



Mémoire en vue de l'obtention du diplôme
D'Ingénieur d'État.

- Filière -
Télécommunications

- Spécialité -
Systèmes de Télécommunication et Réseaux

– Thème –

Conception et mise en œuvre d'une architecture SDN pour la gestion des services réseau

- Réalisé par -

**BOUDJAMMALA Abdessalam
SARI Mohammed Islam**

Membres du jury :

Prénom et NOM	Etablissement	Qualité
Mme. Akila SID	ENSTA	Président
M. Abdeldjallil MOUZAOU	CERIST	Promoteur
Mme. Seloua CHOUAF	ENSTA	Co-Promoteur
M. Tarek CHERIFI	ENSTA	Examineur
Mme. Imane CHIALI	ENSTA	Examineur

Dédicace

Je dédie ce travail à toutes les personnes qui ont marqué mon parcours universitaire, mais tout particulièrement à :

*À **ma petite famille, Mama**, ta tendresse est ce doux rayon de soleil qui réchauffe chacun de mes jours, et **Papa**, ta confiance inébranlable est la fondation solide sur laquelle je bâtis mes rêves. Votre amour inconditionnel et vos prières discrètes ont été ma force. Chaque mot d'encouragement, chaque sacrifice m'ont porté jusqu'ici. Vous m'avez donné des ailes, sans vous, rien de tout cela n'aurait été possible. **Mon frère** et **Ma sœur**, votre affection, vos encouragements et votre présence rassurante à chaque étape. Je vous aime plus que les mots ne sauraient l'exprimer.*

*À **Ma Grand-Mère** et à **Ma Tante** pour leur présence constante et leurs précieux conseils. Leur soutien a été une source de force et d'inspiration dans la réalisation de ce travail.*

*À **mes frères Rayane, Mouad et Khireddine** : je repense à notre toute première rencontre, aux heures passées ensemble et à toutes les aventures que nous avons partagées. Il n'a jamais fallu de temps pour que vous deveniez mes véritables frères d'âme. Je vous aime profondément .*

*À **mon binôme** et pote **Abdessalam Boudjammala**, merci d'avoir été à fond dans ce projet à mes côtés. Ta bonne humeur, tes idées folles et ta persévérance ont rendu chaque étape plus cool. Vraiment, sans toi, ça n'aurait pas été pareil !*

*À **Aymen.A, Adlene.Z, Sami.M, Oussama.A**, et à toute personne pour leurs mots justes, leur motivation, leur aide et les moments de partage qui ont égayé ce chemin.*

*Je dédie ce travail à **Madame Chouaf Seloua**, mon enseignante et encadrante, dont la disponibilité et la rigueur ont été essentielles. Ses conseils éclairés et sa bienveillance ont guidé notre curiosité et renforcé notre confiance. Merci pour votre soutien constant et votre engagement sans faille ; sans vous, ce travail n'aurait pas pu atteindre son niveau d'excellence.*

*Et surtout, à mon inséparable compagnon et **mon pote d'enfance Younes Iglouli**, je te souhaite de tout cœur une belle réussite dans ta carrière et dans ta vie. À **Anis Boukhelifa** ; que Dieu t'accueille dans Son vaste paradis. Et à tous mes amis, merci de partager rires, espoirs et souvenirs : ce travail porte aussi votre empreinte, et sans vous il n'aurait pas la même chaleur.*

SARI Mohammed Islam

Dédicace

Ce travail représente bien plus qu'un simple mémoire de fin d'études. Il est le fruit d'un parcours soutenu par l'écoute, le soutien et la générosité de nombreuses personnes auxquelles je tiens à adresser toute ma gratitude :

À mes parents,

Pour leur amour inconditionnel, leur patience infinie et les sacrifices qu'ils n'ont jamais cessé de faire pour me voir réussir. Leur confiance a toujours été ma plus grande source de force.

À mes frères,

Pour leur présence discrète mais toujours rassurante, et leur soutien constant tout au long de mon chemin.

À Madame Chouaf,

Notre encadrante, pour son accompagnement rigoureux, sa bienveillance et la qualité de ses conseils qui ont profondément enrichi ce travail.

À mon binôme SARI Med Islam,

Pour son implication sérieuse, son esprit d'équipe et la complémentarité qui a marqué notre collaboration du début à la fin.

À mes amis proches, Akhzeroune Aymen et Boukaroui Rayane,

Pour leur soutien moral, leur amitié sincère et les moments de répit partagés durant cette aventure.

Enfin, à Monsieur Boukais Amine,

Ingénieur réseau chez Icosnet, dont l'expertise, la disponibilité et les conseils techniques m'ont été d'une aide précieuse dans la concrétisation de ce projet.

À vous tous, je dédie ce travail avec gratitude et respect.

BOUDJAMMALA Abdessalam

Remerciements

Je tiens tout d'abord à remercier Dieu le Tout-Puissant, le Miséricordieux, de nous avoir permis de mener à bien ce travail.

Je tiens aussi à exprimer mes sincères remerciements et ma profonde gratitude envers mes encadrants. Mme CHOUAF pour l'orientation, la confiance et la patience qui ont constitué un apport considérable, et M. MOUZAOUÏ du CERIST pour son encadrement et pour ses conseils , mais surtout pour son expertise.

J'adresse mes remerciements aux membres du jury madame SID Akila, madame CHIALI Imane et monsieur CHERIFI Tarek, merci d'avoir consacré une partie de votre temps à lire, examiner et évaluer ce travail.

Enfin, je veux exprimer par ces quelques lignes de remerciements ma gratitude envers nos parents pour leur présence, leurs soutiens et leurs conseils, afin de trouver le courage d'accomplir ce projet.

ملخص:

مع التوسع السريع في خدمات الحوسبة السحابية وتطبيقات إنترنت الأشياء، برزت الحاجة الملحة لشبكة أكثر مرونة وقابلة للبرمجة، مما جعل SDN حلاً مثالياً لهذه المتطلبات. يتناول هذا البحث تصميم وتنفيذ بنية SDN لمركز بحث. انطلقنا من إنشاء بيئة محاكاة باستخدام Mininet وMiniEdit، ثم اعتمدنا OpenDaylight للتحكم المركزي واستخدمنا Wireshark لمراقبة حركة البيانات و OpenFlow Manager للتحكم في أداء الشبكة. أظهرت الاختبارات انخفاضاً في زمن الاستجابة، وتحسناً في سهولة الإدارة، وتعزيزاً للأمان.

الكلمات المفتاحية: SDN، المحاكاة، Mininet، MiniEdit، OpenDaylight، OpenFlow Manager، Wireshark

Résumé :

Face à l'essor des services cloud et de l'Internet des objets, il est devenu indispensable de disposer de réseaux plus flexibles et programmables : le SDN répond à cette révolution. Ce mémoire décrit la mise en place d'une architecture SDN pour un centre de recherche. Nous avons émulé le réseau avec Mininet/MiniEdit, piloté les flux via OpenDaylight, l'OpenFlow Manager pour contrôler le réseau et Wireshark pour surveiller le trafic. Les tests ont mis en évidence une meilleure réactivité, une administration simplifiée et une sécurité renforcée.

Mots-clés : SDN, émulation, Mininet, MiniEdit, OpenDaylight, OpenFlow Manager, Wireshark

Abstract :

Amid the rapid growth of cloud services and IoT applications, networks must become more flexible and programmable, and SDN stands at the forefront of this transformation. This thesis presents the deployment of an SDN architecture for a research center. We emulated the network using Mininet/MiniEdit, controlled flows via OpenDaylight, supervised the network with the OpenFlow Manager, and monitored traffic using Wireshark. Test results demonstrate reduced latency, simplified management, and enhanced security.

Keywords : SDN, emulation, Mininet, MiniEdit, OpenDaylight, OpenFlow Manager, Wireshark

Table des matières

Dédicace	i
Remerciements	iii
Résumé	iv
Liste des acronymes	vii
Liste des figures	x
Liste des tableaux	xi
Introduction Générale	1
1 État de l’art sur la technologie SDN	2
1.1 Introduction	2
1.2 Comprendre le SDN : Définition et Enjeux	2
1.3 Historique de développement des SDN	4
1.4 Présentation de la technologie SDN	5
1.4.1 La couche de contrôle	6
1.4.2 La couche d’application	6
1.4.3 La couche de données	6
1.4.4 Interface Nord (Northbound API)	7
1.4.5 Interface Sud (Southbound API)	7
1.5 Protocoles de communication dans le SDN	7
1.6 Réseau OpenFlow	8
1.6.1 Le Contrôleur SDN	8
1.6.2 Le protocole OpenFlow	11
1.6.3 Les commutateurs OpenFlow	13
1.7 Conclusion	16
2 Analyse de l’existant et optimisation	18
2.1 Introduction	18
2.2 Présentation de l’architecture actuelle	18
2.3 Analyse des failles et des limites	19
2.4 Étude de terrain pour dimensionnement du système réseau	20
2.5 Proposition d’une alternative SDN	21
2.5.1 Modèle hiérarchique et sécurisation périmétrique	21
2.5.2 L’architecture réseau de chaque bloc	25
2.5.3 Architecture globale du système	28

2.6	Conclusion	30
3	Validation de l'architecture SDN	31
3.1	Introduction	31
3.2	Indisponibilité matérielle	31
3.3	Le passage vers l'émulation	32
3.4	Préparation de l'environnement d'émulation	32
3.4.1	Installation et configuration de Mininet	33
3.4.2	Choix et installation du contrôleur OpenDaylight	33
3.4.3	Installation d'OpenFlow Manager	35
3.4.4	Installation et configuration de WireShark	37
3.5	Réalisation de l'architecture à l'aide de MiniEdit	38
3.5.1	Présentation de l'outil MiniEdit	38
3.5.2	Topologie déployée	39
3.6	Test et validation	43
3.6.1	Test de connectivité	43
3.6.2	Test de sécurité : filtrage des flux interdits	45
3.7	Conclusion	52
	Conclusion Générale	53
	Bibliographie	xii

Liste des acronymes

API Application Programming Interface

BGP-LS Border Gateway Protocol Link-State

DMZ Zone Démilitarisée

D_OVS Distribution open virtual switch

GUI Graphical User Interface

ICMP Internet Control Message Protocol

IoT Internet of Things

IP Internet Protocol

LAN Local Area Network

MD-SAL Model-Driven Service Abstraction Layer

MAC Media Access Control

ODL Open Daylight

OFM Openflow Manager

ONF Open Network Foundation

ONOS Open Network Operating System

OS Operating System

OSI Open Systems Interconnection

OvS Open virtual Switch

PCEP Path Computation Element Protocol

QoS Quality of Service

RYU C'est le flux en japonais

SDN Software Defined Networking

SNMP Simple Network Management Protocol

SONiC Software for Open Networking in the Cloud

STP Spanning Tree Protocol

TLS Transport Layer Security

TCP Transmission Control Protocol

VIP Virtual IP

VLAN Virtual Local Area Network

VM Virtual Machine

VRRP Virtual Router Redundancy Protocol

Liste des figures

FIG. 1.1	Le réseau traditionnel contre le SDN	2
FIG. 1.2	L'évolution du Software-Defined Networking (SDN)	5
FIG. 1.3	Architecture SDN	6
FIG. 1.4	vSwitches (les composants de la couche de données)	7
FIG. 1.5	Comparaison de la structure des contrôleurs SDN centralisé et distribué.	9
FIG. 1.6	Logos des différents contrôleurs SDN	11
FIG. 1.7	Les messages du protocole OpenFlow [15]	11
FIG. 1.8	L'architecture du OpenFlow switch	13
FIG. 1.9	Structure détaillée d'une entrée de flux dans un commutateur OpenFlow	14
FIG. 1.10	Déroulement du traitement d'un paquet dans une table des flux	15
FIG. 1.11	Comparaison des modes proactif et réactif	16
FIG. 2.1	L'architecture actuelle du réseau	18
FIG. 2.2	Comparaison des architectures en cascade et hiérarchique	22
FIG. 2.3	L'empilage de switches [27]	23
FIG. 2.4	La DMZ avec un seul pare-feu [28]	24
FIG. 2.5	La DMZ avec deux pare-feu [28]	24
FIG. 2.6	L'architecture proposée pour le bloc A	25
FIG. 2.7	L'architecture proposée pour le bloc B	26
FIG. 2.8	L'architecture proposée pour les blocs C et D	27
FIG. 2.9	L'architecture proposée pour le bloc E	27
FIG. 2.10	L'architecture globale de la solution SDN	28
FIG. 3.1	Limitation du matériel disponible	31
FIG. 3.2	Description de l'architecture des composants OFM [35]	35
FIG. 3.3	L'interface principale de Wireshark	37
FIG. 3.4	L'architecture proposée sur MiniEdit	39
FIG. 3.5	Lancement de l'ODL	42
FIG. 3.6	L'interface graphique vierge d'Open Daylight (ODL)	42
FIG. 3.7	Connexion entre le plan de contrôle et le plan de données	43
FIG. 3.8	Négociation entre le contrôleur et les OvS	43
FIG. 3.9	Vue d'architecture de la part de ODL	44
FIG. 3.10	La validation du test de connectivité	44
FIG. 3.11	Le lancement d'OpenFlow Manager	45
FIG. 3.12	La détection de l'architecture par l'OpenFlow Manager	45
FIG. 3.13	La visualisation des flux existants	46
FIG. 3.14	La création manuelle d'une entrée de flux dans la section (Match)	46
FIG. 3.15	La création manuelle d'une entrée de flux dans la section (Action)	47

FIG. 3.16	Le fichier JSON créé par l'OpenFlow Manager	47
FIG. 3.17	La confirmation de la création de l'entrée de flux	49
FIG. 3.18	Le bon fonctionnement de l'API REST	49
FIG. 3.19	L'ajout de notre entrée de flux dans le S3 par Openflow	49
FIG. 3.20	L'installation du flux de blocage dans le S3 (switch du Bloc A) . . .	50
FIG. 3.21	L'entrée de flux envoyé par OFM vers S3	50
FIG. 3.22	Première validation du test (filtrage des flux interdits)	51
FIG. 3.23	Deuxième validation du test (filtrage des flux interdits)	51

Liste des tableaux

TAB. 1.1	Comparaison des réseaux SDN et traditionnels [2]	3
TAB. 1.2	Comparaison des contrôleurs SDN centralisé et distribué.	9
TAB. 1.3	Les différentes versions d'OpenFlow de 1.0 à 1.5 [15]	12
TAB. 2.1	Présentation des blocs et de leurs interconnexions	21
TAB. 2.2	Composantes de l'architecture globale	29
TAB. 3.1	Configuration IP de chaque nœud	39

Introduction générale

Les réseaux informatiques occupent aujourd’hui une place centrale dans les systèmes d’information modernes. Qu’il s’agisse d’organisations privées, d’institutions publiques ou d’infrastructures critiques, la connectivité, la sécurité et la disponibilité sont devenues des exigences fondamentales. Cependant, les architectures réseau traditionnelles, fondées sur une gestion statique des équipements, peinent à répondre aux besoins actuels en matière de flexibilité, de reconfiguration dynamique et de contrôle centralisé.

C’est dans ce contexte que le *Software-Defined Networking* (SDN) s’est imposé comme une alternative innovante. En introduisant une séparation claire entre le plan de contrôle — chargé de la prise de décision — et le plan de données — responsable du traitement des flux — le SDN permet une gestion centralisée, programmable et évolutive de l’ensemble du réseau. Cette nouvelle approche offre une vision globale de la topologie, facilite l’application des politiques réseau et améliore considérablement la capacité d’adaptation face aux changements de trafic ou aux pannes.

Le présent mémoire s’inscrit dans cette dynamique et vise à concevoir et valider une architecture réseau entièrement basée sur le paradigme SDN. L’objectif principal est de démontrer, à travers une démarche théorique et pratique, l’efficacité et la pertinence du SDN.

Pour atteindre cet objectif, ce travail s’articule autour de trois chapitres complémentaires :

- Le **chapitre 1** présente une étude théorique complète des principes du SDN, en abordant son architecture, ses composants clés et ses protocoles fondamentaux.
- Le **chapitre 2** propose une analyse critique d’une architecture réseau classique, identifie ses limites, et présente une nouvelle architecture fondée sur le SDN.
- Le **chapitre 3** est consacré à la mise en œuvre de l’architecture proposée par émulation, à l’intégration du contrôleur SDN et aux tests de validation pour évaluer sa performance, sa résilience et sa sécurité.

En conclusion, ce mémoire démontre la faisabilité et la robustesse de l’approche SDN à travers une expérimentation concrète, tout en ouvrant la voie à plusieurs perspectives d’amélioration, notamment en matière de tolérance aux pannes, de déploiement multi-contrôleurs et de passage vers une infrastructure physique réelle.

Chapitre 1

État de l'art sur la technologie SDN

1.1 Introduction

Dans ce premier chapitre, nous dressons un panorama complet de l'état de l'art du Software-Defined Networking (SDN). Nous commencerons par exposer les limites des réseaux IP traditionnels face à la montée en charge et aux besoins d'agilité et d'automatisation des infrastructures modernes. Nous retracerons ensuite l'historique du SDN, depuis ses débuts avec le protocole OpenFlow à l'université de Stanford jusqu'à sa standardisation et son adoption par la communauté industrielle. Puis, nous détaillerons les concepts fondamentaux du modèle SDN — les plans applicatif, de contrôle et de données ainsi que les interfaces northbound et southbound qui les interconnectent. Cette introduction prépare le terrain pour l'étude des protocoles de communication, en particulier OpenFlow, et pour la présentation des principaux composants d'une architecture SDN, à savoir les contrôleurs et les commutateurs OpenFlow.

1.2 Comprendre le SDN : Définition et Enjeux

L'Internet a transformé notre monde en une société numérique, connectée et accessible partout. Cependant, malgré leur adoption généralisée, les réseaux IP traditionnels sont très complexes et difficiles à contrôler. Il est à la fois difficile de configurer ces réseaux selon des politiques prédéfinies et de les reconfigurer pour répondre aux pannes, à la charge et aux changements. De plus, les réseaux traditionnels sont intégrés verticalement, ce qui signifie que les plans de contrôle et de données sont regroupés, rendant leur gestion encore plus complexe [1].

Le Réseau Défini par Logiciel (**SDN**, pour Software-Defined Networking) propose une approche novatrice qui sépare le plan de contrôle du plan de données. Cette séparation permet de centraliser la gestion du réseau et de le rendre programmable. Le SDN simplifie ainsi la création de nouvelles fonctions et facilite le contrôle et l'évolution du réseau.

Pour comprendre l'importance de cette évolution, il est essentiel de comparer l'architecture SDN à une architecture réseau traditionnelle. L'illustration suivante présente les différences majeures entre les deux architectures, mettant en lumière les avantages du SDN.

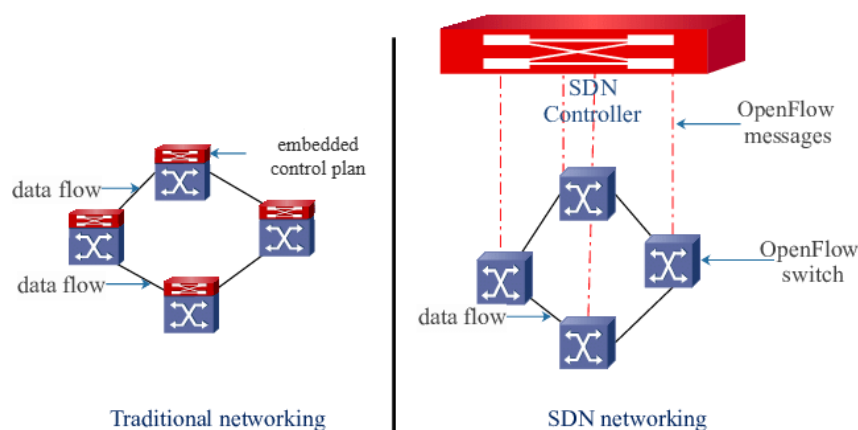


FIG. 1.1 : Le réseau traditionnel contre le SDN

Comme le montre la figure 1.1, l'architecture SDN dissocie totalement le plan de contrôle du plan de données, ce qui permet une gestion centralisée et plus flexible des réseaux.

Cela représente une avancée par rapport aux réseaux traditionnels où la gestion des équipements est souvent locale et complexe.

Dans cette section, nous comparons une architecture à base de SDN et une simple architecture traditionnelle pour souligner l'utilité du SDN.

TAB. 1.1 : Comparaison des réseaux SDN et traditionnels [2]

Caractéristiques	SDN	Réseau Traditionnel
Contrôle centralisé du réseau	✓	✗
Programmabilité et Automatisation	✓	✗
Flexibilité du réseau	✓	✗
Réseau de contrôle complexe	✗	✓
Configuration sujette aux erreurs	✗	✓
Innovation et Réduction des coût	✓	✗
Facilité d'utilisation et d'intégration	✓	✗
Réaction rapide au panne	✓	✗
Évitement des boucles	✓	✗
Contrôleur unique sensible aux pannes	✓	✗
Contrôleur unique sensible aux attaques	✓	✗

Comme illustré par le tableau précédent, il est important de retenir les aspects suivants :

- **Contrôle centralisé du réseau :** Dans le SDN, le contrôleur central gère l'ensemble des équipements du réseau, offrant une vision unifiée pour une prise de décision globale. À l'inverse, dans les réseaux traditionnels, chaque nœud réseau prend des décisions localement, ce qui rend la coordination plus difficile.
- **Programmabilité et automatisation :** Dans le SDN, programmation centralisée via des interfaces ouvertes (APIs), facilitant l'automatisation et l'orchestration. Par contre, dans les réseaux traditionnels, une configuration manuelle via des interfaces spécifiques aux fournisseurs[2].
- **Flexibilité du réseau :** Le SDN permet d'adapter rapidement le comportement du réseau. Grâce à une gestion centralisée, cette souplesse facilite l'intégration de nouveaux services et la reconfiguration sans intervention matérielle locale.
- **Réseau de contrôle complexe :** Avec le SDN, configurer un réseau est beaucoup plus facile parce que tout se gère depuis un seul endroit : le contrôleur. On peut tout surveiller et modifier sans avoir à toucher aux appareils un par un. Mais dans les réseaux classiques, c'est bien plus compliqué : si on a des centaines de routeurs ou de switches, il faut aller physiquement sur chaque machine pour les configurer. Imagine le temps perdu ! [1].
- **Configuration sujette aux erreurs :** Avec une architecture traditionnelle, configurer une dizaine d'équipements peut vite devenir compliqué et prendre du temps, ce qui peut facilement entraîner des erreurs. [3][4].
- **Innovation Réduction des coûts :** Le SDN se distingue par sa capacité à encourager l'évolution des réseaux. Si un nouveau protocole de routage doit être introduit, il peut être ajouté directement via la couche applicative, puis pris en charge par le contrôleur, qui le transmet aux équipements concernés. Cette méthode évite d'avoir

à remplacer le matériel existant et permet ainsi de réduire les dépenses tout en restant flexible, ce qui reste impossible dans les réseaux traditionnels [2].

