):
8 cookie=0xb2b00000000000001, duration=78.03ds, table=0, n_packets=0, n_bytes=0, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
9 cookie=0xb2b00000000000000, duration=74.13ss, table=0, n_packets=79, n_bytes=7518, idle_timeout=300, hard_timeout=600, priority=10,dl_src=52:54:00:12:33:02,dl_dst=52:54:00:12:33:01 actions=output:1
10 cookie=0xb2b000000000000001, duration=76.027fs, table=0, n_packets=12, n_bytes=3560, priority=2,in_port=2 actions=output:3,output:1,CONTROLLER:65535
11 cookie=0xb2b000000000000000, duration=76.027fs, table=0, n_packets=0, n_bytes=0, priority=2,in_port=3 actions=output:2,output:1,CONTROLLER:65535
12 cookie=0xb2b0000000000000001, duration=78.03ds, table=0, n_packets=3, n_bytes=294, priority=0 actions=drop
```

FIGURE 6.15 – Résultats de l'écran d'exécution d'alerte et de désinstallation automatisée de flux malveillants

Comme l'indique la figure 6.9, notre maquette d'implémentation est réalisée entièrement avec des outils open source et en environnement de production sur des serveurs virtualisés sous Vmware.

Nous avons aussi développé un script pour automatiser notre solution pour exécuter la collecte des flux avec un port mirroring, d'analyser, de détecter et d'alerter le contrôleur OpenDaylight des flux malveillants. Le code permet aussi au contrôleur OpenDaylight, une fois l'alerte reçue, d'agir pour supprimer le flux non désiré en envoyant à l'OVS une règle de suppression de flux, selon la procédure de Algorithm2 ci-dessous.

Comme indiqué sur la capture de la figure 6.16 ci-dessous, nous avons ajouté seulement un seul flux pour le besoin d'explication, pour voir si notre code exécute bien l'algorithme. Nous constatons à l'exécution de la dernière commande de lecture de l'OVS que le flux a bien été supprimé.

Algorithme 2 Détection et suppression de flux malveillants

```

1: AddressMacSrc
2: AddressMacDest
3: AddressIpSrc
4: AddressIpDest
5: Bridge
6: IDSinterface
7: FluxMalveillant=" AddressMacSrc, AddressMacDest, In-port, type, AddressIpSrc, Address-
  sIpDest, actions"
8: Snort -A Console -q -c path/to/snort/conf -i IDSinterface
9: ovs-ofctl add-flow Bridge FluxMalveillant
10: while Vrai do
11:   if FluxMalveillant présent dans la table de flux de l'OVS then
12:     ovs-ofctl del-flows Bridge FluxMalveillant
13:   else
14:     Échange de flux normal
15:   end if
16: end while

```

```
root@ubuntu-LTS:~# sh ids.sh
Etape 1: Ajout d'un flux malveillant sur l'OVS!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

OFPST_FLOW reply (OF1.3) (xid=0x2):
    cookie=0x0, duration=0.005s, table=0, n_packets=0, n_bytes=0, icmp_in_port=1,dl_src=52:54:00:12:33:01,dl_dst=52:54:00:12:33:02,nw_src=10.10.10.1,nw_dst=10.10.10.2 actions=output:2
Etape 2: Lecture de l'alerte Snort !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Etape 3: Suppression du flux malveillant !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Etape 4: Lecture de l'OVS pour voir si le flux a été bien supprimé !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

OFPST_FLOW reply (OF1.3) (xid=0x2):
root@ubuntu-LTS:~#
```

FIGURE 6.16 – Ecran d'exécution de l'algorithme

Au cours de ces expériences nous nous sommes particulièrement intéressés à un système de surveillance de trafic réseau pour détecter, analyser et isoler les comportements anormaux via une application. Cette application communique avec OpenDaylight via l'API REST. Le système mis en place permet la détection d'intrusions avec Snort, qui génère un fichier logs à chaque tentative d'intrusion. Ensuite, notre application extrait, lit, analyse et envoie une règle de sécurité à OpenDaylight pour isoler tout flux malveillant. Des stratégies d'accès sécurisé et des informations de connexion peuvent être mises en oeuvre sur OpenDaylight pour renforcer la sécurité. Il existe plusieurs types d'attaque comme rançongiciel, logiciel malveillant, zero day, ..., mais nous nous sommes concentrés sur la détection d'intrusions avec Snort et la mise en oeuvre de l'application qui permet d'extraire, de lire, d'analyser les logs et d'envoyer une instruction à OpenDaylight pour isoler un flux malveillant dynamiquement via l'API REST.

6.5 Quelques exemples d'attaques simulées

Pour simuler une attaque et voir si Snort la détecte ou pas, nous avons installé sur une des machines virtuelles clientes l'outil Nmap. Ensuite, nous avons successivement lancé une attaque de déni de service, de scan de port et une usurpation d'adresse IP de la machine avec la commande Nmap spécifique au cas par cas.

6.5.1 Déni de services

L'objectif ici est de détecter et de bloquer les tentatives de saturation d'une machine cible par des attaques de type DoS avec le protocole ICMP. Nous avons procédé à l'envoi des requêtes ICMP à la deuxième machine de notre réseau, afin de voir si Snort a réagi en détectant les flux non désirés. Avec cet exemple, nous avons constaté que suite à cette requête ICMP, notre IDS Snort a détecté et enregistré un fichier log sur le répertoire spécifique du serveur Snort.

Ce type de tentative d'attaque peut rendre le contrôleur ou une machine indisponible pour ses utilisateurs. Il interrompt ou suspend temporairement ou indéfiniment les services du réseau. Avec la solution proposée, il est possible de bloquer la communication des noeuds malveillants de façon automatisée.

6.5.2 Scan de ports

Il s'agit dans ce cas de détecter les tentatives de scan de ports sur les protocoles TCP et UDP et de bloquer ces requêtes provenant d'une même source avec l'outil Nmap. Nmap est un logiciel de scan de ports open source conçu pour détecter les ports ouverts et, plus généralement, obtenir des informations sur le système d'exploitation d'un ordinateur distant. Pour connaître les ports ouverts sur une machine, Nmap envoie un paquet sur tous les ports de la machine cible et analyse les réponses.

Pour simuler le scan de ports, nous avons installé l'outil Nmap sur une des machines host alpine Linux de notre réseau. Ensuite, nous avons lancé un balayage de ports sur une des machines du réseau avec la commande spécifique (*nmap -p "*" Adresse Ip machine cible*) et de la même manière, Snort a détecté cette tentative d'attaque et a enregistré le log correspondant.

6.5.3 Usurpation d'adresse IP ou MAC

Cette usurpation est faite lorsqu'un attaquant malveillant essaye d'usurper une adresse MAC ou IP légitime afin d'envoyer des paquets vers le réseau, en utilisant une adresse de confiance. La réplique de l'adresse MAC/IP force les systèmes à croire que la source est digne de confiance. De la même manière que le scan de ports, nous avons expérimenté avec Nmap, et à travers la

commande spécifique (*nmap spoof-mac adresseMAC de la machine cible ou AdresseIp machine cible*), l'usurpation d'adresse IP et d'adresse MAC et nous avons constaté que Snort a détecté la menace et a enregistré le log associé.

Nous avons remarqué que Snort a détecté l'ensemble des attaques et a enregistré les fichiers correspondants dans le répertoire des logs. Cette procédure peut être étendue à d'autres types de menaces plus complexes et intelligentes.

6.6 Conclusion

L'implémentation de notre approche a été, pour nous, l'occasion d'acquérir une expérience pratique avec le SDN et a été très utile pour le développement de notre solution de sécurisation des échanges dans un réseau. Au cours de cette étude, nous nous sommes familiarisés avec les techniques d'installation et de la mise en service des solutions réseaux utilisant OpenFlow. Cela a été une étape très importante pendant nos travaux et nous aidera dans la suite de nos recherches qui se porteront à la fois sur OpenFlow, mais aussi sur l'amélioration de l'intégration des règles de sécurité de manière automatisée.

Conclusion et Perspectives

Conclusion

Le SDN est un paradigme qui fournit des innovations majeures en terme d'architecture et de gestion des réseaux, par la séparation entre le plan de contrôle et le plan de données dans les équipements réseaux, tels que des commutateurs et des routeurs. L'introduction d'un contrôleur externe permet de programmer les flux grâce aux protocoles OpenFlow et OpFlex.

En utilisant cette nouvelle approche de gestion des réseaux, nous avons proposé une solution de sécurisation des échanges dans un réseau en fonction des événements détectés. C'est dans ce contexte que nous avons mis en oeuvre un réseau SDN, même si certains aspects techniques de ce paradigme sont encore au stade de développement.

Les résultats ont montré qu'il est possible de mettre en oeuvre un réseau SDN en utilisant le protocole OpenFlow. Nous avons également observé à travers les expérimentations qu'il est possible de programmer un flux de bout en bout avec le SDN.

Après la mise en oeuvre d'un réseau SDN en environnement virtuel, nous avons ensuite ajouté des règles de sécurité pour bloquer un flux malveillant. De plus, nous avons introduit un IDS pour analyser en temps réel tous les flux du réseau et alerter le contrôleur en cas de tentative d'intrusion, afin de les supprimer de manière automatisée.

Par ailleurs, pour valider notre solution, nous avons réalisé plusieurs simulations en utilisant un contrôleur OpenDaylight et un IDS de type Snort qui échangent entre eux pour réaliser les actions d'analyse, de détection et de suppression de flux malveillants. Nous avons également mis en oeuvre différents scénarios d'attaque (DoS, scan de port et usurpation d'adresse IP) en utilisant l'outil Nmap afin de concrétiser notre solution.

Nous avons aussi proposé une architecture de sécurité avec du SDN étendu aux objets connectés. En divisant un réseau en cluster et en utilisant plusieurs contrôleurs, il est possible non seulement d'équilibrer la charge des contrôleurs et d'éviter que le contrôleur ne devienne un point critique, mais aussi de gérer la sécurité de manière collaborative grâce aux *East-WestBound* API.