- **Facilité d'utilisation et d'intégration** : Le SDN simplifie la gestion réseau en centralisant le contrôle, ce qui permet de configurer plusieurs équipements via une interface unique. Il facilite aussi l'intégration avec d'autres systèmes grâce à ses interfaces programmables.
- **Réaction rapide aux pannes** : Dans les réseaux classiques, la propagation des informations de panne suit un chemin en cascade entre les équipements, ce qui ralentit considérablement la réaction. À l'inverse, le SDN centralise la détection et la réponse : le contrôleur, alerté immédiatement, met à jour les routes en envoyant de nouvelles instructions aux équipements concernés.
- **Évitement des boucles** : Contrairement aux réseaux traditionnels, où des protocoles comme le STP (Spanning Tree Protocol) sont nécessaires pour éviter les boucles de commutation, le SDN n'en a pas besoin. Grâce à sa vision globale de la topologie réseau, le contrôleur peut établir des chemins optimisés et sans boucle, ce qui simplifie la gestion du trafic et évite les tempêtes de diffusion.
- **Contrôleur unique sensible aux pannes** : L'un des inconvénients majeurs du SDN réside dans la dépendance à un contrôleur central. En cas de défaillance de ce dernier, les équipements du plan de données ne peuvent plus acheminer les paquets, ce qui peut entraîner une interruption complète du service réseau.[5]
- **Contrôleur unique sensible aux attaques** : Comme le contrôleur gère tout le réseau, s'il est attaqué, l'ensemble du système devient vulnérable. Un pirate qui y accède pourrait alors prendre le contrôle du réseau et en modifier le fonctionnement à sa guise.[5]

Ainsi, le SDN représente une véritable révolution dans la gestion des réseaux, rendant ces derniers plus flexibles, programmables et évolutifs.

1.3 Historique de développement des SDN

L'architecture des réseaux informatiques a longtemps reposé sur des équipements matériels traditionnels (routeurs, commutateurs) qui intègrent à la fois les fonctions de transfert des données (plan de données) et les fonctions de décision (plan de contrôle). Cette intégration rigide rendait les réseaux peu flexibles, difficiles à configurer et coûteux à faire évoluer [1].

Face à ces limitations, la communauté scientifique et industrielle a commencé à explorer de nouvelles approches dès les années 2000. Une étape décisive a été franchie avec l'émergence du SDN, qui propose une séparation claire entre le plan de contrôle et le plan de données. Cette idée a été concrétisée par le projet OpenFlow au sein de l'université de Stanford, l'un des premiers protocoles à permettre cette séparation et à centraliser le contrôle du réseau via un contrôleur logiciel [6].

Depuis cette avancée, le SDN a connu une adoption croissante, notamment dans les data centers, chez les fournisseurs de services cloud, dans les environnements de recherche et dans l'infrastructure des opérateurs télécoms. Il permet une gestion plus fine du trafic, une automatisation des tâches de configuration et une plus grande souplesse dans l'introduction de nouvelles politiques réseau [5]. Des plateformes open source telles qu'Open-

Daylight, ONOS (Open Network Operating System) et RYU ont joué un rôle clé dans sa démocratisation [7].

Aujourd'hui, le SDN est également un pilier dans des domaines en pleine expansion comme la 5G, la virtualisation des fonctions réseau (NFV) et la cybersécurité, où la programmabilité du réseau offre des capacités de détection et de réaction avancées face aux attaques [1][8].

En somme, le SDN constitue une rupture technologique majeure dans l'ingénierie réseau et représente une solution idéale pour les environnements nécessitant agilité, évolutivité et résilience. La figure 1.2 montre l'ordre chronologique de son évolution.

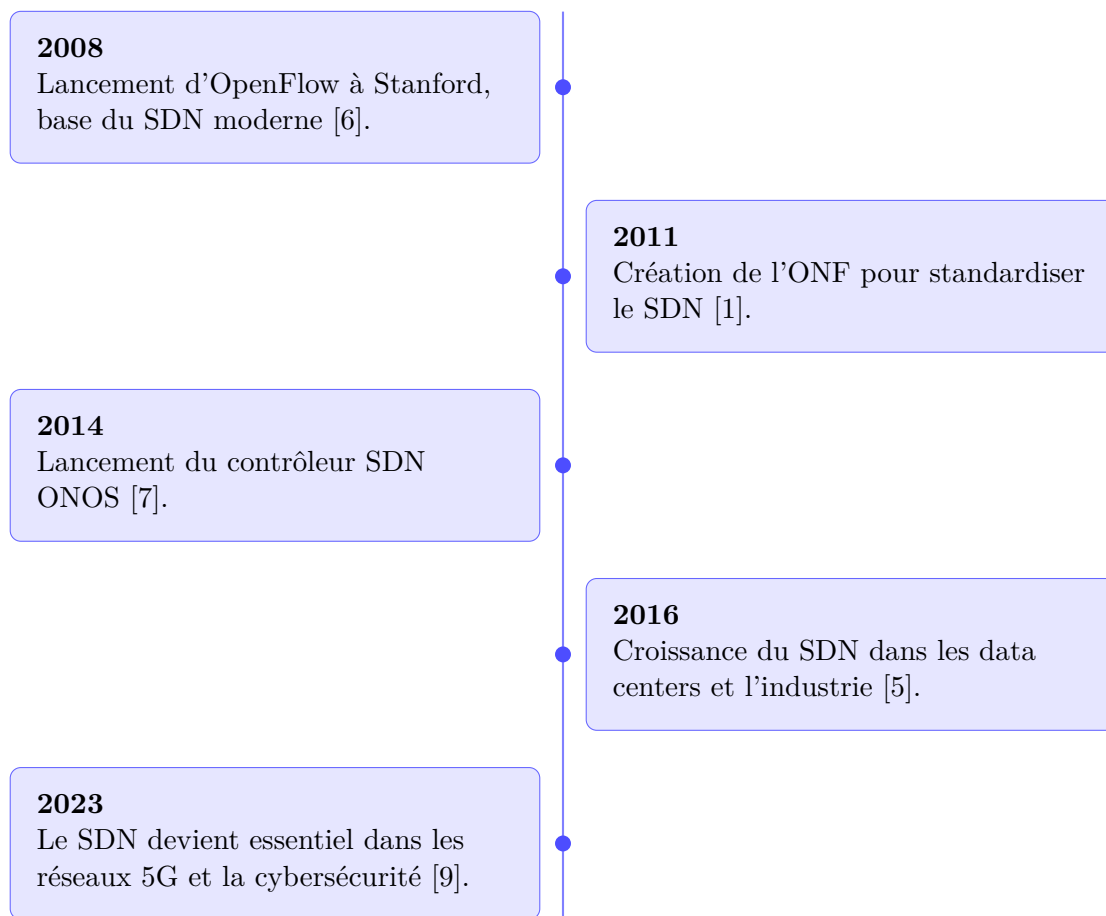


FIG. 1.2 : L'évolution du Software-Defined Networking (SDN)

1.4 Présentation de la technologie SDN

Les SDN repose sur l'utilisation d'un contrôleur chargé de la gestion de l'infrastructure réseau. Cette approche est structurée autour de trois couches principales : le plan applicatif (application plane), le plan de contrôle (control plane) et le plan de données (data plane).

1.4.1 La couche de contrôle

Il s'agit de la couche qui comporte le contrôleur, qui agit comme le **cerveau** du réseau et qui assure la communication entre les deux autres couches grâce à des interfaces de programmation appelées API. D'une part, les Southbound APIs permettent l'échange d'informations entre la couche de contrôle et la couche de données. D'autre part, les Northbound APIs facilitent la communication entre la couche de contrôle et la couche applicative. Cela est illustré par la figure 1.3.

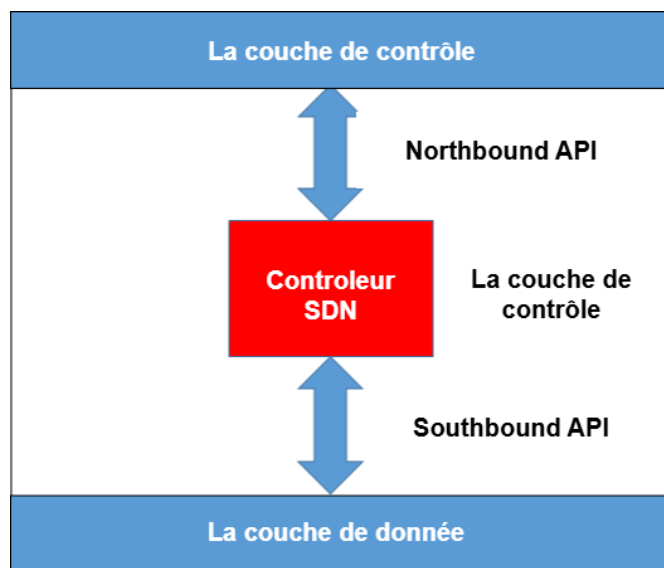


FIG. 1.3 : Architecture SDN

1.4.2 La couche d'application

La couche d'application dans une architecture SDN (Software-Defined Networking) regroupe un ensemble de services intelligents qui exploitent les informations du réseau pour automatiser, contrôler ou améliorer son fonctionnement. Elle inclut notamment des outils d'orchestration permettant de gérer dynamiquement les ressources réseau selon les besoins des applications ou des utilisateurs [10].

1.4.3 La couche de données

Cette couche regroupe l'ensemble des équipements physiques et virtuels chargés d'acheminer les paquets selon les règles de flux définies par le contrôleur SDN. Ces dispositifs illustrés dans la figure 1.4 (**commutateurs matériels** ou **vSwitches logiciels**) exposent une interface southbound standardisée permettant l'installation, la modification et la suppression dynamiques des règles de traitement des paquets. OpenFlow en constitue l'implémentation la plus connue, mais d'autres protocoles et connecteurs peuvent également jouer ce rôle, pourvu qu'ils garantissent une abstraction uniforme entre le plan de contrôle et le plan de données [1].

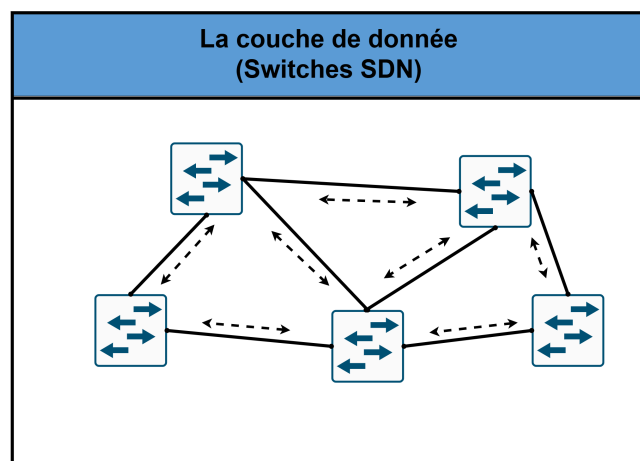


FIG. 1.4 : vSwitches (les composants de la couche de données)

1.4.4 Interface Nord (Northbound API)

Le Network Operating System (NOS) offre une interface destinée aux développeurs souhaitant concevoir des applications réseau. Cette interface, appelée API nord (northbound), sert de pont entre le contrôleur SDN et les applications externes. Elle permet d'interagir avec l'infrastructure réseau de manière simplifiée et standardisée [1]. Parmi les types d'interfaces couramment utilisés, on retrouve la **REST API**[10], qui facilite la communication via des requêtes structurées. Tout d'abord, les applications peuvent être développées dans des langages comme Java ou Python, puis, à l'aide d'outils comme Postman, ce code se transforme au format JSON ou XML (format standardisé).

1.4.5 Interface Sud (Southbound API)

Un élément fondamental dans les SDN réside dans la connexion entre le plan de données et le plan de contrôle. Le protocole **OpenFlow** représente l'une des meilleures solutions permettant d'assurer la communication entre le matériel de commutation et le contrôleur réseau. Il définit précisément comment ces deux entités interagissent et constitue donc une implémentation concrète des échanges entre un contrôleur et les commutateurs[5].

1.5 Protocoles de communication dans le SDN

Dans les architectures SDN, plusieurs protocoles ont été développés pour assurer la communication entre le plan de contrôle (contrôleur) et le plan de données (commutateurs et routeurs). Ces protocoles permettent au contrôleur d'installer, de modifier ou de supprimer dynamiquement des règles sur les équipements réseau.

Parmi les principaux protocoles de communication utilisés dans le SDN, on peut citer :

- **OpenFlow** : OpenFlow a été proposé en 2008, c'est le protocole le plus connu dans SDN [11]. Il définit la communication directe entre un contrôleur et les équipements de commutation, permettant une gestion fine des flux de données à travers des tables de flux [12].

- **SNMP (Simple Network Management Protocol)** : Proposé à la fin des années 80, le SNMP est un protocole de gestion de réseau très largement déployé [13]. Initialement conçu pour la configuration des équipements réseau, il est aujourd'hui principalement utilisé pour la surveillance des performances et la détection des pannes. Malgré son succès, SNMP présente plusieurs limites, notamment un manque de sécurité, qui a été partiellement corrigé dans ses versions ultérieures [11].
- **NETCONF (Network Configuration Protocol)** : En 2006, le groupe de travail IETF Network Configuration a proposé le protocole NETCONF comme protocole de gestion pour modifier la configuration des dispositifs réseau. Ce protocole permet aux équipements réseau d'exposer une API à travers laquelle des données de configuration extensibles peuvent être envoyées et récupérées. [14].

Bien que plusieurs protocoles existent, ce mémoire se concentrera principalement sur l'utilisation d'**OpenFlow**, car il reste le protocole de référence pour la mise en œuvre des communications entre le contrôleur et les équipements de données dans de nombreux environnements SDN.

1.6 Réseau OpenFlow

Le réseau OpenFlow constitue l'une des premières implémentations concrètes du paradigme SDN. Ce réseau repose sur trois éléments fondamentaux : le **contrôleur SDN**, qui agit comme un cerveau central, les **commutateurs OpenFlow (OpenFlow switch)**, qui exécutent les règles envoyées par le contrôleur pour gérer les flux de données, et enfin le **protocole OpenFlow**, qui permet la communication entre le contrôleur et les équipements réseau (OpenFlow) [15].

1.6.1 Le Contrôleur SDN

Le contrôleur SDN est l'élément central dans une architecture OpenFlow. Il prend toutes les décisions de gestion et de routage du trafic. Contrairement aux réseaux traditionnels, où chaque équipement réseau possède une intelligence locale, le contrôleur centralise la logique de contrôle, offrant ainsi une vue globale et cohérente du réseau [1].

Son rôle est **double** : d'une part, il communique avec les équipements de la couche inférieure via les interfaces *Southbound* (comme OpenFlow), afin de configurer dynamiquement les tables de flux (flow table) et gérer le trafic ; d'autre part, il interagit avec les applications réseau (la couche supérieure) à travers les interfaces *Northbound*, généralement sous forme d'APIs REST. Cela permet aux applications de haut niveau (comme les pare-feux, systèmes de surveillance ou de routage dynamique) d'exploiter l'état du réseau, d'appliquer des politiques, ou encore d'optimiser l'utilisation des ressources [11].

Il existe deux types d'architectures de contrôleur SDN centralisé et distribué :

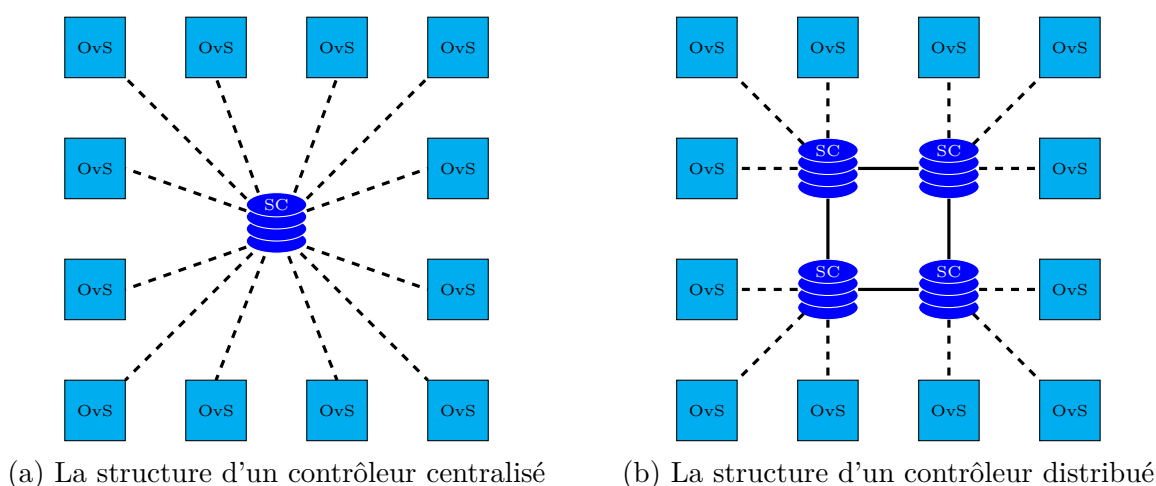


FIG. 1.5 : Comparaison de la structure des contrôleurs SDN centralisé et distribué.

- **Contrôleur centralisé** : comme illustré sur la figure 1.5a, un seul contrôleur gère l'ensemble du réseau. Cette approche est simple à déployer mais présente un risque de point de défaillance unique.
- **Contrôleur distribué** : on peut voir dans la figure 1.5b que plusieurs instances de contrôleurs coopèrent pour assurer la redondance, la tolérance aux pannes et une meilleure scalabilité. Cette architecture distribuée repose également sur des mécanismes de *load balancing* entre les contrôleurs. La charge de gestion du réseau (ex. contrôle des switches, gestion des flux, collecte des statistiques) est répartie entre plusieurs instances afin d'éviter les goulots d'étranglement et de garantir une meilleure performance globale [1]. Chaque contrôleur peut être assigné à un sous-ensemble de switches, et un système de basculement permet à un autre contrôleur de reprendre la main en cas de défaillance. Ce type de répartition dynamique de la charge améliore la résilience, la disponibilité, et la scalabilité de l'infrastructure SDN [16] [17].

Le tableau 1.2 présente une comparaison entre les approches centralisée et distribuée des contrôleurs SDN

TAB. 1.2 : Comparaison des contrôleurs SDN centralisé et distribué.

Critère	Centralisé	Distribué
Simplicité de déploiement	✓	✗
Tolérance aux pannes	✗	✓
Latence de décision	✓	✓
Scalabilité	✗	✓
Risque de point unique de défaillance	✓	✗
Complexité de gestion	✗	✓

Il existe aujourd'hui un grand nombre de contrôleurs SDN développés pour répondre à différents besoins, avec des architectures variées. On distingue principalement deux catégories : les **contrôleurs open source**, qui sont libres d'utilisation et modifiables, et

les **contrôleurs propriétaires**, développés par des entreprises comme *Cisco*, *Juniper* ou encore *HP*, souvent intégrés dans des solutions commerciales.

Dans le cadre de ce travail, nous nous intéressons uniquement aux **contrôleurs open source**, car ils sont gratuits (free) et peuvent être utilisés avec des équipements de marques différentes grâce à leur compatibilité multi-vendeurs. Parmi les contrôleurs open source les plus utilisés, on peut citer : ODL, ONOS et RYU qui seront présentés avec plus de détails dans cette suite :

- ODL (OpenDaylight)

OpenDaylight est un contrôleur SDN open source majeur, annoncé en **avril 2013** sous l'impulsion de l'*Open Network Foundation (ONF)*, elle-même créée dans le cadre de la *Linux Foundation* [18]. Ce projet a été lancé avec la participation de nombreux industriels pour répondre aux limites observées dans les premiers contrôleurs SDN (voir figure 1.6a).

ODL est **développé en Java** et s'appuie sur l'architecture du *contrôleur Beacon*. Il introduit des fonctionnalités clés comme la **haute disponibilité**, le **clustering** et la **tolérance aux pannes**, ce qui le rend adapté aux **réseaux à grande échelle**. Son architecture distribuée repose sur l'algorithme *Raft* [19], permettant l'élection d'un contrôleur **leader** parmi les autres (appelés *followers*) et la gestion centralisée des mises à jour de données. Avec le soutien de nombreux fournisseurs, ODL est devenu l'un des plus **grands projets de contrôleurs SDN open source**. Il a fortement influencé le secteur commercial des SDN, avec de nombreux contrôleurs basés sur son architecture. Aujourd'hui, ODL est utilisé dans plus de **150 organisations à travers le monde**, notamment par des opérateurs télécoms (couvrant collectivement près d'un milliard d'abonnés), des fournisseurs de contenu web, des entreprises, des centres de recherche et des universités.

- ONOS (Open Network Operating System)

Lancé par la Linux Foundation, ONOS est une plateforme de contrôleur SDN conçue pour faciliter la transition des réseaux traditionnels « brownfield » vers des réseaux SDN « greenfield », permettant un déploiement plus rapide et moins coûteux **onos2019**. Développé en Java, ONOS se distingue par son architecture distribuée garantissant performances élevées, disponibilité maximale et forte capacité de montée en charge. Cette plateforme est utilisée à l'échelle mondiale par des fournisseurs de services, des entreprises, des centres de recherche et diverses autres organisations (voir figure 1.6b).

- RYU

Ryu est un framework SDN open source basé sur des composants, entièrement implémenté en Python, bien qu'il supporte aussi des modules écrits dans d'autres langages **ryu2019**. Son objectif est de fournir un système d'exploitation SDN robuste et de haute qualité, adapté à de grands réseaux. Ryu inclut la gestion des événements, le suivi d'état en mémoire, la gestion d'applications et une série de bibliothèques réutilisables (NetCONF, sFlow/NetFlow, OF-Config), offrant ainsi une grande flexibilité (voir figure 1.6c).



(a) OpenDayLight



(b) ONOS



(c) RYU

FIG. 1.6 : Logos des différents contrôleurs SDN

1.6.2 Le protocole OpenFlow

Le protocole **OpenFlow** constitue l'un des piliers fondamentaux des SDN. Il a été développé à l'origine par l'Université de Stanford en 2008 dans le cadre du projet Clean Slate, puis standardisé par l'ONF. Ce protocole est conçu pour assurer l'interaction entre le plan de contrôle, représenté par le contrôleur SDN, et le plan de données, représenté par les commutateurs OpenFlow [20].

OpenFlow permet au contrôleur de dicter dynamiquement le comportement des commutateurs en leur envoyant des instructions via un **canal sécurisé** (Secure Channel) ou une session TCP (voir figure 1.8), typiquement chiffré par TLS. Ces communications permettent de gérer les règles de traitement des paquets au sein des tables de flux (flow tables) des commutateurs [15].

Le protocole OpenFlow définit trois grandes catégories de messages pour la communication entre le contrôleur et le commutateur, la figure suivante illustre ces messages :

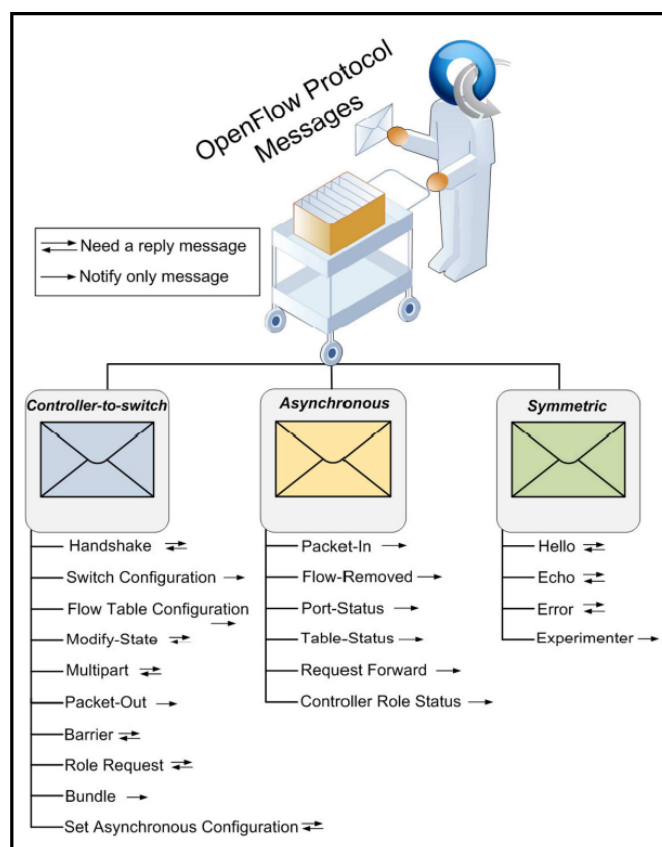


FIG. 1.7 : Les messages du protocole OpenFlow [15]

- **Messages du contrôleur vers le commutateur** : ils sont utilisés pour affirmer le contrôle du contrôleur sur le commutateur, lire son état, ou modifier ses tables de flux.
- **Messages asynchrones (du commutateur vers le contrôleur)** : ils servent à notifier le contrôleur de l'arrivée d'un nouveau flux, d'un changement d'état du commutateur, ou encore pour demander une modification de table.
- **Messages symétriques** : ceux-ci peuvent être initiés par le contrôleur ou le commutateur. Ils incluent les messages *Hello*, *Echo*, *Error* et *Experimenter*, ce dernier servant notamment à identifier le fournisseur du matériel ou du logiciel utilisé [21].

Les versions du protocole OpenFlow

Le protocole OpenFlow a connu plusieurs évolutions depuis sa première publication, chacune apportant de nouvelles fonctionnalités et améliorations destinées à répondre aux exigences croissantes des réseaux programmables. Afin de mieux comprendre cette évolution, le tableau 1.3 présente une comparaison synthétique des principales propriétés des versions OpenFlow 1.0 à 1.5.

TAB. 1.3 : Les différentes versions d'OpenFlow de 1.0 à 1.5 [15]

Version / Propriété	1.0	1.1	1.2	1.3	1.4	1.5
Date de publication	31 déc. 2009	28 févr. 2011	5 déc. 2011	25 juin 2012	15 oct. 2013	déc. 2014
Largement déployée	Oui	Oui	Non	Non	Non	Non
Table de flux	unique	multiple	multiple	multiple	multiple	multiple
Table de groupe	Non	Oui	Oui	Oui	Oui	Oui
Table de mesure	Non	Non	Non	Oui	Oui	Oui
VLAN et étiquette MPLS	Non	Oui	Oui	Oui	Oui	Oui
Mode de repli de connexion contrôleur	Urgence	Autonome	Autonome / sécurisé	Autonome / sécurisé	Autonome / sécurisé	Autonome / sécurisé
Support IPv6	Non	Non	Oui	Oui	Oui	Oui
Contrôleur multiple	Non	Non	Oui	Oui	Oui	Oui
Éviction / vacance / synchronisation	Non	Non	Non	Non	Oui	Oui
Ports optiques	Non	Non	Non	Oui	Oui	Oui

1.6.3 Les commutateurs OpenFlow

Les commutateurs OpenFlow jouent un rôle essentiel dans l'architecture SDN. Ils sont chargés d'exécuter les décisions de routage ou de traitement des paquets dictées par le contrôleur, via le protocole OpenFlow. L'architecture interne d'un commutateur OpenFlow est standardisée, et elle comprend principalement plusieurs composants critiques : les tables de flux (flow tables), la table de groupes (group table), la table de métriques (meter table) ainsi que divers ports [15] [5]. La figure 1.8 illustre l'architecture interne d'un commutateur OpenFlow, mettant en évidence ses principaux composants.

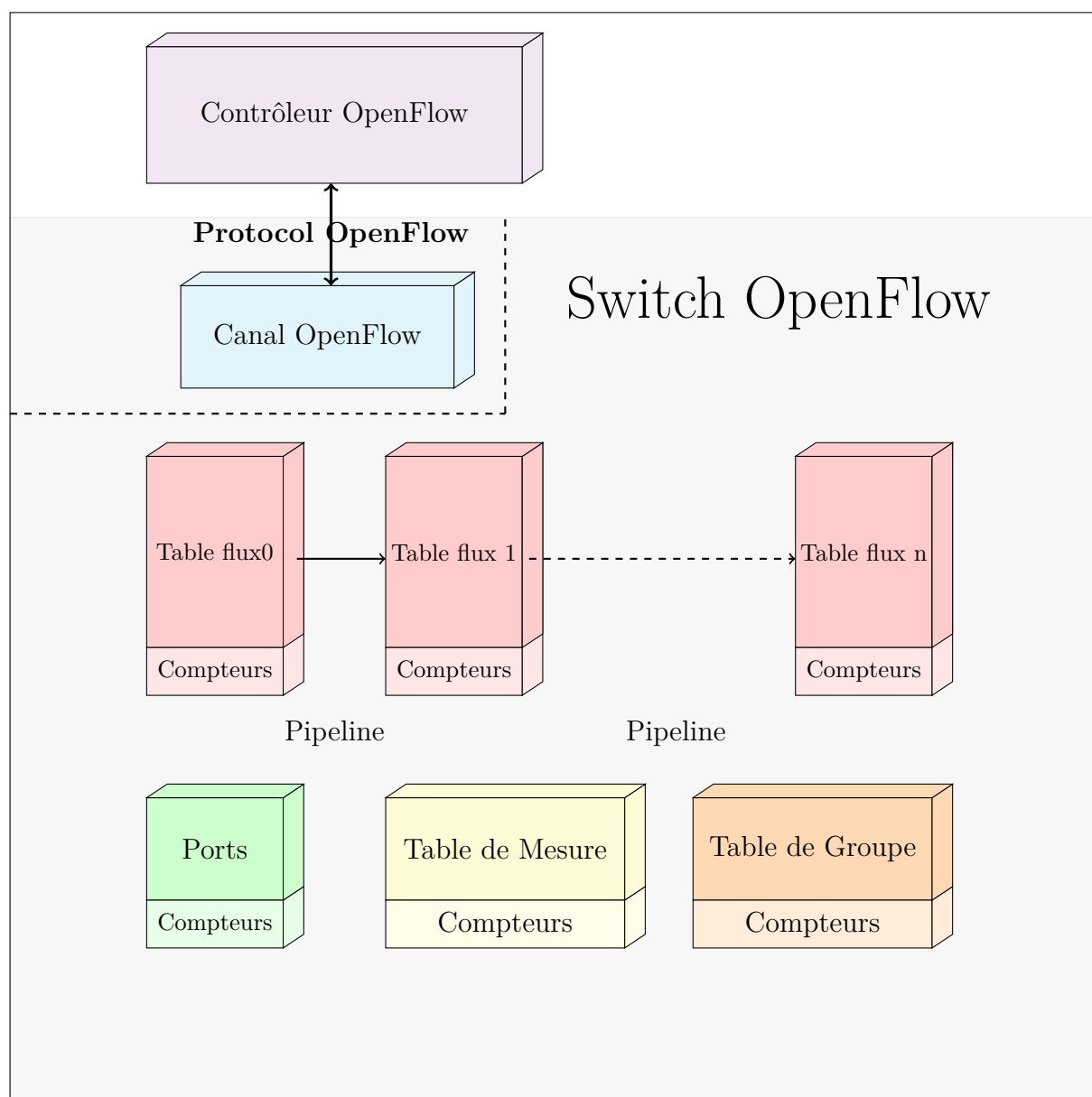


FIG. 1.8 : L'architecture du OpenFlow switch

- **Table de Groupe** : cette table permet d'implémenter des fonctionnalités plus complexes comme la redondance, la diffusion (broadcast) ou encore la répartition de charge (load balancing). Chaque entrée dans cette table spécifie un groupe d'actions à appliquer à un paquet.
- **Table de Mesure** : elle permet de mesurer et de contrôler le débit du trafic réseau. Les entrées de cette table définissent des règles de limitation de bande passante (rate-limiting) et peuvent être utilisées pour prioriser certains flux par rapport à d'autres, ou pour la gestion de la qualité de service (QoS).
- **Port et compteurs** : les commutateurs disposent également de ports physiques ou logiques, utilisés pour envoyer ou recevoir les paquets. Chaque table et chaque port sont associés à des compteurs permettant de collecter des statistiques sur le nombre de paquets ou d'octets traités, facilitant ainsi la supervision du réseau par le contrôleur.
- **Table de flux** : il s'agit du cœur de la logique de traitement des paquets. Chaque table de flux (flow table) contient 256 entrées de flux (flow entries) [15], qui définissent des règles de correspondance (match fields), des instructions d'action (instructions) et des compteurs (counters) associés. La figure 1.9 illustre la structure typique d'une entrée de flux (flow entry). Dans ce qui suit, nous allons examiner les différents modes d'installation de ces entrées dans la table de flux

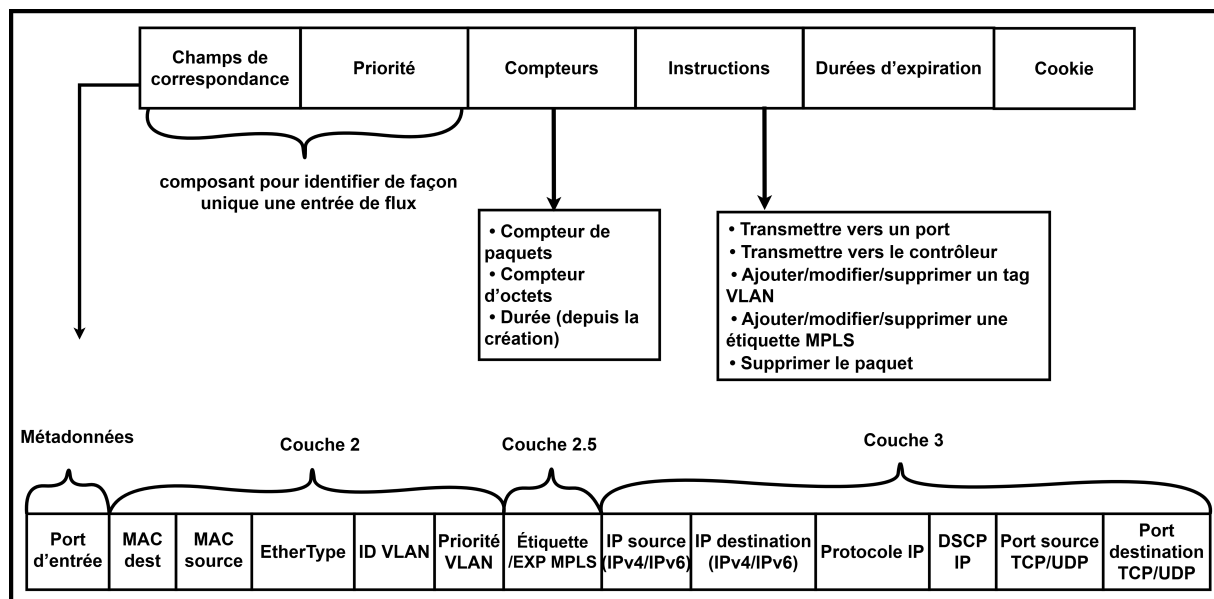


FIG. 1.9 : Structure détaillée d'une entrée de flux dans un commutateur OpenFlow

Lorsqu'un paquet est reçu par le commutateur OpenFlow, l'**en-tête** (header) de ce paquet est comparé avec la section **champs de correspondance** des entrées de flux (flow entries) dans la table de flux (flow table) [15].

Le schéma 1.10 illustre le déroulement détaillé de ce processus.

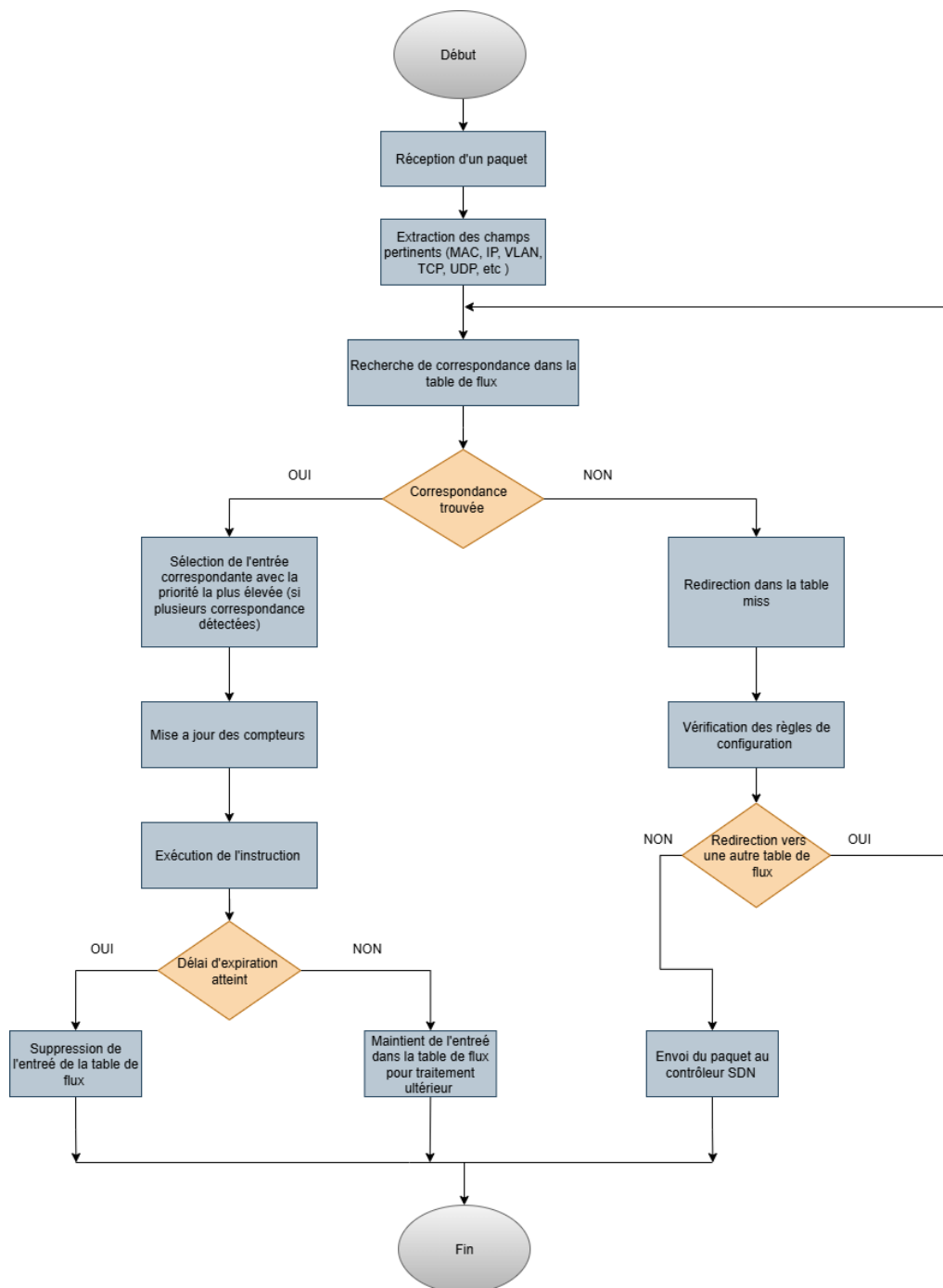


FIG. 1.10 : Déroulement du traitement d'un paquet dans une table des flux

L'installation des entrées dans les tables de flux peut se faire de deux manières, en envoyant des messages **flow_mod** au commutateur : de manière **réactive** ou **proactive**[16][20], comme illustré dans la figure 1.11

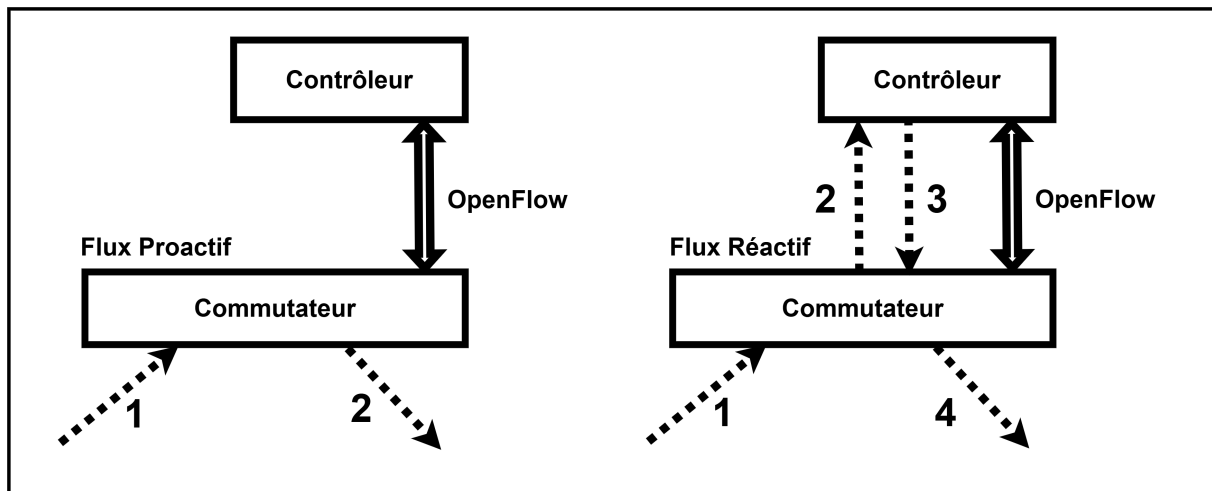


FIG. 1.11 : Comparaison des modes proactif et réactif

En **mode réactif**, généralement le contrôleur est dans le LAN. Lorsqu'un commutateur reçoit le premier paquet d'un nouveau flux, il génère un événement **PacketIn** et envoie ce paquet au contrôleur[20]. Ce dernier décide ensuite, selon la logique de l'application, de l'action à entreprendre. Si la décision est d'installer une entrée de flux, le contrôleur l'ajoute à la table de flux avec une durée d'expiration définie (*idle timeout* et *hard timeout*). Ce mode permet d'économiser la mémoire, souvent limitée, des commutateurs. Cependant, il introduit une latence due à l'aller-retour entre le commutateur et le contrôleur, et présente un risque fonctionnel en cas de perte de connexion ou d'attaque ciblée sur le contrôleur.

À l'inverse, le **mode proactif** du contrôleur est loin du LAN. Ce dernier consiste à préinstaller les règles dans les tables de flux dès l'établissement de la connexion (**ConnectionUp**) avec le canal sécurisé du Ov. Cela permet de réduire la latence au moment du traitement des paquets et garantit un fonctionnement continu même en cas de perte de communication avec le contrôleur. Néanmoins, étant donné le coût élevé de la mémoire dans les commutateurs, les règles proactives sont souvent généralisées à l'aide d'entrées à *masques larges* (*wildcard entries*) pour représenter des flux agrégés.

1.7 Conclusion

Au fil de ce chapitre, nous avons exploré l'architecture SDN, ses composants essentiels (contrôleurs, commutateurs OpenFlow, protocoles) et les principaux mécanismes d'installation des règles de flux. Toutefois, au-delà de la théorie, la compréhension et la maîtrise du SDN nécessitent une mise en pratique concrète.