Les résultats des expérimentations de mise en oeuvre du SDN, et les différentes possibilités de définition et d'implémentation des règles de sécurité, ont prouvé l'efficacité du SDN par rapport aux mécanismes de gestion classique des réseaux, d'où l'importance du SDN dans l'amélioration de l'administration et de la sécurité des réseaux IoT.

Perspectives

Aujourd'hui, il est extrêmement important de protéger tout système contre les intrusions et les accès non autorisés, car les usages ont évolué avec l'arrivée des objets connectés. La sécurité est devenue une composante essentielle et cruciale de toute solution réseau. Heureusement, des perspectives de recherche apparaissent grâce à la technologie SDN. Les fonctionnalités assurées par les contrôleurs SDN peuvent facilement évoluer en terme de capacité de traitement, en profitant des avantages offerts par le *cloud computing*.

L'ensemble des solutions présentées dans le cadre de cette thèse peuvent servir de socle pour la programmation et l'automatisation de la sécurisation des réseaux IoT.

Notre solution de sécurisation des échanges dans un réseau est aujourd'hui testée pour sécuriser un réseau classique. Mais elle peut être étendue à la sécurisation des objets connectés dans le futur.

À court terme, nous comptons mettre en oeuvre notre solution expérimentée en environnement virtuel, dans un environnement réel de production sur des équipements physiques. Nous pourrions mesurer l'impact de notre solution en situation réelle.

À moyen terme, des améliorations peuvent être apportées en ce qui concerne la détection des divers types de menaces spécifiques aux objets connectés. Ainsi, nous prévoyons d'améliorer le module de détection Snort en le comparant avec d'autres types d'IDS open source, tels que Bro¹⁰ ou Suricata¹¹. Nous pouvons également envisager d'expérimenter l'équilibrage de charge des équipements de contrôle pour qu'aucun contrôleur SDN ne soit hors service par manque de capacité de traitement, en cas d'augmentation de la taille du réseau. Ainsi, nous pouvons également évaluer les performances du réseau en terme de latence si la taille du réseau augmente.

Une question que nous nous posons pour le long terme serait de savoir quels outils utiliser pour simuler des équipements IoT non IP de type LoRA par exemple, afin d'expérimenter un réseau IoT hybride IP et Non IP.

10. [https://fr.wikipedia.org/wiki/Bro_\(IDS\)](https://fr.wikipedia.org/wiki/Bro_(IDS))/(Online : accessed on 04/03/2018)

11. <https://suricata-ids.org/>/(Online : accessed on 04/03/2018)

Liste des acronymes

AD-SAL	Application-Driven Service Abstraction Layer
API	Application Programming Interface
BGP	Border Gateway Protocol
CA	Certification Authority
DARPA	Defense Advanced Research Projects Agency
DoS	Denial of Service
DDoS	Distributed Denial of Service
EP	End Point
EPG	End Point Group
EPR	End Point Registry
ETSI	European Telecommunications Standards Institute
ForCES	Forwarding and Controle Element Separation
GBP	Group Based Policy
GSMA	GSM Association
GUI	Graphical User Interface
IDS	Intrusion Detection System
IoE	Internet of Everythings
IoT	Internet of Things
IPS	Intrusion Prevention System
IPSO	Internet Protocol for Smart Objects
IETF	Internet Engineering Task Force
ISO	International Organisation for Standardization
ITU-T	International Telecommunication Union - Telecom
JSON	Java Script Object Notation
JVM	Java Virtual Machine
KVM	Kernel-based Virtual Machine
LE	Low Energy

LFB	Logical Fonction Blocks
LLDP	Link Layer Discovery Protocol
LLDPDU	Link Layer Discovery Protocol Data Unit
M2M	Machine to Machine
MD-SAL	Model-Driven Service Abstraction Layer
NFC	Near Field Communication
NIDS	Network Intrusion Detection System
NIST	National Institute of Standards and Technology
ODL	OpenDayLight
OF	OpenFlow
OFDP	OpenFlow Discovery Protocol
ONF	Open Network Foundation
OSGi	Open Services Gateway initiative
OSI	Open System Interconnexion
OSPF	Open Shortest Path First
OS	Operating System
OVS	Open vSwitch
PE	Policy Element
PKI	Public Key Infrastructure
PR	Policy Repository
Qemu	Quick Emulator
QoS	Quality of Service
RAM	Random Access Memory
RFID	Radio Frequency IDentification
RPC	Remote Procedure Call
SAL	Service Abstraction Layer
SDN	Software-Defined Network
SSL	Secure Sockets Layer
TCAM	Ternary Content Addressable Memory
TLS	Transport Layer Security
TLV	Type-Length-Value
VM	Virtual Machine
VNC	Virtual Network Computing
VPN	Virtual Private Network
XML	Extensible Markup Language
XNC	Extensible Network Controler

Bibliographie

- [1] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, “Are we ready for sdn? implementation challenges for software-defined networks,” *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, July 2013.