Le SDN trouve des applications dans les datacenters, les réseaux de télécommunications, les environnements cloud ou encore la cybersécurité, où il permet d'automatiser les politiques de sécurité réseau [20]. Pour approfondir ce domaine, il est donc essentiel de coupler l'étude théorique à des expérimentations dans des environnements réalistes, et c'est précisément l'objet des chapitres suivants.

En définitive, cette approche programmée des infrastructures réseau marque une véritable rupture vers des systèmes toujours plus dynamiques, adaptatifs et faciles à gérer, répondant pleinement aux exigences actuelles de flexibilité et d'automatisation.

Chapitre 2

Analyse de l'existant et optimisation

2.1 Introduction

Après avoir exploré l'état de l'art des SDN dans le chapitre précédent incluant leur définition, leur historique, leur architecture et les protocoles essentiels tels qu'OpenFlow.

Ce chapitre débute par une analyse critique de l'architecture actuelle, avant de présenter notre proposition pour une nouvelle solution. Nous identifions d'abord les points faibles majeurs du système en place, puis détaillons la conception de notre architecture renouvelée, fondée sur le paradigme SDN et organisée selon un modèle hiérarchique modulaire. Après avoir rappelé les principes et atouts d'une topologie en trois couches accompagnée d'une zone démilitarisée (DMZ), nous expliquons sa déclinaison pour chacun des cinq blocs du campus (A à E) et en exposons la vision globale.

L'objectif est de montrer comment l'approche SDN assure redondance, scalabilité et sécurité périmétrique, tout en simplifiant l'exploitation et l'automatisation des services réseau.

2.2 Présentation de l'architecture actuelle

L'architecture réseau étudiée est répartie sur trois blocs (bloc A, bloc B, bloc C et D présenté comme un seul bloc), reliés entre eux selon une topologie en cascade. Chaque bloc dispose d'un seul commutateur (switch) qui assure la distribution du réseau au sein du bâtiment. Ces commutateurs sont interconnectés de manière séquentielle, formant une structure linéaire simple.

L'ensemble du réseau est piloté par un unique routeur, placé à la tête de l'architecture, qui assure la gestion du trafic vers l'extérieur. Plusieurs serveurs sont déployés dans l'environnement réseau, mais ils sont représentés de manière simplifiée par un seul élément centralisé dans la figure 2.1, désigné comme un "data center".

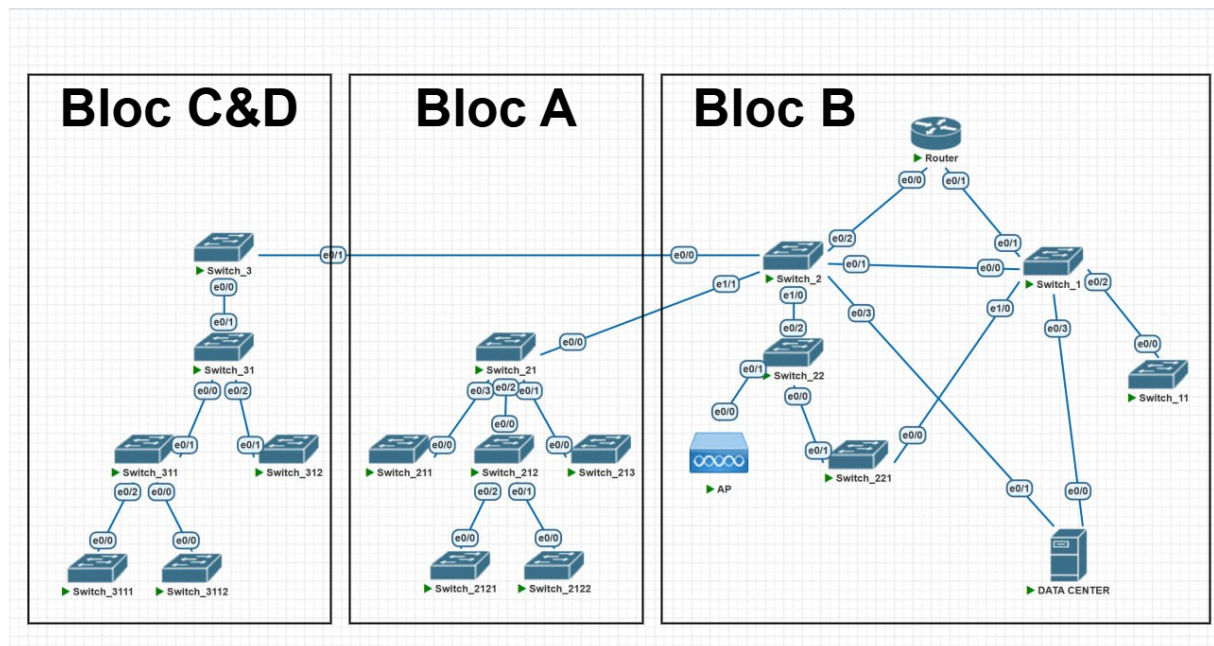


FIG. 2.1 : L'architecture actuelle du réseau

2.3 Analyse des failles et des limites

Suite à l'observation et à l'analyse de l'architecture présentée précédemment, plusieurs points de faiblesse et de défaillance peuvent être dégagés, notamment :

1. **Architecture en cascade** : une architecture en cascade consiste à relier les équipements réseau (commutateurs, routeurs) de manière linéaire, c'est-à-dire qu'un équipement dépend directement du précédent. Cette méthode est simple à mettre en place, mais présente plusieurs inconvénients majeurs :
 - **Propagation des pannes** : une défaillance unique peut entraîner une panne généralisée sur tout un segment du réseau.
 - **Performance dégradée** : chaque nœud (commutateur, routeur) ajoute de la latence, particulièrement si le trafic doit traverser plusieurs équipements pour atteindre sa destination.
 - **Difficulté de maintenance** : toute modification ou panne dans une partie du réseau nécessite souvent une interruption pour tout le segment en aval.
 - **Faible évolutivité** : l'ajout de nouveaux équipements devient rapidement complexe à mesure que la profondeur du réseau augmente.
2. **Absence de redondance (liens et matériel)** : l'architecture réseau analysée souffre d'un manque flagrant de redondance, tant au niveau des connexions que des équipements. D'une part, l'absence de liaisons redondantes entre les commutateurs introduit des points de défaillance critiques : par exemple, si le lien entre le switch_3 et Switch_2 (entre le bloc B et le bloc C&D) est rompu, l'ensemble du bloc C&D devient hors service, entraînant une indisponibilité totale de cette section du réseau. D'autre part, la dépendance à un unique routeur pour l'accès à Internet constitue un point de défaillance unique. Une panne de ce routeur rendrait le réseau incapable de se connecter à l'extérieur, affectant gravement la continuité des services. Cette double vulnérabilité réduit considérablement la fiabilité et la résilience globale du système.
3. **Ancienneté du pare-feu** : Le réseau utilise un pare-feu ancien qui ne correspond pas aux standards des pare-feux de type "next-generation". Il fonctionne principalement sur les couches 3 et 4 du modèle OSI, ce qui limite son analyse au niveau des adresses IP, ports et protocoles. Ce dispositif présente des limites notables face aux menaces actuelles telles que les attaques par injection. Il ne dispose pas des capacités avancées de détection, d'analyse approfondie du trafic ou de prévention des intrusions, ce qui compromet la sécurité globale du réseau face aux cybermenaces modernes.
4. **Présence des serveurs (le centre de données) dans le LAN** : Actuellement, les serveurs (Mail, Web, DNS, ...) sont directement intégrés dans le réseau local LAN interne, ce qui constitue une faille de sécurité majeure. Cette configuration expose tout le réseau interne aux attaques externes. En effet, si un pirate parvient à compromettre un serveur exposé publiquement, il peut alors accéder facilement au reste du réseau interne, y compris aux postes utilisateurs, aux bases de données sensibles ou aux équipements critiques.

5. **Commutateurs obsolètes** : Le réseau repose sur des commutateurs Cisco déclarés "End of Life" (EoL), c'est-à-dire en fin de support par le constructeur. Ces équipements ne bénéficient plus de mises à jour logicielles ni de correctifs de sécurité, ce qui représente un risque important pour la fiabilité, la stabilité et la sécurité du réseau. Leur maintenance devient également plus difficile, et leur compatibilité avec des technologies est limitée.
6. **Systèmes d'exploitation dépassés** : Plusieurs machines du réseau fonctionnent avec des systèmes d'exploitation anciens, qui ne sont plus maintenus par leurs éditeurs. Par exemple, certains postes utilisent encore Windows 7, un système dont le support officiel a été interrompu par Microsoft depuis le 14 janvier 2020 [22]. L'absence de mises à jour de sécurité rend ces systèmes particulièrement vulnérables aux malwares, aux failles critiques non corrigées et aux attaques ciblées. Cette vétusté nuit également à la compatibilité avec les logiciels récents et aux performances générales des postes de travail.

2.4 Étude de terrain pour dimensionnement du système réseau

L'objectif de cette étude de terrain est de réaliser un inventaire précis et une cartographie détaillée de l'infrastructure réseau existante sur les cinq blocs du site (A, B, C, D et le nouveau bloc E). Cette phase, base de la conception SDN ultérieure, consiste à :

- Identifier l'emplacement des commutateurs pour chaque bloc.
- Décrire les liaisons physiques (cuivre, fibre).
- Vérifier les modalités d'empilement (stacking) et la redondance des liens.

Description physique des blocs

Bloc A Rez-de-chaussée (RDC) + 3 étages, chaque niveau ayant besoin minimum de 80 hôtes et étant en liaison directe avec le bloc B.

Bloc B RDC + 2 étages, abritant un centre de données au RDC avec :

- RDC pour la centralisation de réseau,
- 2 étages contiennent environ 120 hôtes,
- 2 routeurs pour l'accès à l'internet pour la redondance des flux.

Bloc C RDC + 1 étage : le bloc a besoin d'environ 160 hôtes et en liaison directe vers D et vers le bloc B.

Bloc D RDC uniquement, lien vers le bloc C.

Bloc E (nouveau) Sous-sol + RDC + 5 étages.

Le tableau suivant présente une synthèse complète de la structure physique des blocs.

TAB. 2.1 : Présentation des blocs et de leurs interconnexions

Bloc	Description	Connexions principales
A	Rez-de-chaussée + 3 étages, 80 hôtes par étage	Liaisons montantes vers les switches de distribution du Bloc B
B	RDC + 2 étages, centre de données au RDC (routage, stockage...), 120 hôtes sur 2 étages, 2 routeurs redondants Internet	Point central de distribution pour A, C, E
C	RDC + 1 étage, 160 hôtes	Liaison directe vers Bloc B (distribution) et Bloc D
D	RDC uniquement	Raccordé uniquement au Bloc C au niveau RDC
E	Sous-sol + RDC + 5 étages	L'empilage des étages monte jusqu'au Bloc B.

Les interconnexions principales sont réalisées de sorte que :

- Tous les commutateurs de A, C et E montent jusqu'aux switches de distribution du bloc B.
- Le bloc D est relié uniquement au bloc C au niveau du RDC.

2.5 Proposition d'une alternative SDN

Dans cette section, nous présentons notre proposition d'architecture réseau rénovée, conçue pour répondre aux limites identifiées de l'ancien modèle. Nous décrirons un schéma à base de SDN et de couches modulaires, assurant redondance, évolutivité et segmentation fine. Nous mettrons en avant la centralisation de la gestion et de la supervision, l'automatisation des déploiements et l'optimisation des flux, de manière à garantir performance, résilience et sécurité renforcée pour l'ensemble de l'infrastructure.

2.5.1 Modèle hiérarchique et sécurisation périmétrique

Pour garantir à la fois performance, modularité et robustesse face aux menaces extérieures, on adopte souvent un modèle hiérarchique à trois couches associé à des mécanismes de sécurisation périmétrique. Cette organisation claire permet de :

- Segmenter le réseau
- Appliquer des politiques adaptées à chaque niveau.
- Renforcer le contrôle d'accès et le filtrage du réseau.

Avantages d'une architecture hiérarchique :

Contrairement à la figure 2.2a, la figure 2.2b présente une architecture hiérarchique qui repose sur un modèle à trois couches, bien défini dans les bonnes pratiques de conception réseau [23][24] :

- **Couche d'accès (Access Layer)** : elle connecte les utilisateurs et les terminaux finaux. Elle intègre souvent des fonctions de sécurité comme le contrôle d'accès (802.1X).
- **Couche de distribution (Distribution Layer)** : elle agrège les connexions venant de la couche d'accès et implémente des politiques de sécurité, de routage, de QoS.
- **Couche cœur (Core Layer)** : elle assure un routage rapide et fiable entre les différentes parties du réseau, généralement sans traitement de filtrage.

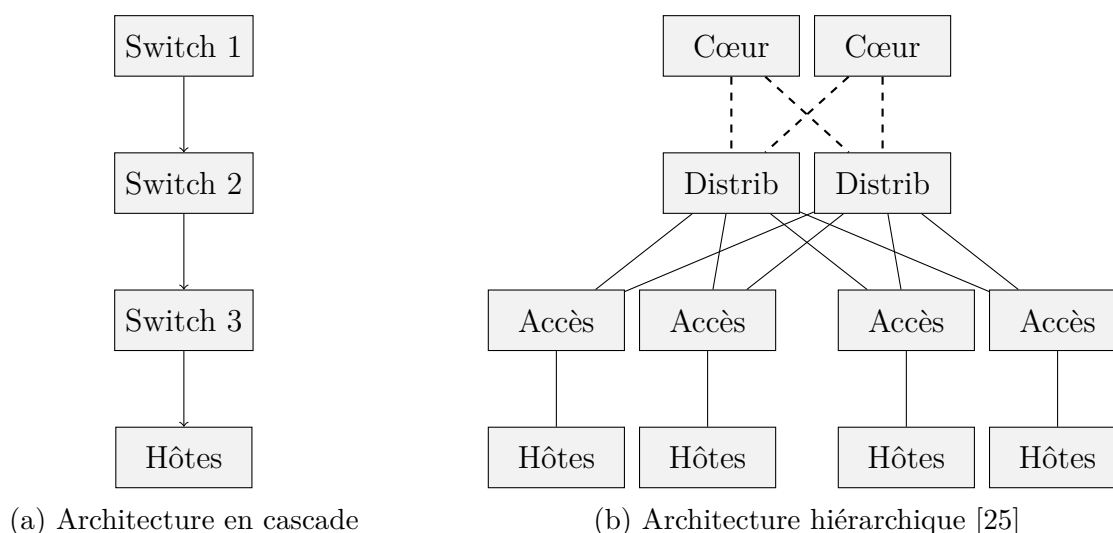


FIG. 2.2 : Comparaison des architectures en cascade et hiérarchique

Cette hiérarchie apporte de nombreux avantages :

- **Résilience** : grâce à des liaisons redondantes (ex. : STP, EtherChannel, VRRP), le réseau peut continuer de fonctionner même en cas de panne d'un lien ou d'un équipement.
- **Meilleure performance** : chaque couche étant optimisée pour des fonctions précises, le traitement du trafic est plus rapide et plus efficace.
- **Facilité d'évolutivité** : il est plus facile d'ajouter de nouveaux équipements ou segments de réseau sans impact majeur sur les couches supérieures.
- **Isolation des pannes** : une panne dans une couche n'impacte pas nécessairement les autres, ce qui améliore la disponibilité globale.

- **Maintenance ciblée** : la hiérarchie facilite l'identification et la résolution des problèmes.

Avantages d'empilage de switches :

L'empilage de switches (ou « stacking ») est une technologie qui permet de regrouper physiquement et logiquement plusieurs commutateurs réseau (switches) pour qu'ils fonctionnent comme une seule unité de gestion[26]. Grâce à une liaison dédiée (souvent appelée « stacking cable » ou « stacking module »), les switches échangent des informations de niveau contrôle et données. L'ensemble est alors administré via une seule adresse IP, un seul plan de contrôle, et apparaît comme un switch unique du point de vue des dispositifs connectés (voir la figure 2.3).

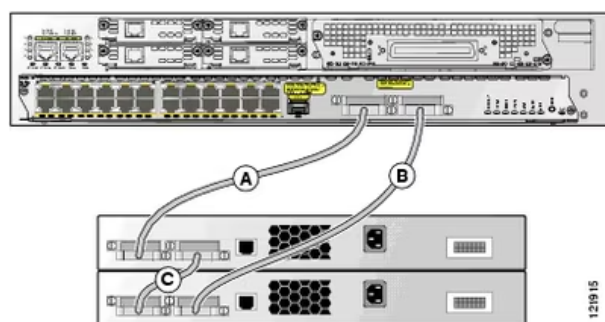


FIG. 2.3 : L'empilage de switches [27]

- **Redondance et haute disponibilité** :

Grâce au mécanisme de standby automatique, si le switch maître venait à tomber, l'un des autres membres du stack lui succède instantanément, sans coupure perceptible du réseau. Par ailleurs, la protection des liaisons montantes permet de répartir les uplinks (vers le routeur ou le cœur de réseau) sur plusieurs switches physiques, tout en les gérant comme un seul lien logique, éliminant ainsi tout point unique de défaillance.

- **Optimisation des coûts** :

Avec une gestion centralisée, on configure et maintient tout le stack depuis une seule interface, ce qui réduit le temps et les frais opérationnels. De plus, une seule licence de fonctionnalités avancées ou de support suffit pour l'ensemble des switches, ce qui allège le budget logiciel. Enfin, l'évolution modulaire permet de démarrer avec quelques unités et d'en ajouter au fur et à mesure sans remplacer l'infrastructure existante, évitant ainsi des investissements initiaux trop importants.

- **Amélioration de la qualité et des performances** :

Le lien de stacking regroupe la capacité de plusieurs switches pour offrir une bande passante très élevée, évitant ainsi les bouchons entre eux. Enfin, l'équilibrage de charge distribue le trafic sur plusieurs switches du stack, ce qui réduit la latence et renforce la fiabilité.

Sécurisation avec Démilitarisation :

Pour renforcer la sécurité périmétrique, il est essentiel d'implémenter une **Demilitarized zone DMZ**. La DMZ est une zone réseau intermédiaire entre le réseau externe (Internet)

et le réseau interne (LAN), dans laquelle on place les serveurs accessibles depuis l'extérieur. Elle permet d'isoler ces serveurs afin que, même en cas de compromission, l'attaquant ne puisse pas facilement pénétrer dans le LAN.

Deux architectures principales de DMZ sont couramment utilisées :

- **DMZ avec un seul pare-feu (Single Firewall)** : la figure 2.4 montre que une DMZ, à un seul pare-feu dispose de trois interfaces : une pour Internet, une pour la DMZ et une pour le LAN interne. Le pare-feu applique des règles strictes pour limiter les flux, par exemple, autoriser uniquement les connexions entrantes vers les serveurs dans la DMZ depuis Internet et interdire toute communication directe entre la DMZ et le LAN sans inspection. Bien que plus économique, cette architecture dépend entièrement de la configuration du pare-feu.

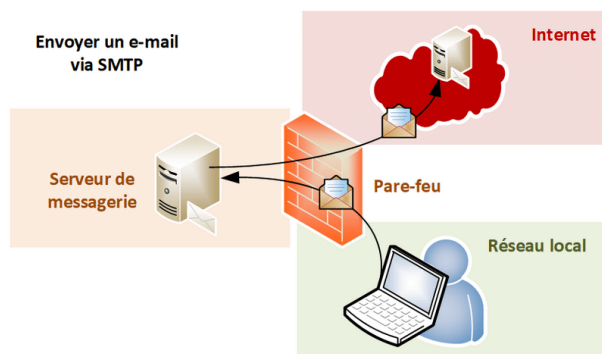


FIG. 2.4 : La DMZ avec un seul pare-feu [28]

- **DMZ avec deux pare-feux (Dual Firewall)** : la figure 2.5 plus robuste utilise deux pare-feux : le premier est connecté à Internet pour protéger l'accès à la DMZ, et le second entre la DMZ et le LAN pour protéger le réseau interne. Cette séparation physique augmente considérablement la sécurité. Même si un serveur de la DMZ est compromis, l'attaquant devra encore franchir un second pare-feu, ce qui limite fortement le risque d'intrusion dans le réseau interne.

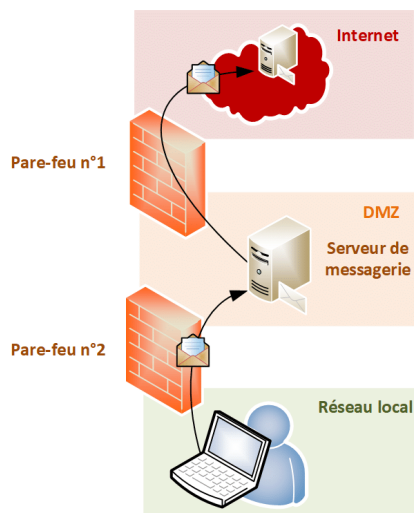


FIG. 2.5 : La DMZ avec deux pare-feu [28]

En résumé, l'intégration de ces serveurs dans une DMZ est une **pratique incontournable pour toute architecture réseau sécurisée**. Elle réduit la surface d'attaque et permet de contrôler plus finement les flux entre l'extérieur, les zones exposées, et le réseau interne. [28]

2.5.2 L'architecture réseau de chaque bloc

Dans cette partie, on va montrer la nouvelle proposition de chaque bloc, en citant le matériel et le type de câblage.

- **Bloc A** : Comme la figure 2.6 montre, ce bloc a besoin de 8 switches (2 par niveau), tous empilés afin d'assurer tout l'accès du bloc et le switch de RDC relié en fibre multimode au bloc B.