- [2] A. Feghali, R. Kilany, and M. Chamoun, “Sdn security problems and solutions analysis,” in *2015 International Conference on Protocol Engineering (ICPE) and International Conference on New Technologies of Distributed Systems (NTDS)*, July 2015, pp. 1–5.
- [3] H. Lin and N. W. Bergmann, “Iot privacy and security challenges for smart home environments,” vol. 7, p. 44, 07 2016.
- [4] Q. Jing, A. V. Vasilakos, J. Wan, J. Lu, and D. Qiu, “Security of the internet of things : Perspectives and challenges,” *Wirel. Netw.*, vol. 20, pp. 2481–2501, Nov. 2014.
- [5] C. J. GONZALEZ, “Phd,” in *Management of a heterogeneous distributed architecture with the SDN*, 2017, p. 43.
- [6] I. Ahmad, S. Namal, M. Ylianttila, and A. Gurtov, “Security in software defined networks : A survey,” *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2317–2346, Fourthquarter 2015.
- [7] Y. Li, D. Li, W. Cui, and R. Zhang, “Research based on osi model,” in *2011 IEEE 3rd International Conference on Communication Software and Networks*, May 2011, pp. 554–557.
- [8] T. Benson, A. Akella, and D. Maltz, “Unraveling the complexity of network management,” in *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI’09, 2009, pp. 335–348.
- [9] D. Kreutz, F. M. V. Ramos, P. E. VerÃssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking : A comprehensive survey,” *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan 2015.

- [10] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turetetti, "A Survey of Software-Defined Networking : Past, Present, and Future of Programmable Networks," *IEEE Communications Surveys & Tutorials*, vol. 16, pp. 1617–1634, 2014.
- [11] T. Das, M. Caria, A. Jukan, and M. Hoffmann, "Insights on SDN migration trajectory," in *Communications (ICC), 2015 IEEE International Conference on*, 2015, pp. 5348–5353.
- [12] S. Wang, D. Li, and S. Xia, "The problems and solutions of network update in sdn : A survey," in *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2015, pp. 474–479.
- [13] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and openflow : From concept to implementation," *IEEE Communications Surveys Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.
- [14] C. Martinez, R. Ferro, and W. Ruiz, "Next generation networks under the sdn and open-flow protocol architecture," in *2015 Workshop on Engineering Applications - International Congress on Engineering (WEA)*, Oct 2015, pp. 1–7.
- [15] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow : enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 69–74, 2008.
- [16] M. Jarschel, T. Zinner, T. Hossfeld, P. Tran-Gia, and W. Kellerer, "Interfaces, attributes, and use cases : A compass for sdn," *IEEE Communications Magazine*, vol. 52, no. 6, pp. 210–217, June 2014.
- [17] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 136–141, February 2013.
- [18] M. Shin, K. Nam, and H. Kim, "Software-defined networking (sdn) : A reference architecture and open apis," in *2012 International Conference on ICT Convergence (ICTC)*, Oct 2012, pp. 360–361.
- [19] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," *ACM Trans. Internet Technol.*, pp. 115–150, 2002.
- [20] Y. E. Oktian, S. Lee, H. Lee, and J. Lam, "Secure your northbound sdn api," in *2015 Seventh International Conference on Ubiquitous and Future Networks*, July 2015, pp. 919–920.
- [21] J. A. Stankovic, "Research directions for the internet of things," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 3–9, Feb 2014.

- [22] V. Tyagi and A. Kumar, "Internet of things and social networks : A survey," in *2017 International Conference on Computing, Communication and Automation (ICCCA)*, May 2017, pp. 1268–1270.
- [23] P. Sethi and S. R. Sarangi, "Internet of things : Architectures, protocols, and applications," *Journal of Electrical and Computer Engineering*, pp. 243–259, Jan. 2017.
- [24] A. Luigi, I. Antonio, and M. Giacomo, "The internet of things : A survey," *Comput. Netw.*, vol. 54, pp. 2787–2805, Oct. 2010.
- [25] R. Khan, S. U. Khan, R. Zaheer, and S. Khan, "Future internet : The internet of things architecture, possible applications and key challenges," in *2012 10th International Conference on Frontiers of Information Technology*, Dec 2012, pp. 257–260.
- [26] Z. Yang, Y. Yue, Y. Yang, Y. Peng, X. Wang, and W. Liu, "Study and application on the architecture and key technologies for iot," in *2011 International Conference on Multimedia Technology*, July 2011, pp. 747–751.
- [27] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things : A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2347–2376, Fourthquarter 2015.
- [28] M. Wu, T.-J. Lu, F.-Y. Ling, J. Sun, and H.-Y. Du, "Research on the architecture of internet of things," in *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*, vol. 5, Aug 2010, pp. V5–484–V5–487.
- [29] M. Ojo, D. Adami, and S. Giordano, "A sdn-iot architecture with nfv implementation," in *2016 IEEE Globecom Workshops (GC Wkshps)*, Dec 2016, pp. 1–6.
- [30] L. Tan and N. Wang, "Future internet : The internet of things," in *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*, vol. 5, Aug 2010, pp. V5–376–V5–380.