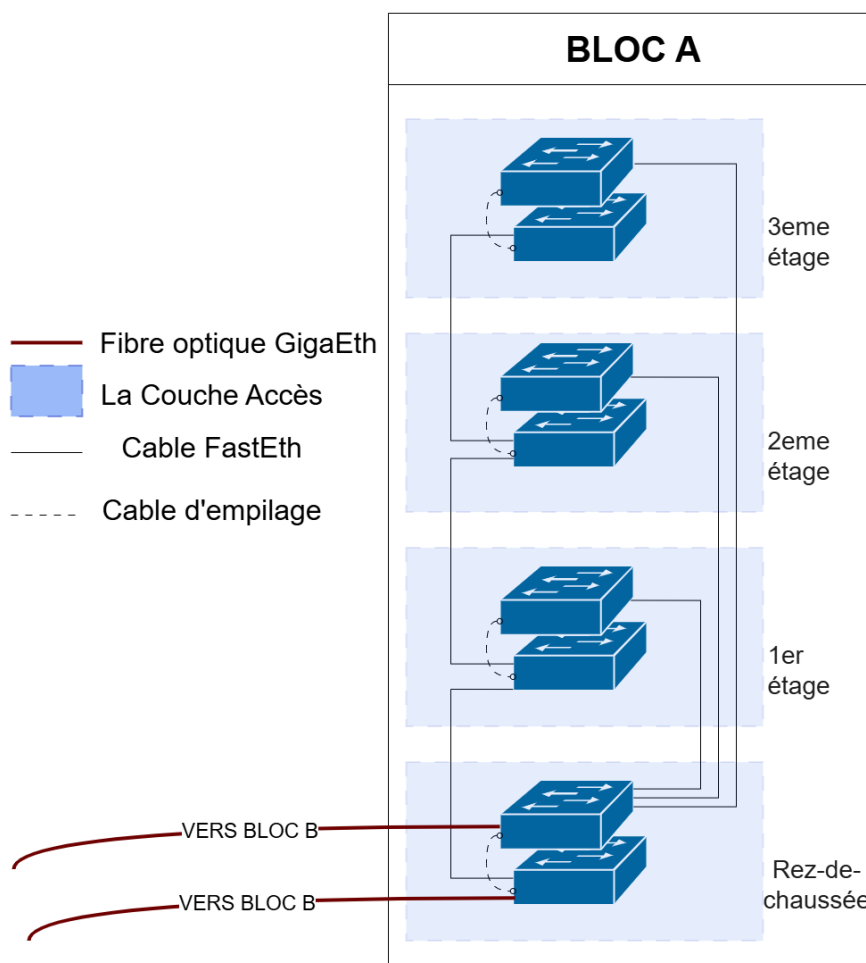


FIG. 2.6 : L'architecture proposée pour le bloc A

- **Bloc B** : la figure 2.7 montre que ce bloc est divisé en deux parties. On a le RDC qui contient le centre de données et deux switches de distribution afin d'assurer la distribution de réseau, deux routeurs pour l'accès à Internet, et les deux étages contiennent six switches empilés. Les deux switches de distribution sont en liaison fibre optique vers les blocs A, C et E.

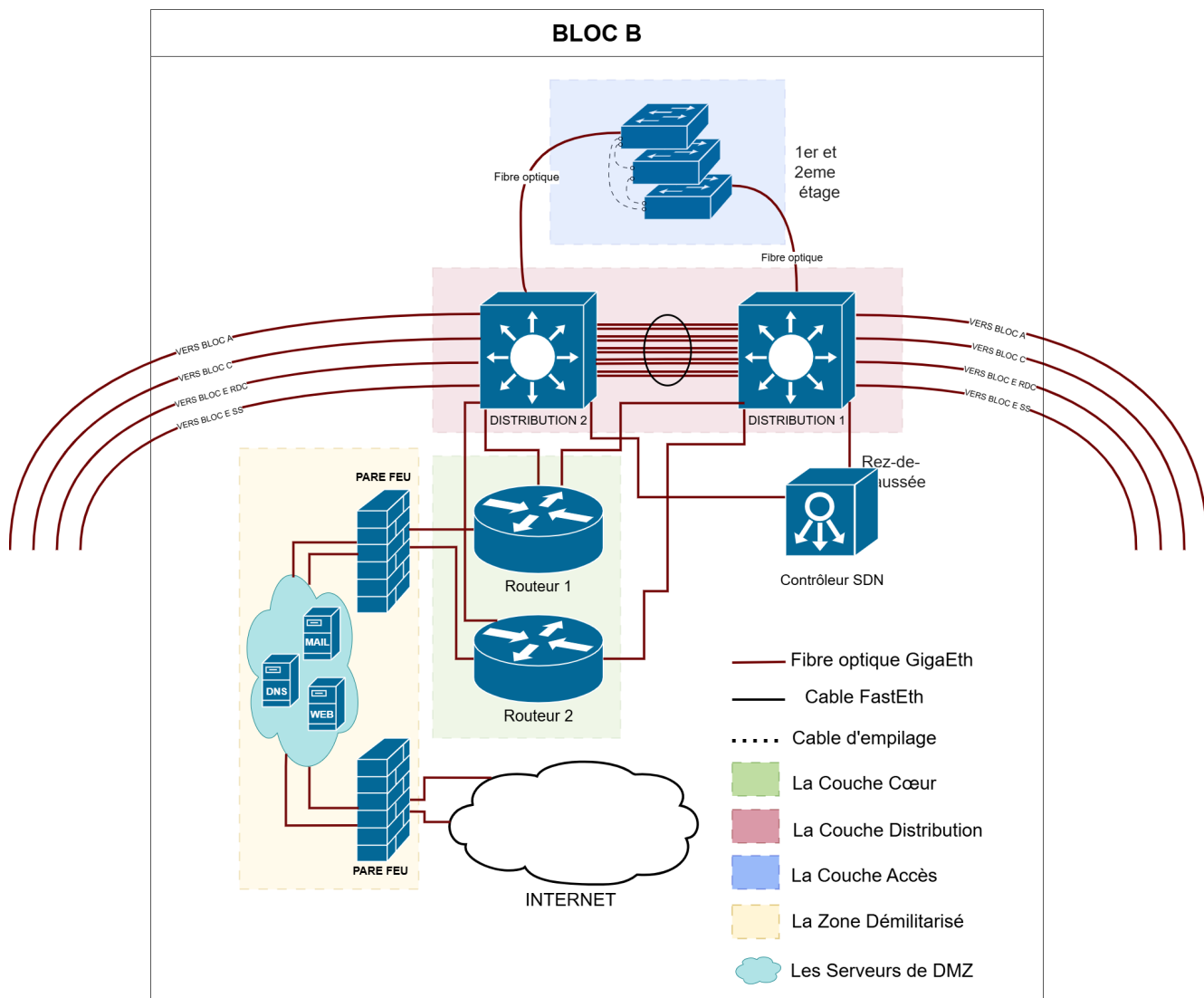


FIG. 2.7 : L'architecture proposée pour le bloc B

- **Bloc C et Bloc D** : pour le bloc C, besoin de quatre switches empilés et d'une liaison fibre vers B et le D qui a un seul étage, besoin de deux switches empilés. L'architecture est illustrée dans la figure 2.8.

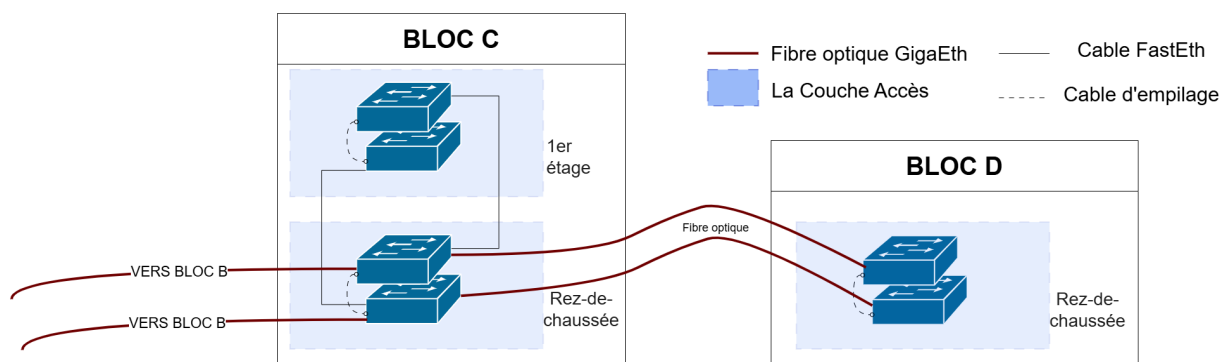


FIG. 2.8 : L'architecture proposée pour les blocs C et D

- **Bloc E** : la figure 2.9 représente le bloc avec les 14 switches (des switches empilés) deux pour sous-sol + deux pour RDC + dix pour cinq étages, liens directs vers B et distribution locale (switch de RDC).

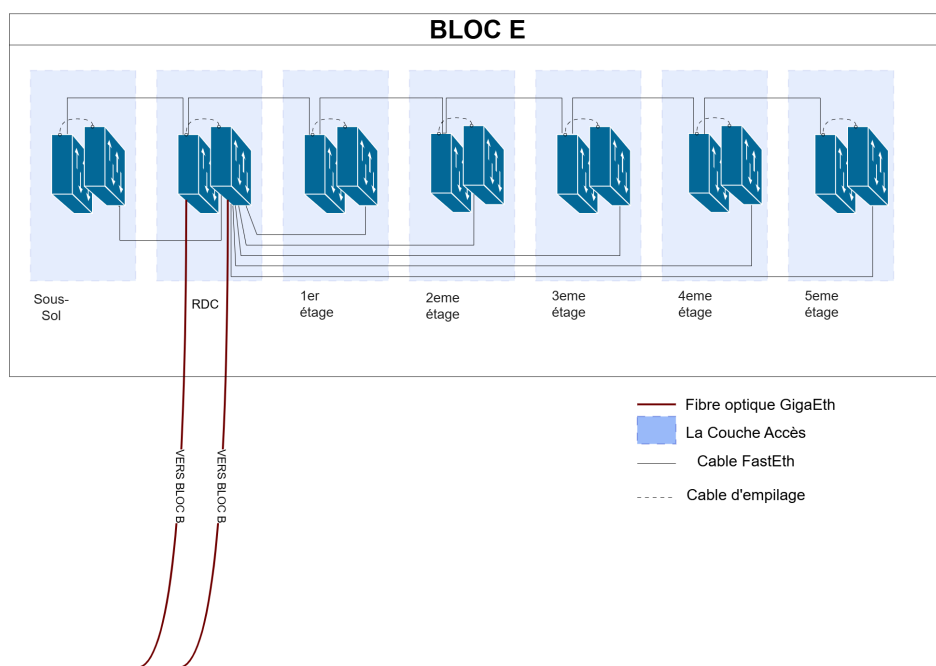


FIG. 2.9 : L'architecture proposée pour le bloc E

2.5.3 Architecture globale du système

Afin de répondre aux besoins de flexibilité, d'évolutivité et de supervision centralisée à un réseau d'envergure multisite, nous proposons une architecture SDN répartie sur l'ensemble des bâtiments du campus (blocs A à E) représentée sur la figure 2.10. Cette infrastructure repose sur un modèle *three-tier* optimisé pour la redondance, la sécurité et l'automatisation des flux.

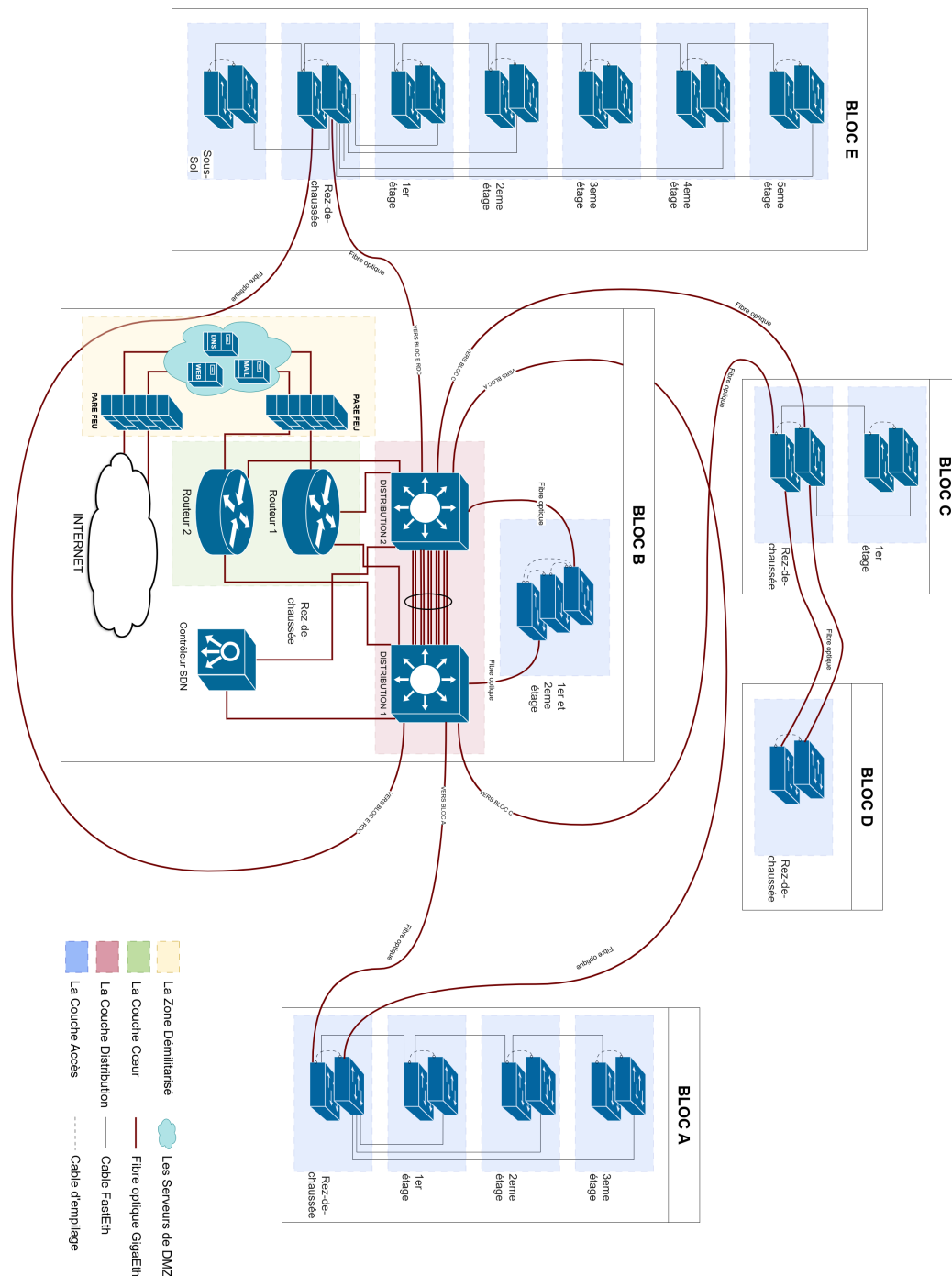


FIG. 2.10 : L'architecture globale de la solution SDN

Vue d'ensemble L'architecture s'appuie sur un modèle **three-tier** SDN étendu à plusieurs bâtiments (blocs A, B, C, D et E) :

- **Couche Accès** : switchs empilés par étage dans chaque bloc (bleu).
- **Couche Distribution** : deux châssis redondants au RDC du bloc B (rouge).
- **Couche Cœur** : routeurs/pare-feux au RDC du Bloc B (vert).
- **Contrôleur SDN** : relier avec les deux switches de distribution.

Description des composantes La table ci-dessous présente les principaux éléments de l'architecture réseau, leur rôle dans l'infrastructure, ainsi que leurs caractéristiques techniques essentielles.

TAB. 2.2 : Composantes de l'architecture globale

Élément	Rôle	Caractéristiques clés
Switchs empilés (Blocs A–E)	Couche Accès : connectivité locale par étage	Uplinks fibre GigaEth vers distribution, empilage pour redondance locale
Switchs Distribution 1 & 2 (Bloc B)	Agrégation et transit inter-blocs	Fibre 10/40 Gbps, redondance
Routeurs 1 & 2 (Bloc B)	Sortie vers internet	BGP, haute disponibilité
Contrôleur SDN	Contrôle centralisée du réseau	API Northbound REST, API Southbound Openflow
Double Pare-feux	Sécurité périmétrique et DMZ	IDS/IPS, Least privilege, haute disponibilité
Internet	Point d'entrée / sortie externe	Bande passante dédiée
Liens Inter-Blocs (fibre GigaEth)	Connectivité multi-bâtiments	1 Gbps full-duplex, faible latence

Points forts

- **Redondance multi-couches** : empilage local, double distribution.
- **Scalabilité horizontale** : ajout de nouveaux switchs leaf sans reconfigurer le cœur.
- **Sécurité renforcée** : DMZ, pare-feux redondants, .
- **Automatisation et supervision** : contrôleur SDN centralisé, API northbound, monitoring temps réel.

Limites et axes d'amélioration

- **Un seul point de défaillance du contrôleur SDN** : La présence d'un unique contrôleur dans une architecture SDN représente un risque majeur. En cas de panne, tout le réseau peut devenir inopérant, car le contrôle centralisé est interrompu.
- **Complexité de déploiement** : configuration initiale des politiques OpenFlow et des règles de pare-feu.

Perspectives d'évolution

- Évolution des uplinks de distribution vers du **40/100 Gbps** pour supporter la croissance.
- Renforcement du **contrôleur SDN** en cluster géo-réparti pour haute disponibilité et latence réduite.

2.6 Conclusion

Ce chapitre a montré comment notre proposition SDN, structurée en couches Accès, Distribution et Cœur, répond aux faiblesses de l'architecture traditionnelle : tolérance aux pannes, montée en charge transparente et segmentation dynamique via DMZ. La description bloc par bloc et la synthèse globale illustrent la cohérence d'ensemble et la capacité d'évolution de ce modèle. Forts de cette base solide, nous aborderons dans le chapitre suivant les aspects de mise en œuvre pratique, notamment le choix du contrôleur, la configuration OpenFlow et les outils d'orchestration.

Chapitre 3

Validation de l'architecture SDN

3.1 Introduction

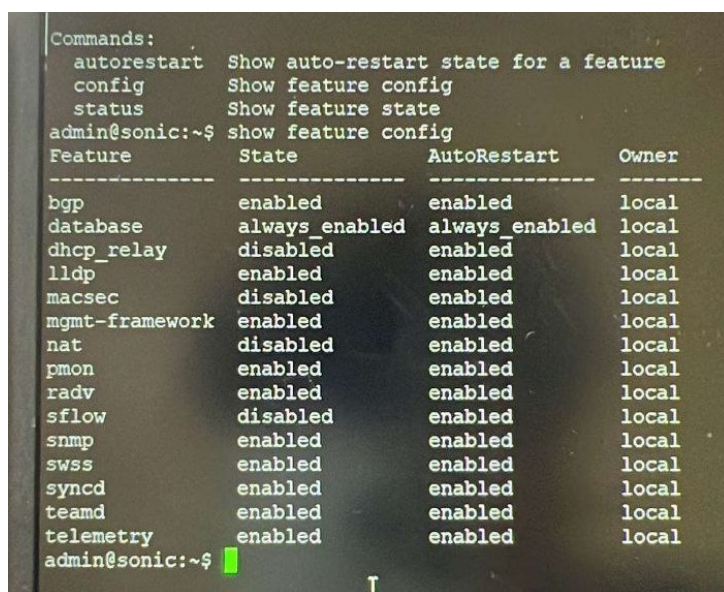
Après avoir analysé l'architecture réseau existante dans le chapitre précédent et mis en évidence ses nombreuses limites, notamment en termes de fiabilité, de sécurité et de flexibilité, il devient clair qu'une nouvelle approche s'impose. Cette nouvelle architecture doit répondre aux besoins actuels tout en respectant les contraintes fixées dans le cahier des charges, notamment en ce qui concerne la qualité de service et les coûts.

Dans ce chapitre, nous commencerons par examiner l'indisponibilité matérielle, en détaillant les contraintes qui ont empêché toute mise en œuvre physique. Nous verrons ensuite le passage vers l'émulation, justifiant ce choix face à l'inaccessibilité au matériel, puis la préparation de l'environnement d'émulation, incluant l'installation de Mininet, d'OpenDaylight, d'OpenFlow Manager et de Wireshark. Nous décrirons ensuite la réalisation de l'architecture à l'aide de MiniEdit, qui permet de créer graphiquement notre topologie SDN, avant de conclure par la phase de test et de validation, au cours de laquelle nous mesurerons la connectivité et vérifierons les règles de filtrage de flux.

3.2 Indisponibilité matérielle

Dans le cadre initial du projet, nous avons souhaité pouvoir déployer physiquement une architecture réseau SDN reposant sur le protocole OpenFlow pour assurer la liaison south-bound entre le contrôleur et les équipements réseau. L'équipement mis à notre disposition à cet effet était un commutateur Supermicro SONiC SSEG3748 [29], reconnu pour ses performances et sa compatibilité avec les environnements modernes.

Or, lors des essais, il est apparu que ces commutateurs ne supportent pas OpenFlow et ne disposent pas d'une fonctionnalité native SDN. En l'absence de matériel réellement dédié à la couche de contrôle SDN, la mise en œuvre physique envisagée s'est trouvée compromise. La figure 3.1 illustre une capture d'écran attestant de l'incompatibilité du matériel disponible avec le protocole OpenFlow.



```

Commands:
  autorestart  Show auto-restart state for a feature
  config       Show feature config
  status       Show feature state
admin@sonic:~$ show feature config
Feature      State      AutoRestart  Owner
-----
bgp          enabled    enabled       local
database     always_enabled always_enabled local
dhcp_relay   disabled   enabled       local
lldp         enabled    enabled       local
macsec       disabled   enabled       local
mgmt-framework enabled    enabled       local
nat          disabled   enabled       local
pmon         enabled    enabled       local
radv         enabled    enabled       local
sflow        disabled   enabled       local
snmp         enabled    enabled       local
swss         enabled    enabled       local
syncd        enabled    enabled       local
teamd        enabled    enabled       local
telemetry    enabled    enabled       local
admin@sonic:~$

```

FIG. 3.1 : Limitation du matériel disponible

Pour pallier cette contrainte et compte tenu du temps alloué à la réalisation de ce projet, nous avons choisi de basculer vers une solution d'émulation logicielle. Cette approche permet de reproduire fidèlement le comportement d'un réseau SDN (plans de contrôle et de données) dans un environnement réaliste, sans dépendre d'un commutateur physique.