- [31] K. Sood, S. Yu, and Y. Xiang, "Software-defined wireless networking opportunities and challenges for internet-of-things : A review," *IEEE Internet of Things Journal*, vol. 3, no. 4, pp. 453–463, Aug 2016.
- [32] P. Martinez-Juliai and F. S. Antonio, "Empowering the internet of things with software defined networking," in *FP7 European research project on the future Internet of Things*.
- [33] Z. Qin, G. Denker, C. Giannelli, P. Bellavista, and N. Venkatasubramanian, "A software defined networking architecture for the internet-of-things," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, May 2014, pp. 1–9.
- [34] D. Wu, D. I. Arkhipov, E. Asmare, Z. Qin, and J. A. McCann, "Ubiflow : Mobility management in urban-scale software defined iot," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, April 2015, pp. 208–216.

- [35] H. Huang, J. Zhu, and L. Zhang, "An sdn based management framework for iot devices," in *25th IET Irish Signals Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014)*, June 2014, pp. 175–179.
- [36] Y. Jararweh, M. Al-Ayyoub, A. Darabseh, E. Benkhelifa, M. Vouk, and A. Rindos, "Sdiot : A software defined based internet of things framework," vol. 1, pp. 453–461, 08 2015.
- [37] S. Nastic, S. Sehic, M. Vogler, H. Truong, and S. Dustdar, "Patricia – a novel programming model for iot applications on cloud platforms," in *2013 IEEE 6th International Conference on Service-Oriented Computing and Applications*, Dec 2013, pp. 53–60.
- [38] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *29th Annual IEEE International Conference on Local Computer Networks*, Nov 2004, pp. 455–462.
- [39] E. Baccelli, O. Hahm, M. Gunes, M. Wahlisch, and T. C. Schmidt, "Riot os : Towards an os for the internet of things," in *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2013, pp. 79–80.
- [40] P. Levis, S. Madden, J. Polastre, R. Szewczyk, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, "Tinyos : An operating system for sensor networks," in *in Ambient Intelligence*, 2004.
- [41] Q. Cao, T. Abdelzaher, J. Stankovic, and T. He, "The liteos operating system : Towards unix-like abstractions for wireless sensor networks," in *2008 International Conference on Information Processing in Sensor Networks (ipsn 2008)*, April 2008, pp. 233–244.
- [42] T. Luo, H. Tan, and T. Q. S. Quek, "Sensor openflow : Enabling software-defined wireless sensor networks," *IEEE Communications Letters*, vol. 16, no. 11, pp. 1896–1899, November 2012.
- [43] L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, "Sdn-wise : Design, prototyping and experimentation of a stateful sdn solution for wireless sensor networks," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, April 2015, pp. 513–521.
- [44] C. Li, Z. Qin, E. Novak, and Q. Li, "Securing sdn infrastructure of iot fog networks from mitm attacks," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1156–1164, Oct 2017.
- [45] K. Kalkan and S. Zeadally, "Securing internet of things with software defined networking," *IEEE Communications Magazine*, vol. 56, no. 9, pp. 186–192, Sep. 2018.
- [46] H. Ling-Fang, "The firewall technology study of network perimeter security," in *2012 IEEE Asia-Pacific Services Computing Conference*, Dec 2012, pp. 410–413.

- [47] F. Sabahi and A. Movaghar, "Intrusion detection : A survey," in *2008 Third International Conference on Systems and Networks Communications*, Oct 2008, pp. 23–26.
- [48] R. Koch, M. Golling, and G. D. Rodosek, "Behavior-based intrusion detection in encrypted environments," *IEEE Communications Magazine*, vol. 52, no. 7, pp. 124–131, July 2014.
- [49] K. Burbeck and S. Nadjm-Tehrani, "Adaptive real-time anomaly detection with improved index and ability to forget," in *25th IEEE International Conference on Distributed Computing Systems Workshops*, June 2005, pp. 195–202.
- [50] K. Ilgun, R. A. Kemmerer, and P. A. Porras, "State transition analysis : a rule-based intrusion detection approach," *IEEE Transactions on Software Engineering*, vol. 21, no. 3, pp. 181–199, March 1995.
- [51] Y. Wang, Y. Shen, and G. Zhang, "Research on intrusion detection model using ensemble learning methods," in *2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, Aug 2016, pp. 422–425.
- [52] H. Annuar, B. Shanmugam, A. Ahmad, N. B. Idris, S. H. AlBakri, and G. N. Samy, "Enhancement and implementation of network access control architecture for virtualization environments," in *2013 International Conference on Informatics and Creative Multimedia*, Sept 2013, pp. 314–320.
- [53] X. Song and T. You, "A network access control mechanism based on role and behavior," in *2010 3rd IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT)*, Oct 2010, pp. 95–99.
- [54] F. M. V. R. D. Kreutz and P. Verissimo, "Towards secure and dependable software-defined networks," in *in Proc. 2nd ACM Workshop Hot Topics in Software Defined Networks (HotSDN)*, 2013, pp. 55–60.
- [55] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, "A survey of active network research," *IEEE Communications Magazine*, vol. 35, no. 1, pp. 80–86, Jan 1997.