3.3 Le passage vers l'émulation

Face aux contraintes matérielles énoncées, notamment l'incompatibilité des commutateurs Supermicro SONiC SSEG3748 avec le protocole OpenFlow, il a été nécessaire de trouver une alternative pour poursuivre le développement de l'architecture réseau basée sur le SDN. L'émulation s'est imposée comme une solution viable, permettant de simuler un environnement réseau réaliste sans dépendre du matériel physique.

L'émulation réseau consiste à recréer virtuellement un réseau complet, incluant les équipements, les protocoles et les conditions de trafic, afin de tester et valider des configurations dans un environnement contrôlé. Contrairement à la simulation, qui modélise le comportement du réseau, l'émulation exécute le logiciel réel sur des plateformes virtuelles, offrant ainsi une représentation plus fidèle du fonctionnement du réseau [30] [31].

Cette approche présente plusieurs avantages :

- **Flexibilité** : elle permet de tester différentes configurations et scénarios sans les contraintes liées au matériel physique.
- **Économie** : en éliminant le besoin d'investir dans du matériel coûteux pour les phases de test et de développement.
- **Réalisme** : en reproduisant fidèlement les conditions du réseau, y compris les latences, les pertes de paquets et les congestions, ce qui est essentiel pour évaluer la performance des applications et des services.

Ainsi, l'adoption de l'émulation dans ce projet permet non seulement de surmonter les limitations matérielles, mais aussi d'offrir une plateforme flexible et réaliste pour le développement.

3.4 Préparation de l'environnement d'émulation

Pour réaliser l'émulation de notre architecture SDN, quatre composants ont été retenus :

- **Mininet** : émulateur de topologies SDN pour tester des réseaux virtuels.
- **OpenDaylight (ODL)** : contrôleur SDN open source pour piloter les flux.
- **OpenFlow Manager (OFM)** : application cliente web pour déployer et superviser les règles OpenFlow.
- **Wireshark** : analyseur de paquets réseau pour capturer et inspecter le trafic OpenFlow.

3.4.1 Installation et configuration de Mininet

Mininet est un émulateur de réseau virtuel entièrement écrit en Python permettant de créer, sans difficulté notable, un réseau complet (hôtes, commutateurs, contrôleur) sur une seule machine, en exécutant du code réel (noyau Linux, Open vSwitch, applications). Il est largement utilisé pour le prototypage et les tests en SDN [32], [33].

Étapes d'installation

1. Mettre à jour la liste des paquets :

```
sudo apt-get update
```

2. Installer Mininet depuis les dépôts :

```
sudo apt-get install -y mininet
```

3. Installer toutes les dépendances nécessaires au bon fonctionnement de Mininet (Open vSwitch et les APIs en une seule commande) :

```
cd mininet  
sudo ./util/install.sh -a
```

3.4.2 Choix et installation du contrôleur OpenDaylight

Dans cette partie, nous présentons les critères de sélection du contrôleur ainsi que les étapes d'installation et de configuration jusqu'à sa mise en service.

Motivations du choix d'OpenDaylight

OpenDaylight est un contrôleur SDN open source, développé en Java et piloté par la Linux Foundation. Plusieurs critères ont motivé son adoption :

- **Modularité et extensibilité** : son architecture basée sur Apache Karaf permet d'installer uniquement les bundles (features) souhaités, et d'y ajouter facilement de nouveaux protocoles ou applications.
- **Écosystème riche** : support natif de nombreux protocoles (OpenFlow, NETCONF, PCEP), interfaces nord (RESTCONF, gRPC) et intégrations avec des outils tiers (SNMP, télémétrie, OpenFlow Manager).
- **Communauté et support industriel** : contributions régulières de Cisco, Huawei, IBM, Ericsson... assurant mises à jour fréquentes, correctifs de sécurité et documentation exhaustive.

- **Interface graphique (DLUX)** : un tableau de bord web intégré pour visualiser la topologie, piloter les flux et surveiller les performances, idéal en phase de prototypage.
- **Compatibilité éprouvée avec OFM** : malgré son ancienneté (juin 2015), la version *Lithium* a été choisie pour sa compatibilité totale avec OpenFlow Manager et sa grande stabilité, garantissant un environnement fiable pour la gestion avancée des règles OpenFlow.

Étapes d'installation

La version *Lithium* nécessite Java 8 (Oracle JDK ou OpenJDK). Les commandes ci-dessous fonctionnent sur Ubuntu 18.04 ou ultérieur :

1. Installer Java 8 :

```
sudo apt-get update
sudo apt-get install -y openjdk-8-jre-headless
```

2. Télécharger et extraire ODL Lithium :

```
wget https://nexus.opendaylight.org/content/repositories/public/org/
↳ opendaylight/integration/distribution-karaf/0.3.0-Lithium/
tar -xzf distribution-karaf-0.3.0.tar.gz
cd distribution-karaf-0.3.0
```

3. Démarrer Apache Karaf et activer les features essentielles :

```
./bin/karaf
feature :install odl-restconf
feature :install odl-l2switch-switch
feature :install odl-dlux-all
feature :install odl-mdsal-apidocser
```

4. Vérifier et accéder à GUI :

- Ouvrir `http://localhost:8181/index.html` (login/password : admin/admin).

Architecture d'OpenDaylight

Comme tous les contrôleurs SDN, OpenDaylight se compose de trois parties principales [34] :

- **Interface Sud (Southbound)** : elle permet la communication avec les nœuds réseaux (switches, appareils) via différents protocoles pris en charge par ODL, notamment OpenFlow, BGP-LS (Border Gateway Protocol Link-State), PCEP (Path Computation Element Protocol), NETCONF, etc. Ceci assure le control et le pilotage fin du plan de données.
- **Noyau du contrôleur (Controller Core)** : cœur logique basé sur MD-SAL (Model-Driven Service Abstraction Layer), il orchestre les services internes, gère le cycle de vie des modules OSGi/Karaf, stocke l'état réseau et garantit l'extensibilité et la haute disponibilité.
- **Interface Nord (Northbound)** : elle expose les API REST (RESTCONF) et autres interfaces (gRPC, WebSocket) à des applications externes ou à un plan applicatif supérieur, permettant d'effectuer des opérations HTTP de base (GET, POST, PUT, DELETE) pour la configuration, la supervision ou l'analyse du réseau.

3.4.3 Installation d'OpenFlow Manager

OpenFlow Manager (OFM) est une application cliente SDN pour OpenDaylight développée en Node.js. Il fournit une interface web dédiée au déploiement, à la supervision et à l'analyse des flux OpenFlow. Elle simplifie l'abstraction et la gestion centralisée des flux dans un réseau SDN. La figure ci-dessous montre la description de l'architecture des composants OFM [35].

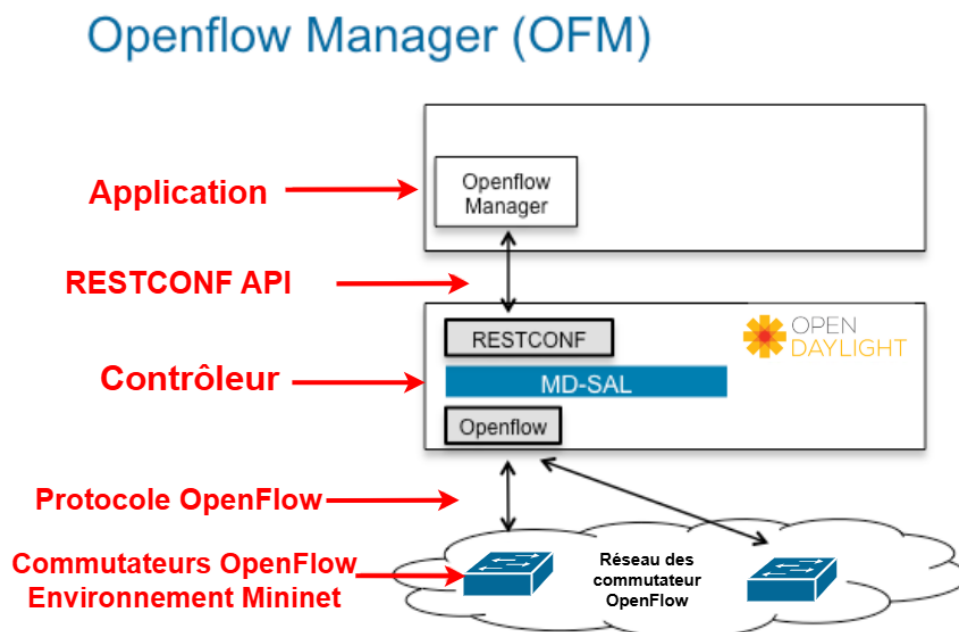


FIG. 3.2 : Description de l'architecture des composants OFM [35]

Étapes d'installation

Les étapes suivantes détaillent l'installation à partir du dépôt GitHub [36] :

1. Installer Node.js :

```
curl -sL https://deb.nodesource.com/setup_4.x | sudo -E bash -  
sudo apt-get install -y nodejs
```

2. Télécharger le code source :

```
git clone  
→ https://github.com/CiscoDevNet/OpenDaylight-Openflow-App.git
```

3. Configurer l'adresse du controlleur OpenDaylight :

```
cd OpenDaylight-Openflow-App  
sed -i 's/localhost/<ODL_ip>/g' ./ofm/src/common/config/env.module.js
```

4. Installer les dépendances et l'outil de build Grunt :

```
npm install  
npm install -g grunt-cli
```

5. Lancer l'application :

```
cd OpenDaylight-Openflow-App  
grunt
```

Puis accéder à l'interface web via un navigateur à l'adresse configurée, avec le port 9000.

ouvrir `http://localhost:9000/`

3.4.4 Installation et configuration de Wireshark

Wireshark est un analyseur de paquets réseau open source développé par la Wireshark Foundation. Il permet de capturer, décoder et afficher en temps réel le trafic passant sur une ou plusieurs interfaces réseau. Grâce à ses filtres d'affichage et de capture avancés, il offre une analyse de centaines de protocoles, facilite la mesure des délais de transmission, l'identification des goulets d'étranglement et le diagnostic des anomalies (pertes de paquets, retransmissions, etc.). La figure ci-dessous montre l'interface principale de Wireshark [37].

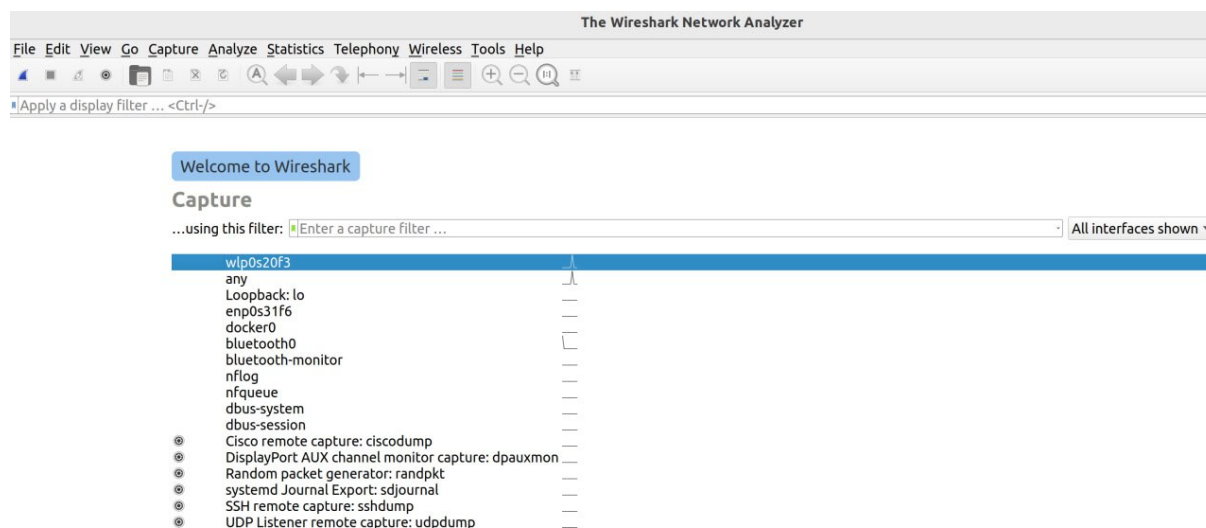


FIG. 3.3 : L'interface principale de Wireshark

Étapes d'installation

Les étapes suivantes détaillent l'installation à partir du site officiel [37].

1. Mettre à jour la liste des paquets :

```
sudo apt-get update
```

2. Installer Wireshark depuis les dépôts :

```
sudo apt-get install wireshark
```

3. Lancer le Wireshark :

```
sudo wireshark
```

3.5 Réalisation de l'architecture à l'aide de MiniEdit

MiniEdit est une interface graphique de Mininet, développée en Python et reposant sur Tkinter, qui simplifie la création de topologies SDN [38]. Cet outil permet à l'utilisateur de glisser-déposer des icônes d'hôtes, de commutateurs et de routeurs pour composer visuellement son réseau, puis de générer automatiquement le script Mininet correspondant pour lancer l'émulation. Cette approche intuitive facilite l'expérimentation et l'apprentissage du SDN sans recourir à l'écriture manuelle de scripts.

3.5.1 Présentation de l'outil MiniEdit

MiniEdit propose une interface graphique intuitive, décomposée en trois fonctionnalités majeures :

- **Palette d'éléments** : Un panneau latéral regroupe quatre types d'objets que l'on peut glisser-déposer sur le canevas : les hôtes (host), qui représentent des machines virtuelles émettrices ou réceptrices de trafic ; les commutateurs (switch), assurant la commutation SDN via OpenFlow ; les routeurs (router), jouant le rôle d'intermédiaires pour le routage inter-VLAN ou entre sous-réseaux ; et les liens (link), offrant des connexions bidirectionnelles paramétrables en bande passante et latence.
- **Configuration de chaque nœud** : En cliquant sur un élément du canevas, un panneau de propriétés s'ouvre et permet de définir l'adresse IP et le masque de sous-réseau de chaque nœud, de choisir le protocole de contrôle (par exemple OpenFlow v1.3) et d'ajuster des paramètres avancés tels que la bande passante, la latence ou le taux de perte de paquets.
- **Génération d'un script Python** : Une fois la topologie dessinée et paramétrée, MiniEdit exporte automatiquement un script Python conforme à l'API Mininet : il inclut les imports nécessaires, crée les hôtes, commutateurs et routeurs, configure les liens et lance le contrôleur SDN choisi, puis exécute les commandes `net.start()` et `CLI(net)` pour démarrer le réseau et offrir une interface de ligne de commande.

3.5.2 Topologie déployée

La figure 3.4 illustre l'architecture retenue et créée grâce à MiniEdit :

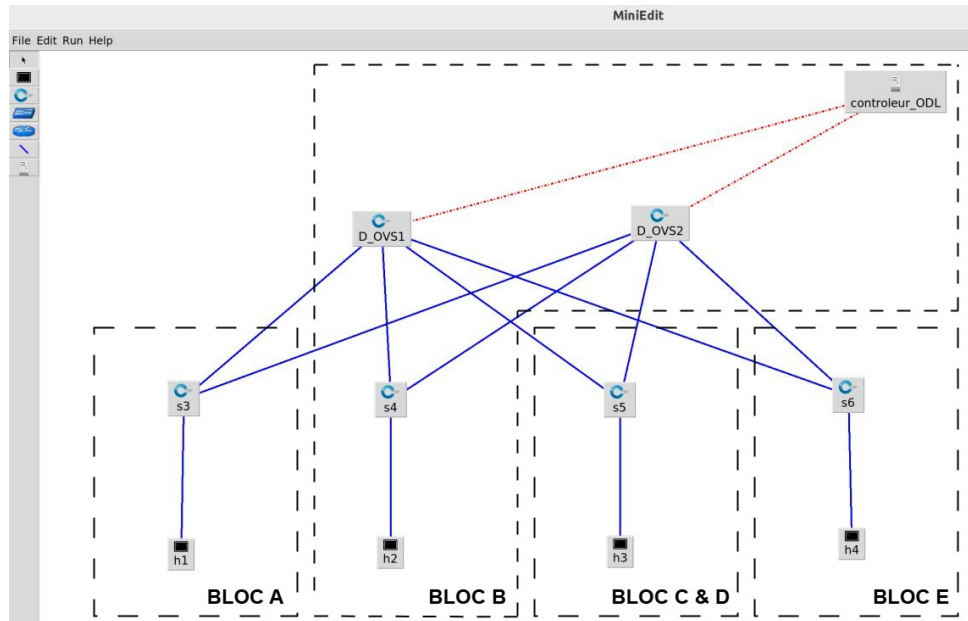


FIG. 3.4 : L'architecture proposée sur MiniEdit

Cette architecture se compose de deux couches principales :

- Couche distribution : 2 commutateurs Open vSwitch reliés au contrôleur ODL,
- Couche accès : 4 commutateurs Open vSwitch (chaque commutateur représente un bloc dans la vraie architecture).

La configuration détaillée de chaque nœud est présentée dans le tableau suivant :

TAB. 3.1 : Configuration IP de chaque nœud

Hôte	VLAN	IP locale (int-vlan)	Passerelle
D_OVS1	10	10.0.1.100/24	
	20	10.0.2.100/24	
	30	10.0.3.100/24	
	40	10.0.4.100/24	
D_OVS2	10	10.0.1.100/24	
	20	10.0.2.100/24	
	30	10.0.3.100/24	
	40	10.0.4.100/24	
PC1	10	10.0.1.1/24	10.0.1.254
PC2	20	10.0.2.2/24	10.0.2.254
PC3	30	10.0.3.3/24	10.0.3.254
PC4	40	10.0.4.4/24	10.0.4.254
Contrôleur ODL		127.0.0.1 :6633	

Pour une meilleure compréhension du tableau, les éléments de chaque colonne sont expliqués ci-dessous.

- **Hôte** : identifie le nom de l'équipement ou de la machine concernée (par exemple : D_OVS1, D_OVS2, PC1).
- **VLAN** : numéro du VLAN (tag 802.1Q) auquel l'interface L3 appartient. Chaque VLAN correspond à un domaine de broadcast isolé.
- **IP locale (int-vlan)** : adresse IP de l'interface interne dédiée au VLAN.
 - Sur les D_OVS : sert de source pour les annoncesVRRP et pour l'accès administratif.
 - Sur les PC : correspond à leur adresse dans le réseau du VLAN.
 - Sur le contrôleur : sert à pointer vers le contrôleur dans notre localhost dans le port openflow 6633 dont la version est 1.3
- **Passerelle** adresse **IP virtuelle** VIP (flottante viaVRRP) que les machines du VLAN utilisent comme gateway par défaut.

Cette adresse IP virtuelle (VIP) est une ressource logique partagée qui permet de présenter une passerelle unique et stable aux clients d'un VLAN, même en cas de défaillance du nœud porteur.

Le protocole VRRP définit la coordination entre plusieurs *nœuds* pour désigner lequel prend la VIP et lesquels restent en veille, assurant ainsi une continuité de service.

Le Open vSwitch (D_OVS) ne dispose pas d'implémentationVRRP native ; nous utilisons donc le démon **keepalived** pour apporter cette fonctionnalité.

Sur chaque nœud D_OVS 1 et 2, on procède comme suit :

1. Création de l'interface interne **int-vlanXX** et affectation de son IP locale (p. ex. 10.0.1.2/24 sur D_OVS1).
2. Installation et configuration de **keepalived** :
 - **D_OVS1** est défini en **MASTER** (priorité 150) et prend la VIP (p. ex. 10.0.1.254/24).
 - **D_OVS2** est défini en **BACKUP** (priorité 100) et reste en veille jusqu'à l'absence des annonces du maître.

3. Démarrage de `keepalived` sur les deux nœuds

Voici le code configuration 3.1 de `keepalived` dans `/etc/keepalived/keepalived.conf` à déployer sur nos deux nœuds D_OVS :

D_OVS1 (MASTER), D_OVS2 (BACKUP)

Code 3.1 : Le fichier `keepalived.conf`

```
global_defs {
    router_id D_OVS1
}

vrrp_instance VI10 {
    state MASTER
    interface int-vlan10
    virtual_router_id 10
    priority 150
    advert_int 1

    authentication {
        auth_type PASS
        auth_pass vrrpSecret
    }

    virtual_ipaddress {
        10.0.1.254/24 dev int-vlan10
    }
}

global_defs {
    router_id D_OVS2
}

vrrp_instance VI10 {
    state BACKUP
    interface int-vlan10
    virtual_router_id 10
    priority 100
    advert_int 1

    authentication {
        auth_type PASS
        auth_pass vrrpSecret
    }

    virtual_ipaddress {
        10.0.1.254/24 dev int-vlan10
    }
}
```

La même procédure est dupliquée pour chaque VLAN (20, 30, 40) avec leurs VIP respectives (10.0.2.254, 10.0.3.254, 10.0.4.254). Le résultat est une passerelle par défaut à haute disponibilité, entièrement transparente pour les clients du réseau.

Mise en route du contrôleur SDN

Avant de lancer la phase de tests et de validation, nous devons initialiser le contrôleur SDN. Nous déployons OpenDaylight (ODL) en `localhost` (voir la figure 3.5), configuré pour écouter sur le port TCP 6633 et supportant le protocole OpenFlow 1.3. Cette étape garantit que le plan de contrôle est opérationnel et prêt à interagir avec nos nœuds OVS selon les spécifications de l'architecture.

```
abd_essalam@abds1m:~/Downloads/distribution-karaf-0.3.0-Lithium$ ./bin/karaf
Java HotSpot(TM) 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.
```

FIG. 3.5 : Lancement de l'ODL

Ensuite, dans un navigateur, saisissez l'URL suivante :

```
http ://127.0.0.1 :8181/index.html
```

La fenêtre illustrée par la figure suivante s'affiche alors :

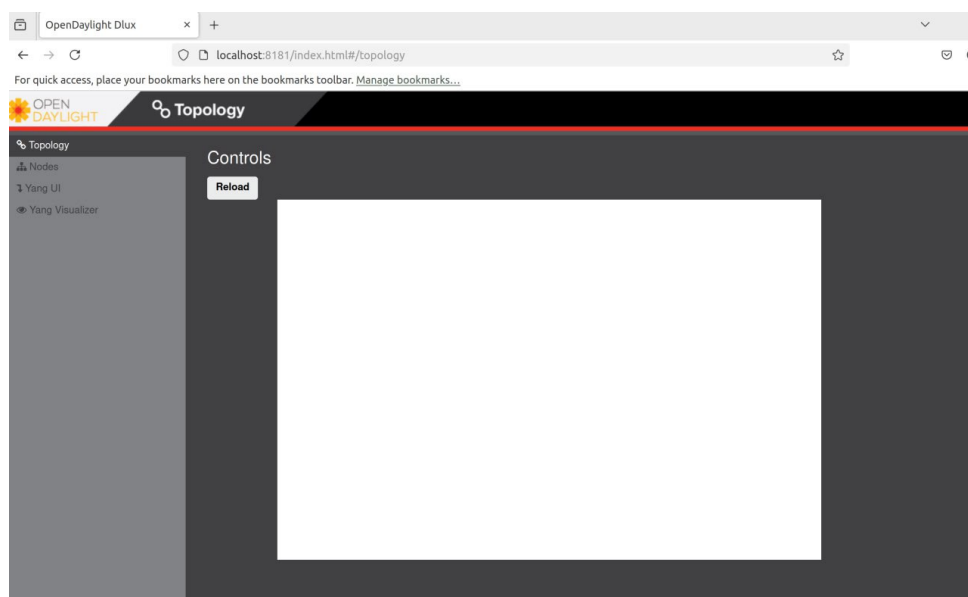


FIG. 3.6 : L'interface graphique vierge d'ODL

3.6 Test et validation

Après avoir achevé la configuration détaillée de notre environnement d'émulation et confirmé son bon fonctionnement, nous considérons que celui-ci est désormais opérationnel et conforme aux spécifications initiales. Forts de cette base stabilisée, nous allons à présent engager la phase de tests et de validation, afin d'évaluer notre travail.

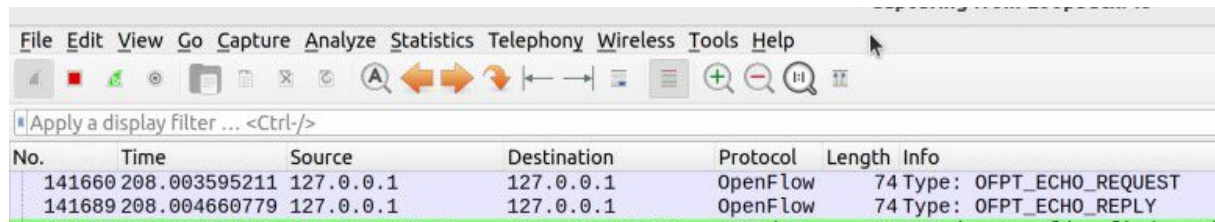
3.6.1 Test de connectivité

Pour débiter la phase de validation, il est essentiel de s'assurer que l'ensemble des composants de l'infrastructure communiquent correctement. Dans un premier temps, nous vérifions que le contrôleur OpenDaylight, déployé en local sur le port 6633 et supportant OpenFlow 1.3, détecte bien notre topologie. Une fois cette connexion établie, nous nous assurons que les hôtes virtuels peuvent échanger des paquets entre eux.

La figure 3.7 représentant une capture dans Wireshark montre cet échange sous la forme de messages :

OFPT_ECHO_REQUEST (contrôleur → commutateur)

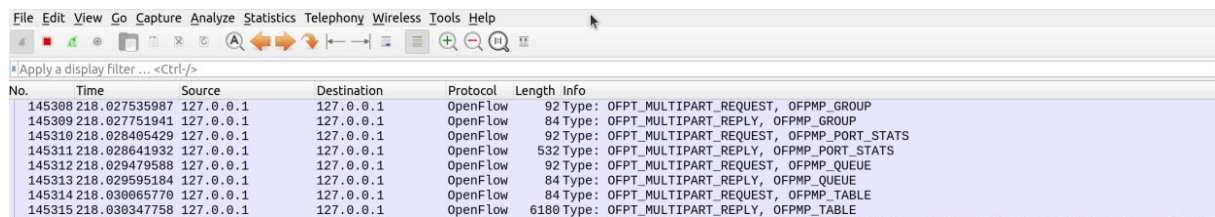
OFPT_ECHO_REPLY (commutateur → contrôleur), qui sert à confirmer que la session TCP/OpenFlow est vivante.



No.	Time	Source	Destination	Protocol	Length	Info
141660	208.003595211	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REQUEST
141689	208.004660779	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REPLY

FIG. 3.7 : Connexion entre le plan de contrôle et le plan de données

Juste après le handshake, le contrôleur interroge la configuration des switches et leurs compteurs à l'aide de requêtes multipart. La figure 3.8 illustre ces échanges :



No.	Time	Source	Destination	Protocol	Length	Info
145308	218.027535987	127.0.0.1	127.0.0.1	OpenFlow	92	Type: OFPT_MULTIPART_REQUEST, OFPMP_GROUP
145309	218.027751941	127.0.0.1	127.0.0.1	OpenFlow	84	Type: OFPT_MULTIPART_REPLY, OFPMP_GROUP
145310	218.028405429	127.0.0.1	127.0.0.1	OpenFlow	92	Type: OFPT_MULTIPART_REQUEST, OFPMP_PORT_STATS
145311	218.028641932	127.0.0.1	127.0.0.1	OpenFlow	532	Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_STATS
145312	218.029479508	127.0.0.1	127.0.0.1	OpenFlow	92	Type: OFPT_MULTIPART_REQUEST, OFPMP_QUEUE
145313	218.029595184	127.0.0.1	127.0.0.1	OpenFlow	84	Type: OFPT_MULTIPART_REPLY, OFPMP_QUEUE
145314	218.030065770	127.0.0.1	127.0.0.1	OpenFlow	84	Type: OFPT_MULTIPART_REQUEST, OFPMP_TABLE
145315	218.030347758	127.0.0.1	127.0.0.1	OpenFlow	6180	Type: OFPT_MULTIPART_REPLY, OFPMP_TABLE

FIG. 3.8 : Négociation entre le contrôleur et les OvS

Ces MULTIPART_REQUEST pour OFPMP_GROUP, OFPMP_PORT_STATS, OFPMP_QUEUE et OFPMP_TABLE, suivies des MULTIPART_REPLY, confirment que le contrôleur connaît la topologie, les capacités et les états des ports.

Puis, dans l'interface Web d'OpenDaylight (ODL), un simple clic sur le bouton « Reload » initie la session TCP OpenFlow 1.3 avec tous les commutateurs OVS. Dès que la communication est établie, comme l'indique la figure 3.9, le contrôleur découvre automatiquement la topologie déployée via Mininet et affiche l'ensemble des nœuds et des liens, confirmant ainsi l'interopérabilité complète de l'architecture.

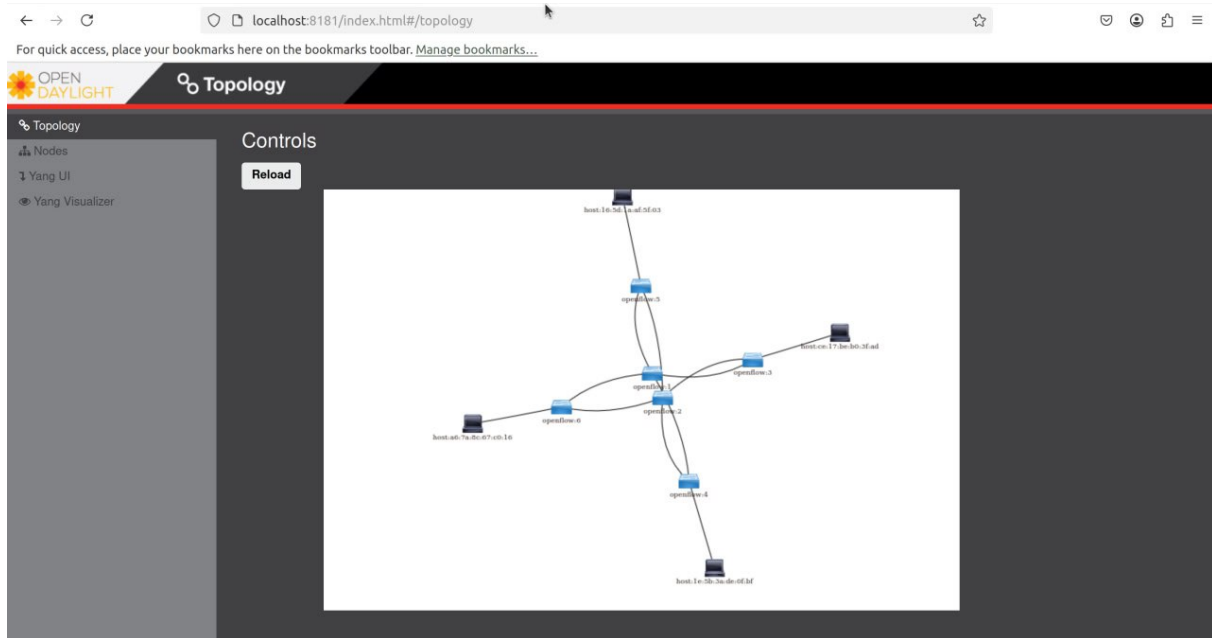


FIG. 3.9 : Vue d'architecture de la part de ODL

Dans l'interface CLI de Mininet, la commande suivante permet de tester simultanément la connectivité de tous les liens :

```
mininet> pingall
```

Un résultat affichant un taux de réussite de 100 % confirme la bonne interconnexion de tous les éléments de l'émulation.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet> h1 ping h4
PING 10.0.4.4 (10.0.4.4) 56(84) bytes of data.
64 bytes from 10.0.4.4: icmp_seq=1 ttl=64 time=0.068 ms
64 bytes from 10.0.4.4: icmp_seq=2 ttl=64 time=0.109 ms
64 bytes from 10.0.4.4: icmp_seq=3 ttl=64 time=0.093 ms
64 bytes from 10.0.4.4: icmp_seq=4 ttl=64 time=0.104 ms
^C
--- 10.0.4.4 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3083ms
```

FIG. 3.10 : La validation du test de connectivité

L'affichage indiqué par la figure 3.10, prouve qu'aucun des 12 paquets de test n'a été

perdu : chaque hôte (h1 Bloc A à h4 Bloc E) a bien atteint tous les autres. Cette couverture en « full mesh » démontre que :

- Les liaisons physiques (virtuelles) sont opérationnelles,
- Le routage inter-VLAN (via nos VIP VRRP) fonctionne sans faille.

Donc, les résultats positifs confirment le succès de notre premier test.

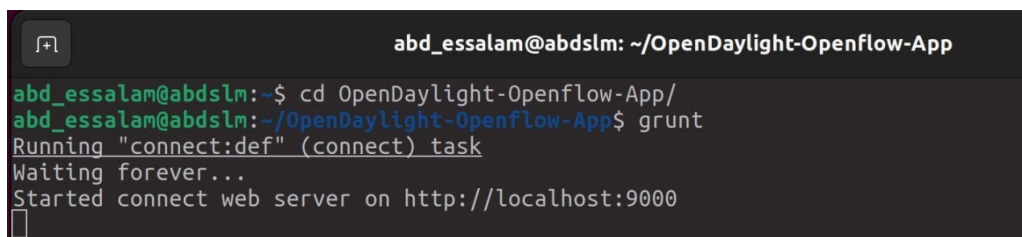
3.6.2 Test de sécurité : filtrage des flux interdits

Dans cette sous section, nous évaluons la capacité du contrôleur et des commutateurs à bloquer de manière sélective les flux non autorisés, en mettant en œuvre un filtrage basé sur l'adresse MAC source. Nous présentons d'abord la mise en place de l'interface de pilotage (OpenFlow Manager), puis la visualisation des règles existantes, et enfin la création d'une règle de blocage ciblée.

Lancement d'OpenFlow Manager et détection de l'architecture

Afin de piloter manuellement la diffusion et le filtrage des flux OpenFlow, nous utilisons OFM. Pour cela, il suffit de :

1. Démarrer l'OpenFlow Manager (voir la figure 3.11).



```
abd_essalam@abds1m: ~/OpenDaylight-Openflow-App
abd_essalam@abds1m:~$ cd OpenDaylight-Openflow-App/
abd_essalam@abds1m:~/OpenDaylight-Openflow-App$ grunt
Running "connect:default" (connect) task
Waiting forever...
Started connect web server on http://localhost:9000
```

FIG. 3.11 : Le lancement d'OpenFlow Manager

2. Ouvrir un navigateur Web à l'adresse <http://127.0.0.1:9000>.

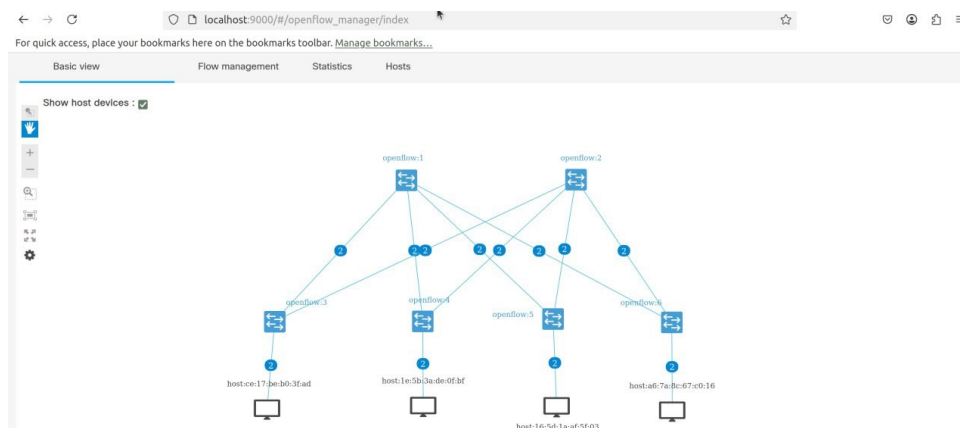


FIG. 3.12 : La détection de l'architecture par l'OpenFlow Manager

Comme montré à la figure 3.12, OFM détecte automatiquement la topologie déployée via Mininet et affiche les commutateurs ainsi que les liens et les hôtes.

Visualisation des flux existants

Dans l'onglet **Flow Management** > **Flows** (voir figure 3.13), on retrouve l'ensemble des règles OpenFlow que le contrôleur a installées pour garantir la connectivité de base entre les hôtes.

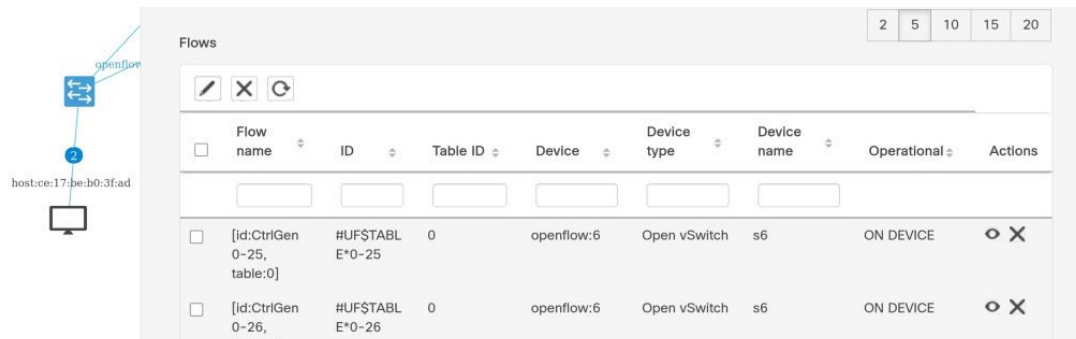


FIG. 3.13 : La visualisation des flux existants

Création d'une entrée de flux de filtrage

Nous illustrons ici un cas réel : bloquer tout trafic émanant de l'adresse MAC malveillante a6 :7a :8c :67 :c0 :16. La procédure est la suivante :

1. Cliquer sur l'icône « crayon » (Créer une entrée de flux) dans l'angle supérieur droit.
2. Dans le champ **Device**, sélectionner le commutateur OVS cible (s3 Bloc A).
3. Définir les propriétés générales :
 - **Table** : numéro de la table de flux (0).
 - **ID** : identifiant unique de la règle (par exemple, 777).
 - **Priority** : niveau de priorité (plus élevé (2000) que les règles existantes (10) pour garantir le blocage).
 - **Cookie** : étiquette de suivi, utilisée pour identifier et gérer ultérieurement cette règle (par exemple 0x77).
4. Dans la section **Match**, activer l'option **Source MAC** et saisir l'adresse a6 :7a :8c :67 :c0 :16 (voir la figure 3.14).

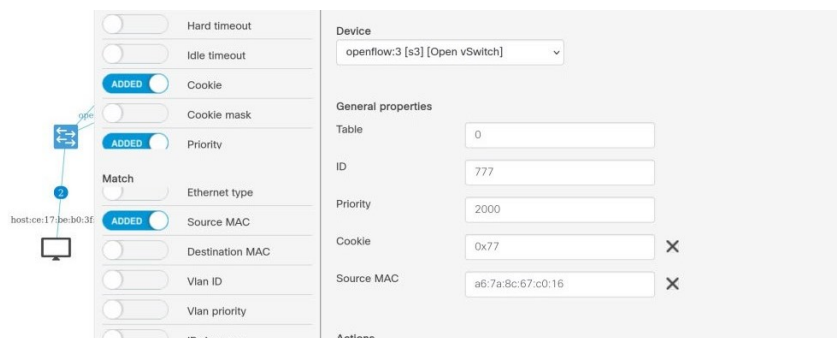


FIG. 3.14 : La création manuelle d'une entrée de flux dans la section (Match)

5. Dans la section **Actions**, sélectionner **Drop** (voir la figure 3.15).

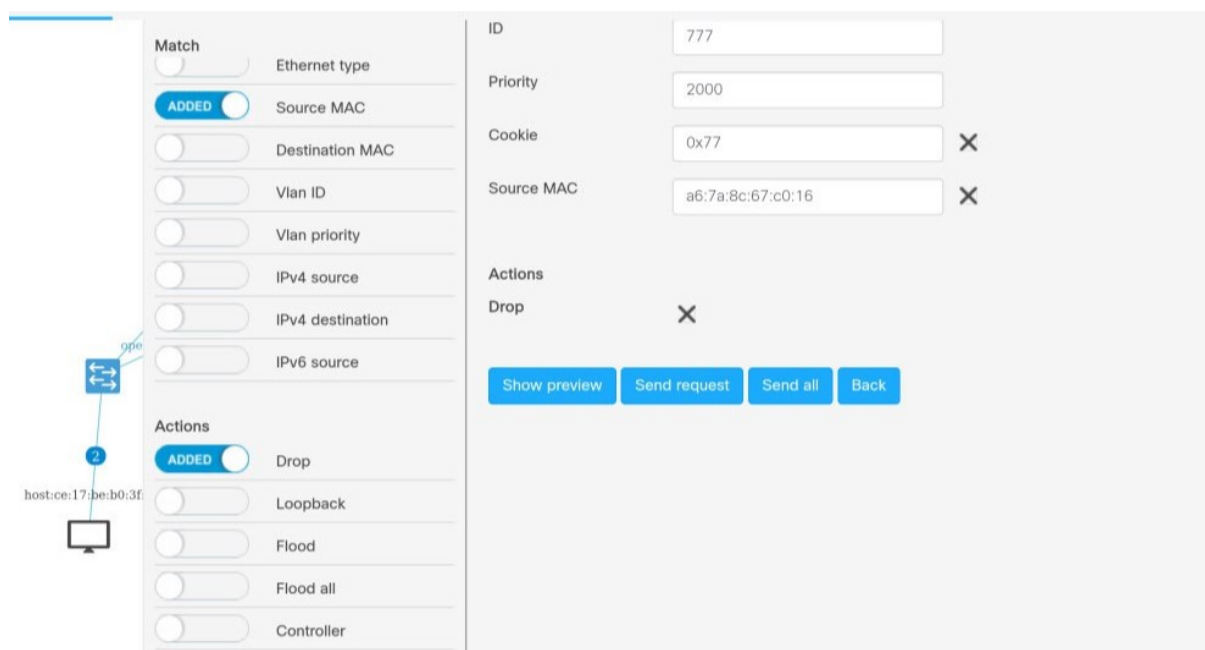


FIG. 3.15 : La création manuelle d'une entrée de flux dans la section (Action)

Les figures 3.14 et 3.15 présentent l'interface de création du flux de filtrage et illustrent comment OFM permet de définir de manière précise le blocage de trafic non autorisé.

Prévisualisation et envoi de la règle de filtrage

Une fois la configuration de cet entrée est complétée dans l'interface graphique d'OpenFlow Manager (OFM), il est possible d'en afficher la représentation exacte au format JSON en cliquant sur le bouton **Show preview** comme indique dans la figure 3.16.

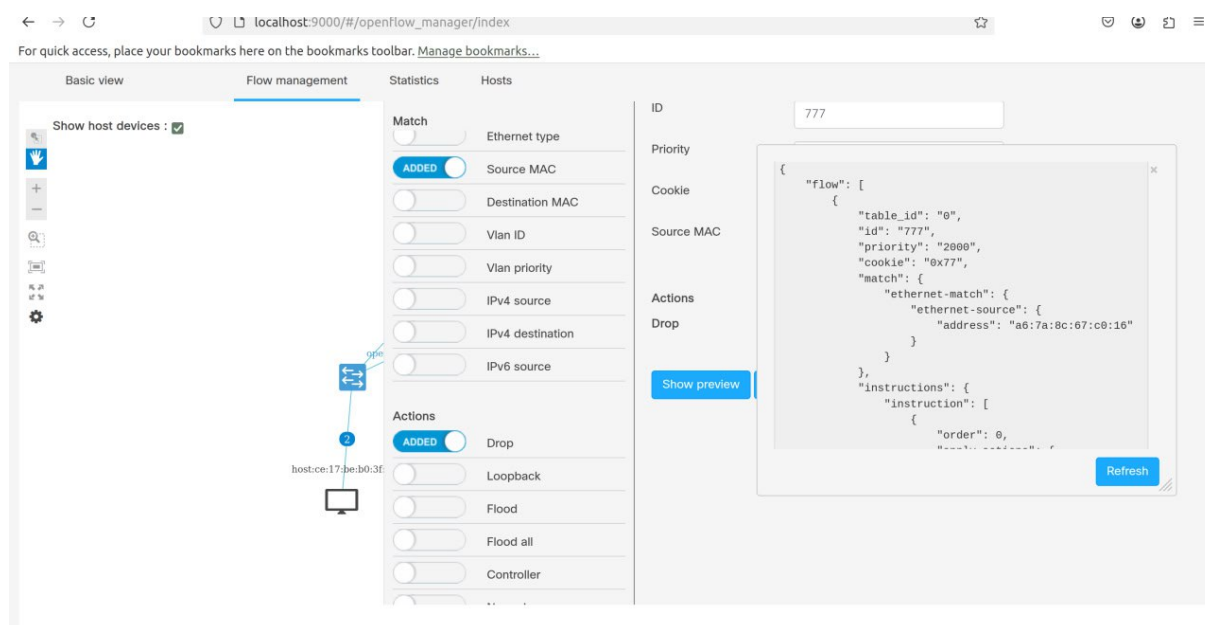


FIG. 3.16 : Le fichier JSON créé par l'OpenFlow Manager

Cette fonctionnalité permet à l'utilisateur de vérifier la structure et les paramètres de son propre entrée avant son envoi effectif. Un exemple typique du script généré est présenté ci-dessous :

Code 3.2 : Le fichier JSON généré par OpenFlow Manager