- [56] Z. Liu, R. H. Campbell, and M. D. Mickunas, "Active security support for active networks," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 33, no. 4, pp. 432–445, Nov 2003.
- [57] S. Murphy, E. Lewis, R. Puga, R. Watson, and R. Yee, "Strong security for active networks," in *2001 IEEE Open Architectures and Network Programming Proceedings. OPENARCH 2001 (Cat. No.01EX484)*, April 2001, pp. 63–70.

- [58] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, and S. Shenker, "Sane : A protection architecture for enterprise networks," in *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, 2006.
- [59] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane : Taking control of the enterprise," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 1–12, aug 2007.
- [60] S. Khan, A. Gani, A. W. A. Wahab, M. Guizani, and M. K. Khan, "Topology discovery in software defined networks : Threats, taxonomy, and state-of-the-art," *IEEE Communications Surveys Tutorials*, vol. 19, no. 1, pp. 303–324, 2017.
- [61] Z. Shu, J. Wan, D. Li, J. Lin, A. V. Vasilakos, and M. Imran, "Security in software-defined networking : Threats and countermeasures," vol. 21. in journal *Mobile Networks and Applications*.
- [62] S. Scott-Hayward, G. O'Callaghan, and S. Sezer, "Sdn security : A survey," in *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*, Nov 2013, pp. 1–7.
- [63] L. J. C. K. Benton and C. Small, "Openflow vulnerability assessment," vol. 21. in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Software Defined Network*.
- [64] P. P. S. Shin, V. Yegneswaran and G. Gu, "Avant-guard : Scalable and vigilant switch flow management in software-defined networks." in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, Berlin, Germany, 2013, pp. 413–424.
- [65] A. Kamisinski and C. Fung, "Flowmon : Detecting malicious switches in software-defined networks." in *Proc. Workshop Autom. Decis. Making Active Cyber Defense*, Denver, CO, USA, 2015, pp. 39–45.
- [66] Y. Zhang, N. Beheshti, and M. Tatipamula, "On resilience of split-architecture networks," in *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011*, Dec 2011, pp. 1–6.
- [67] C. Dixon, D. Olshefski, V. Jain, C. DeCusatis, W. Felter, J. Carter, M. Banikazemi, V. Mann, J. M. Tracey, and R. Recio, "Software defined networking to support the software defined environment," *IBM Journal of Research and Development*, vol. 58, no. 2/3, pp. 3 :1–3 :14, March 2014.
- [68] N. E. Moussaid, A. Toumanari, and M. E. Azhari, "Security analysis as software-defined security for sdn environment," in *2017 Fourth International Conference on Software Defined Systems (SDS)*, May 2017, pp. 87–92.
- [69] G. Yao, J. Bi, and L. Guo, "On the cascading failures of multi-controllers in software defined networks," in *2013 21st IEEE International Conference on Network Protocols (ICNP)*, Oct 2013, pp. 1–2.

- [70] S. Shin and G. Gu, "Attacking software-defined networks : a first feasibility study," in *in Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 165–166.
- [71] C. Sungryung, C. Sungmoon, L. Wooyeob, J. Inwhae, P. Jeman, L. Soohyung, and K. Wontae, "An software defined networking architecture design based on topic learning-enabled data distribution service middleware," in *Adv. Sci. Lett*, vol. 21, 2015, pp. 461–464.
- [72] M. C. P. B. G. Ahmed Khurshid, Wenxuan Zhou, "Veriflow : Verifying network-wide invariants in real time," in *ACM SIGCOMM Computer Communication Review*, vol. 42, Oct 2012, pp. 467–472.
- [73] V. Y. M. F. M. T. P. Porras, S. Shin and G. Gu, "A security enforcement kernel for openflow networks," in *in Proceedings of the First Workshop on Hot Topics in Software Defined Networks, ser. HotSDN 12. ACM*, 2012, pp. 121–126.
- [74] G. Yao, J. Bi, and P. Xiao, "Source address validation solution with openflow/nox architecture," in *2011 19th IEEE International Conference on Network Protocols*, Oct 2011, pp. 7–12.
- [75] E. Al-Shaer and S. Al-Haj, "Flowchecker : Configuration analysis and verification of federated openflow infrastructures," in *in Proceedings of the 3rd ACM Workshop on Assurable and Usable Security Configuration, ser. SafeConfig 10. ACM*, Dec 2010, pp. 37–44.
- [76] T. Sasaki, C. Pappas, T. Lee, T. Hoefler, and A. Perrig, "Sdnsec : Forwarding accountability for the sdn data plane," in *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, Aug 2016, pp. 1–10.
- [77] H. Wang, L. Xu, and G. Gu, "Floodguard : A dos attack prevention extension in software-defined networks," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2015, pp. 239–250.
- [78] M. F. K. S. V. Y. P. Porras, S. Cheung, "Securing the software-defined network control layer," in *Proceedings of the 2015*, 2015.
- [79] K. Phemius, M. Bouet, and J. Leguay, "Disco : Distributed multi-domain sdn controllers," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, May 2014, pp. 1–4.
- [80] A. Tootoonchian and Y. Ganjali, "Hyperflow : A distributed control plane for openflow," in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, 2010, pp. 3–3.

- [81] A. Voellmy and J. Wang, “Scalable software defined network controllers,” in *in Proc. ACM SIGCOMM Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2012, pp. 289–290.
- [82] M. S. Hoyos Ll, G. Isaza, J. Velez, and L. Castillo, “Distributed denial of service (ddos) attacks detection using machine learning prototype,” vol. 474, pp. 33–41, 01 2016.
- [83] N. Foster, A. Guha, M. Reitblatt, A. Story, M. J. Freedman, N. P. Katta, C. Monsanto, J. Reich, J. Rexford, C. Schlesinger, D. Walker, and R. Harrison, “Languages for software-defined networks,” *IEEE Communications Magazine*, vol. 51, no. 2, pp. 128–134, February 2013.
- [84] J. R. M. Canini, D. Kostic and D. Venzano, “Automating the testing of openflow applications,” in *in Proceedings of the 1st International Workshop on Rigorous Protocol Engineering (WRiPE)*, 2011, pp. 1–6.
- [85] H. C. S. C. W. Y. Wen X, Chen Y, “Towards a secure controller platform for openflow applications,” in *In Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, 2013, pp. 171–172.
- [86] S. Son, S. Shin, V. Yegneswaran, P. Porras, and G. Gu, “Model checking invariant security properties in openflow,” in *2013 IEEE International Conference on Communications (ICC)*, June 2013, pp. 1974–1979.
- [87] V. Y. M. F. G. G. S. Shin, P. Porras and M. Tyson, “Fresco : Modular composable security services for software-defined networks,” in *in Proceedings of Network and Distributed Security Symposium*, 2013.
- [88] C. Gonzalez, S. M. Charfadine, O. Flauzac, and F. Nolot, “Sdn-based security framework for the iot in distributed grid,” in *2016 International Multidisciplinary Conference on Computer and Energy Science (SpliTech)*, July 2016, pp. 1–5.
- [89] N. Zope, S. Pawar, and Z. Saquib, “Firewall and load balancing as an application of sdn,” in *2016 Conference on Advances in Signal Processing (CASP)*, June 2016, pp. 354–359.
- [90] T. V. Tran and H. Ahn, “A network topology-aware selectively distributed firewall control in sdn,” in *2015 International Conference on Information and Communication Technology Convergence (ICTC)*, Oct 2015, pp. 89–94.
- [91] J. G. V. Pena and W. E. Yu, “Development of a distributed firewall using software defined networking technology,” in *2014 4th IEEE International Conference on Information Science and Technology*, April 2014, pp. 449–452.
- [92] T. Javid, T. Riaz, and A. Rasheed, “A layer2 firewall for software defined network,” in *2014 Conference on Information Assurance and Cyber Security (CIACS)*, June 2014, pp. 39–42.

- [93] K. Kaur, K. Kumar, J. Singh, and N. S. Ghumman, "Programmable firewall using software defined networking," in *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, March 2015, pp. 2125–2129.
- [94] M. Suh, S. H. Park, B. Lee, and S. Yang, "Building firewall over the software-defined network controller," in *16th International Conference on Advanced Communication Technology*, Feb 2014, pp. 744–748.
- [95] W. M. Othman, H. Chen, A. Al-Moalimi, and A. N. Hadi, "Implementation and performance analysis of sdn firewall on pox controller," in *2017 IEEE 9th International Conference on Communication Software and Networks (ICCSN)*, 2017, pp. 1461–1466.
- [96] O. Flauzac, C. Gonzalez, and F. Nolot, "Original secure architecture for IoT based on SDN," in *2015 International Conference on Protocol Engineering (ICPE) and International Conference on New Technologies of Distributed Systems (NTDS)*, Jul. 2015, pp. 1–6.
- [97] C. Jeong, T. Ha, J. Narantuya, H. Lim, and J. Kim, "Scalable network intrusion detection on virtual sdn environment," in *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*, Oct 2014, pp. 264–265.
- [98] M. A. Sayeed, M. A. Sayeed, and S. Saxena, "Intrusion detection system based on software defined network firewall," in *2015 1st International Conference on Next Generation Computing Technologies (NGCT)*, Sept 2015, pp. 379–382.
- [99] P. J. Chen and Y. W. Chen, "Implementation of sdn based network intrusion detection and prevention system," in *2015 International Carnahan Conference on Security Technology (ICCST)*, Sept 2015, pp. 141–146.
- [100] A. Abubakar and B. Pranggono, "Machine learning based intrusion detection system for software defined networks," in *2017 Seventh International Conference on Emerging Security Technologies (EST)*, Sept 2017, pp. 138–143.
- [101] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion detection in software defined networking," in *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*, Oct 2016, pp. 258–263.
- [102] O. Flauzac, F. Nolot, C. Rabat, and L. A. Steffemel, "Grid of security : A new approach of the network security," in *2009 Third International Conference on Network and System Security*, Oct 2009, pp. 67–72.
- [103] S. M. Charfadine, O. Flauzac, F. Nolot, C. Rabat, and C. Gonzalez, "Secure exchanges activity in function of event detection with the sdn," in *e-Infrastructure and e-Services for*

- Developing Countries. 10th EAI International Conference, AFRICOMM 2018.* Springer International Publishing, March 2019, pp. 315–324.
- [104] J. U. K. J. Y. Jo and K. M. Lee, “System and method for preventing network intrusion,” in *Patent KR101553264*, 2015.
- [105] Z. L. H. Han, L. Yan and J. Zhang, “Network security defense system based on software-defined network and working method of network security defense system,” in *Patent CN104539625*, 2015.
- [106] C. Luo, “Network security traffic platform based on software definition,” in *Patent CN104753951*, 2015.
- [107] P. Bull, R. Austin, E. Popov, M. Sharma, and R. Watson, “Flow based security for iot devices using an sdn gateway,” in *2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*, Aug 2016, pp. 157–163.
- [108] C. Gonzalez, O. Flauzac, F. Nolot, and A. Jara, “A novel distributed sdn-secured architecture for the iot,” in *International Conference on Distributed Computing in Sensor Systems (DCOSS) Washington, DC*, May 2016, pp. 244–249.
- [109] C. Vandana, “Security improvement in iot based on software defined networking,” in *International Journal of Science, Engineering and Technology Research (IJSETR)*, vol. 5, 2016.
- [110] M. Roesch, “Snort lightweight intrusion detection for networks.” In Proc. of the 13th USENIX Conf. on Systems Administration LISA 99, Seattle, Washington, USA, 1999, pp. 229–238.
- [111] A. Alhomoud, R. Munir, J. P. Disso, I. Awan, and A. Al-Dhelaan, “Performance evaluation study of intrusion detection systems,” *Procedia Computer Science*, vol. 5, pp. 173 – 180, 2011.
- [112] F. Pakzad, M. Portmann, W. L. Tan, and J. Indulska, “Efficient topology discovery in software defined networks,” in *Signal Processing and Communication Systems (ICSPCS), 2014 8th International Conference on*, 2014, pp. 1–8.
- [113] L. Ochoa Aday, C. Cervell Pastor, and A. Fernandez Fernandez, “Current Trends of Topology Discovery in OpenFlow-based Software Defined Networks,” 2015.

Gestion dynamique et évolutive de règles de sécurité pour l'Internet des Objets

Avec l'évolution exponentielle de l'Internet des Objets (IoT), assurer la sécurité des réseaux est devenue un grand défi pour les administrateurs réseaux. La sécurité des réseaux est basée sur de multiples équipements indépendants tels que Firewall, IDS/IPS, NAC dont le rôle principal est de contrôler les informations échangées entre le réseau de l'entreprise et l'extérieur. Or, l'administration de ces équipements peut s'avérer très complexe et fastidieuse si elle est réalisée manuellement, équipement après équipement. L'introduction du concept de Software Defined Networking (SDN) depuis ces dernières années, et du protocole OpenFlow, offre beaucoup d'opportunités pour l'amélioration de la sécurité des réseaux en proposant une administration centralisée et programmable.

Dans le cadre de cette thèse, nous avons proposé une nouvelle approche de sécurisation des échanges dans un réseau en fonction des événements détectés et de manière automatisée. Cette solution basée sur l'approche SDN couplé avec un système de détection d'intrusion permet d'analyser, de détecter et de supprimer des menaces de sécurité dans un réseau de manière automatisée. En implémentant cette solution, nous contribuons à faire évoluer la manière de sécuriser les échanges dans un réseau, avec du SDN couplé avec un IDS à travers la mise en place d'une architecture réelle de cas d'usage. Ainsi, la gestion de la sécurité du réseau devient simplifiée, dynamique et évolutive.

Mots-clés : IoT, SDN, Sécurité, OpenFlow, Firewall, IPS/IDS, NAC.

Dynamic and scalable management of security rules for the Internet of Things


With the exponential evolution of the Internet of Things (IoT), ensure the network security has become a big challenge for network administrators. Traditionally, the network security is based on multiple independent devices such as firewall, IDS/IPS, NAC where the main role is to monitor the information exchanged between the inside and the outside perimeters of the enterprises networks. However, the administration of these network devices can be complex and tedious with an independent manual configuration. Recently, with the introduction of the Software Defined Networking concept (SDN) and the OpenFlow protocol offers many opportunities by providing a centralized and programmable network administration.

As part of this research work, we proposed a new approach to secure the network traffic flows exchanges based on a method of events detection, in an automated manner. This solution is based on the SDN approach coupled to an intrusion detection system which allows analyze, detect and remove security threats. With the implementation, we contribute to change the paradigm of secure the network traffic flows exchanges using the SDN principle, coupled with an IDS in a real use case architecture. In this way, the management of network security becomes simplified, dynamic and scalable.

Keywords : IoT, SDN, Security, OpenFlow, Firewall, IPS/IDS, NAC.

Discipline : INFORMATIQUE

Spécialité : Réseaux et Télécommunications



Université de Reims Champagne-Ardenne

CRESTIC - EA 3804

BP 1039 - 51687 Reims CEDEX 2