```
{
  "flow" : [
    {
      "table_id" : "0",
      "id" : "777",
      "priority" : "2000",
      "cookie" : "0x77",
      "match" : {
        "ethernet-match" : {
          "ethernet-source" : {
            "address" : "a6 :7a :8c :67 :c0 :16"
          }
        }
      },
      "instructions" : {
        "instruction" : [
          {
            "order" : 0,
            "apply-actions" : {
              "action" : [
                {
                  "order" : 0,
                  "drop-action" : {}
                }
              ]
            }
          }
        ]
      }
    }
  ]
}
```

Ce script JSON contient l'ensemble des propriétés de la règle de filtrage : l'identifiant de entrée de flux (id), sa priorité, le cookie, la condition de correspondance (ici une adresse MAC source spécifique) ainsi que l'action à appliquer (dans ce cas, un **drop** explicite du paquet).

Lorsque l'utilisateur clique sur **Send request**, OFM affiche le succès de cette entrée de flux. (voir la figure 3.17)

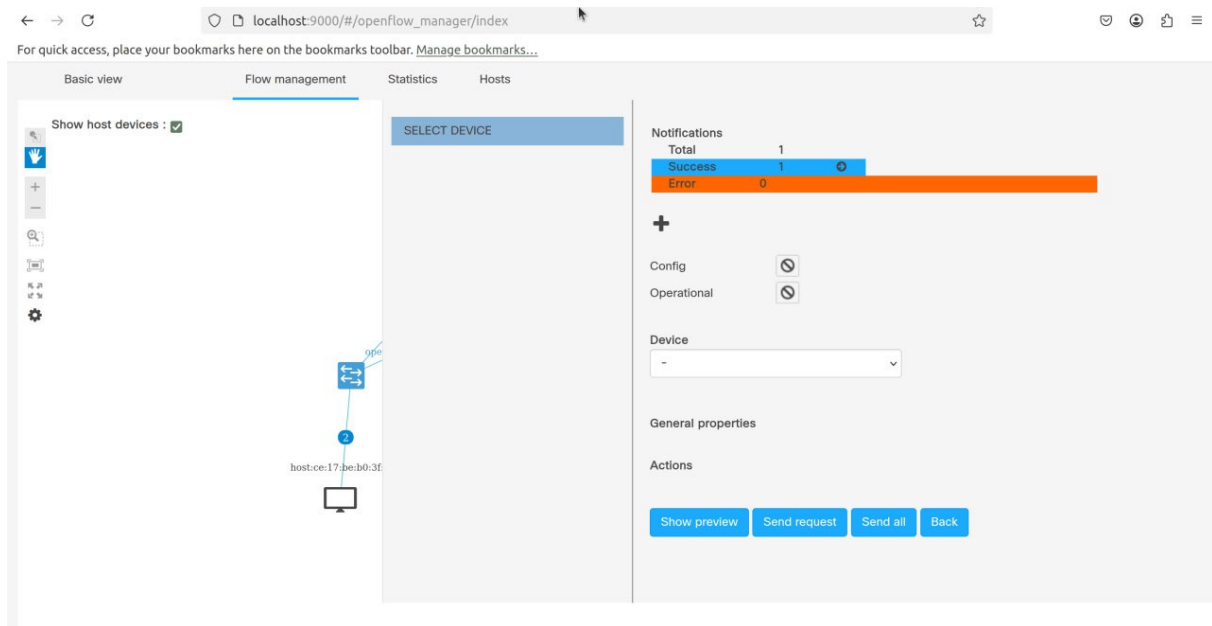


FIG. 3.17 : La confirmation de la création de l'entrée de flux

Ce dernier est transmis via une requête REST à l'API RESTCONF exposée par le contrôleur SDN OpenDaylight, la figure 3.18 montre cette transmission.

101477	152.969504339	192.168.100.68	192.168.100.68	HTTP/JS...	928 PUT /restconf/config/.opendaylight-inventory:nodes/node/openflow%3A3/table/0/flow/777 HTTP/1.1 200 OK, JavaScript Object Notation (application/json)
154082	228.716840172	192.168.100.68	192.168.100.68	HTTP/JS...	71 HTTP/1.1 200 OK, JavaScript Object Notation (application/json)
154072	228.695884189	192.168.100.68	192.168.100.68	HTTP/JS...	71 HTTP/1.1 200 OK, JavaScript Object Notation (application/json)
145662	217.763605689	192.168.100.68	192.168.100.68	HTTP/JS...	73 HTTP/1.1 200 OK, JavaScript Object Notation (application/json)
145652	217.675963280	192.168.100.68	192.168.100.68	HTTP/JS...	71 HTTP/1.1 200 OK, JavaScript Object Notation (application/json)

FIG. 3.18 : Le bon fonctionnement de l'API REST

Ensuite, ODL interprète la structure de l'entrée de flux et le convertit en messages OpenFlow, qui sont envoyés vers le OVS (s3 Bloc A) concernés, cela est affiché dans Wireshark avec `OFPT_FLOW_MOD` comme le montre la figure 3.19

- `OFPT_FLOW_MOD` : ce message permet d'ajouter, de modifier une entrée dans la table de flux du commutateur concerné. Il contient les critères de correspondance (match), la priorité, le cookie et les instructions (ici l'action `drop`) à appliquer aux paquets correspondants.

146318	230.474889897	127.0.0.1	127.0.0.1	OpenFlow	443 Type: OFPT_FLOW_MOD
146319	230.474891672	127.0.0.1	127.0.0.1	OpenFlow	443 Type: OFPT_PACKET_OUT
146320	230.474892371	127.0.0.1	127.0.0.1	OpenFlow	568 Type: OFPT_PACKET_OUT

FIG. 3.19 : L'ajout de notre entrée de flux dans le S3 par Openflow

À l'issue de ce processus, la figure 3.20 illustre que le flux de blocage est activement installé dans la table du commutateur S3, ce qui permet d'interdire immédiatement toute communication émise depuis l'adresse MAC spécifiée.

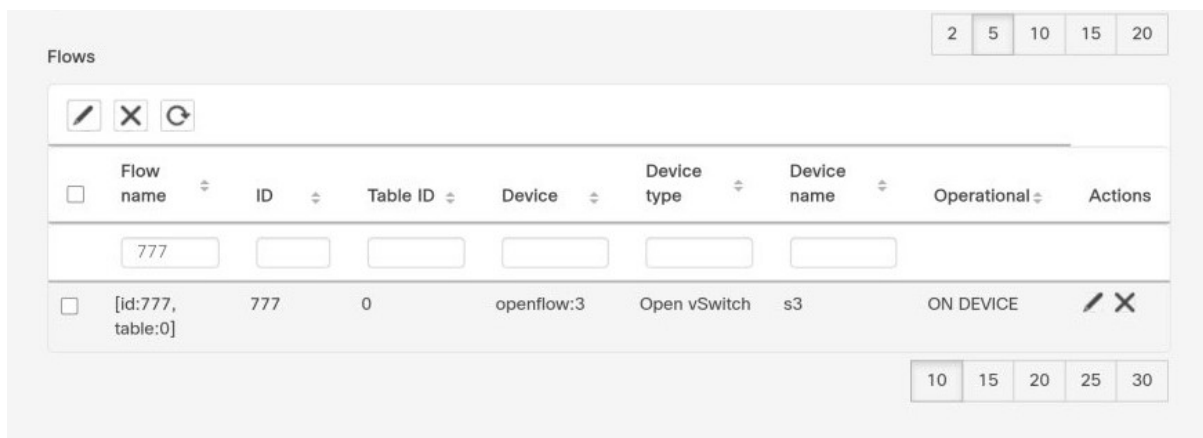


FIG. 3.20 : L'installation du flux de blocage dans le S3 (switch du Bloc A)

Pour confirmer que la règle de filtrage a bien été propagée jusqu'au switch OVS concerné, nous exécutons la commande suivante sur l'hôte :

```
sudo ovs-ofctl -O OpenFlow13 dump-flows s3
```

La figure 3.21 affiche l'exécution de la commande :

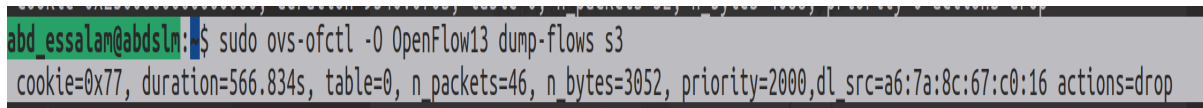


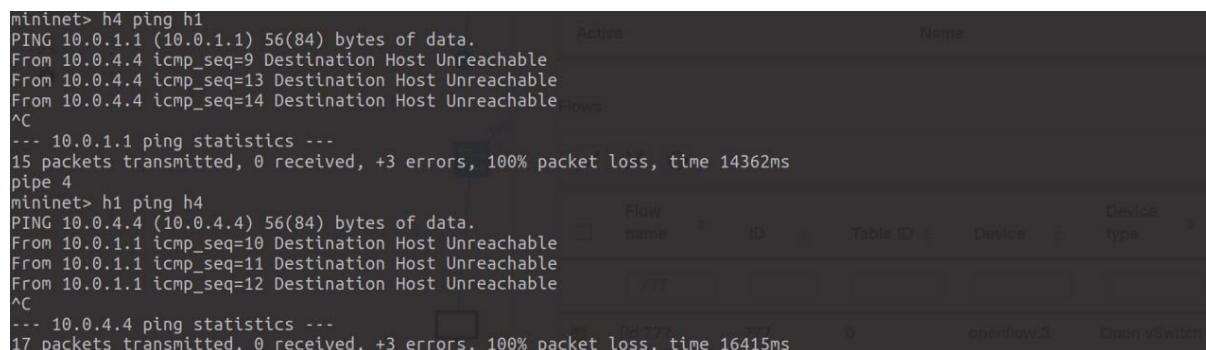
FIG. 3.21 : L'entrée de flux envoyé par OFM vers S3

Cette ligne montre que l'entrée injectée par OpenFlow Manager via l'API RESTCONF est effectivement présente dans la table du switch s3. Les champs `cookie=0x77`, `priority=2000` et `dl_src=a6:7a:8c:67:c0:16` confirment qu'il s'agit bien de la règle de blocage précédemment configurée, et l'action `drop` indique qu'elle est prête à interdire tout trafic provenant de cette adresse MAC.

Cette approche démontre la capacité du contrôleur, à travers OFM, à intégrer dynamiquement des politiques de sécurité sans reconfiguration manuelle au niveau des équipements physiques.

Validation du filtrage en situation réelle

Après avoir injecté la règle de blocage sur le switch **s3** (celui auquel est raccordé le commutateur du bloc A), nous lançons un test de ping de **h4 Bloc E** (celui auquel son adresse MAC est bloqué) vers **h1 Bloc A** afin de vérifier que tout trafic issu de l'adresse MAC de **h4 Bloc E** est effectivement interrompu, la figure 3.22 illustre :



```
mininet> h4 ping h1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
From 10.0.4.4 icmp_seq=9 Destination Host Unreachable
From 10.0.4.4 icmp_seq=13 Destination Host Unreachable
From 10.0.4.4 icmp_seq=14 Destination Host Unreachable
^C
--- 10.0.1.1 ping statistics ---
15 packets transmitted, 0 received, +3 errors, 100% packet loss, time 14362ms
pipe 4
mininet> h1 ping h4
PING 10.0.4.4 (10.0.4.4) 56(84) bytes of data.
From 10.0.1.1 icmp_seq=10 Destination Host Unreachable
From 10.0.1.1 icmp_seq=11 Destination Host Unreachable
From 10.0.1.1 icmp_seq=12 Destination Host Unreachable
^C
--- 10.0.4.4 ping statistics ---
17 packets transmitted, 0 received, +3 errors, 100% packet loss, time 16415ms
```

FIG. 3.22 : Première validation du test (filtrage des flux interdits)

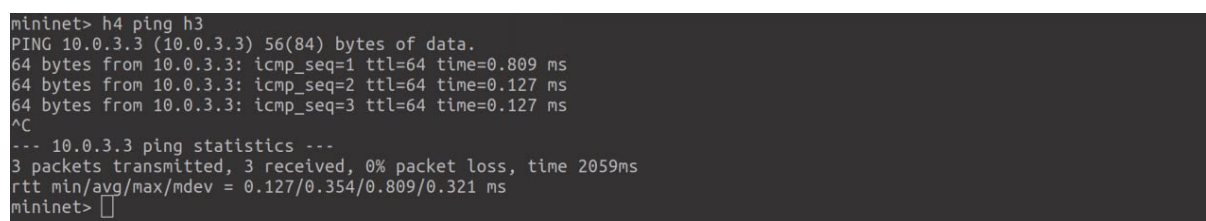
Le message **Destination Host Unreachable** confirme que le flux de blocage défini sur **s3** du Bloc A empêche toute communication de **h4 Bloc E** vers **h1 Bloc A**.

Le filtrage agit également en sens inverse (voir figure 3.22) : les requêtes ICMP émanant de **h1 Bloc A** voient la même règle de blocage s'appliquer, car les paquets de réponse de **h4 Bloc E** tentent de revenir via **s3 Bloc A** et sont eux aussi identifiés par la même adresse MAC source non autorisée.

Pour s'assurer que ce filtrage n'impacte pas les autres chemins, nous testons ensuite la connectivité entre **h4 Bloc E** et **h3**, qui transitent par le switch **s5** non concerné par la règle.

Dans la figure 3.23 le résultat de la commande ci-après :

```
h4 ping h3
```



```
mininet> h4 ping h3
PING 10.0.3.3 (10.0.3.3) 56(84) bytes of data.
64 bytes from 10.0.3.3: icmp_seq=1 ttl=64 time=0.809 ms
64 bytes from 10.0.3.3: icmp_seq=2 ttl=64 time=0.127 ms
64 bytes from 10.0.3.3: icmp_seq=3 ttl=64 time=0.127 ms
^C
--- 10.0.3.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2059ms
rtt min/avg/max/mdev = 0.127/0.354/0.809/0.321 ms
mininet>
```

FIG. 3.23 : Deuxième validation du test (filtrage des flux interdits)

prouve que le filtrage est bien localisé au switch **s3 Bloc A** et n'interfère pas avec les autres liaisons de la topologie.

3.7 Conclusion

Dans ce dernier chapitre, nous avons d'abord exposé les contraintes matérielles et motivé le recours à une émulation logicielle (Mininet, MiniEdit) face à l'incompatibilité des commutateurs physiques avec OpenFlow. Nous avons par la suite détaillé la mise en place de l'environnement : installation et configuration de Mininet, déploiement d'OpenDaylight, intégration d'OpenFlow Manager et captures Wireshark. La réalisation de la topologie SDN a été suivie d'une phase de tests rigoureux. Les essais de connectivité ont confirmé l'établissement et la stabilité de la session OpenFlow, avec 0% de perte de paquets.

Enfin, les tests de sécurité ont démontré la capacité à injecter dynamiquement, via REST-CONF et OFM, des règles de filtrage MAC, validées par l'échec du ping ciblé et l'absence d'impact sur les autres flux. L'ensemble de ces expérimentations atteste la faisabilité, la robustesse et la programmabilité de notre solution SDN émulée, validant ainsi la pertinence de l'architecture et des mécanismes mis en œuvre.

Conclusion générale

Dans un contexte où les réseaux informatiques doivent répondre à des exigences croissantes en matière de flexibilité, de gestion centralisée et de résilience, ce travail a exploré le paradigme du *Software-Defined Networking* (SDN) comme solution innovante. L'objectif principal était de proposer, modéliser et valider une architecture réseau basée sur le SDN, adaptée à un environnement de type campus.

Dans une première phase, nous avons mené une étude théorique approfondie afin de maîtriser les fondements du Software-Defined Networking. Cette phase a permis de comprendre les concepts clés tels que la séparation des plans de contrôle et de données, le rôle du contrôleur SDN, ainsi que les protocoles de communication, en particulier OpenFlow. Cette base conceptuelle a mis en évidence les bénéfices que le SDN peut apporter en termes de programmabilité, d'automatisation et de simplification de la gestion réseau.

La seconde phase a consisté à analyser de manière critique les limites d'une architectures réseau classique, notamment sa complexité de gestion et son manque de flexibilité. Sur cette base, une nouvelle architecture a été conçue, reposant sur une logique hiérarchique et centralisée autour d'un contrôleur SDN (OpenDaylight), avec l'intégration de mécanismes de redondance destinés à assurer une haute disponibilité du réseau.

Enfin, une phase de mise en œuvre pratique a été entreprise. Grâce à des outils d'émulation comme Mininet et MiniEdit, l'architecture proposée a été déployée dans un environnement virtuel. Les commutateurs virtuels ont été configurés, le contrôleur SDN intégré, et plusieurs scénarios de test ont été exécutés. Ceux-ci ont permis de valider la connectivité inter-hôtes, d'évaluer la résilience de l'infrastructure et de vérifier la capacité à filtrer des flux réseau à l'aide de règles dynamiques via OpenFlow Manager.

L'ensemble de ces étapes a démontré la faisabilité et l'efficacité de l'architecture SDN que nous avons conçue et validée par émulation. Elle répond aux objectifs fixés en matière de centralisation, de flexibilité et de sécurité.

Pour approfondir ce travail, plusieurs perspectives peuvent être envisagées. Parmi elles, l'introduction d'un second contrôleur SDN pour éviter le *seul point de défaillance*, l'intégration de fonctions avancées de monitoring, ou encore le déploiement sur une infrastructure physique réelle afin de comparer les performances entre environnement émulé et environnement réel.

Ainsi, ce projet ouvre la voie à des recherches et des applications concrètes autour du SDN, dans un monde où la maîtrise du réseau devient un enjeu stratégique.

Bibliographie

- [1] D. KREUTZ, F. M. V. RAMOS, P. VERISSIMO, C. E. ROTHENBERG, S. AZODOLMOLKY et S. UHLIG, « Software-Defined Networking : A Comprehensive Survey », *Proceedings of the IEEE*, t. 103, n° 1, p. 14-76, 2015. DOI : 10.1109/JPROC.2014.2371999.
- [2] S. HAJI, S. ZEEBAREE, R. SAEED et al., « Comparison of Software Defined Networking with Traditional Networking », *Asian Journal of Computer Science and Information Technology*, t. 9, p. 1-18, mai 2021. DOI : 10.9734/AJRCOS/2021/v9i230216.
- [3] M. SUBRAMANIAN, *Network Management : Principles and Practice*. Addison-Wesley, 2000.
- [4] T. BENSON, A. AKELLA et D. A. MALTZ, *Unraveling the Complexity of Network Management*. Madison, WI et Redmond, WA : University of Wisconsin-Madison et Microsoft Research, 2014.
- [5] B. A. A. NUNES, M. MENDONÇA, X.-N. NGUYEN, K. OBRACZKA et T. TURLETTI, « A Survey of Software-Defined Networking : Past, Present, and Future of Programmable Networks », *IEEE Communications Surveys & Tutorials*, t. 16, n° 3, p. 1617-1634, 2014. DOI : 10.1109/SURV.2014.012214.00180. adresse : <https://inria.hal.science/hal-00825087v5>.
- [6] N. MCKEOWN, T. ANDERSON, H. BALAKRISHNAN et al., « OpenFlow : Enabling Innovation in Campus Networks », *ACM SIGCOMM Computer Communication Review*, t. 38, n° 2, p. 69-74, 2008.
- [7] P. BERDE, M. GEROLA, J. HART et al., « ONOS : Towards an Open, Distributed SDN OS », *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, p. 1-6, 2014.
- [8] J. GUCK, A. VAN BEMTEN et W. KELLERER, « A Survey of the Application of SDN in Network Security », in *5th IEEE International Conference on Software Defined Systems (SDS)*, 2015.
- [9] Y. LI, M. ZHAO, D. ZHOU et al., « Intelligent 5G : When Cellular Networks Meet Artificial Intelligence », *IEEE Wireless Communications*, t. 26, n° 3, p. 76-83, 2019.
- [10] L. RICHARDSON et S. RUBY, *RESTful Web Services*. O'Reilly, 2007.
- [11] A. LARA, A. KOLASANI et B. RAMAMURTHY, « Network Innovation using OpenFlow : A Survey », *IEEE Communications Surveys & Tutorials*, t. 16, n° 1, p. 493-512, 2014. DOI : 10.1109/SURV.2013.081313.00105.
- [12] N. MCKEOWN, T. ANDERSON, H. BALAKRISHNAN et al., « OpenFlow : Enabling Innovation in Campus Networks », *ACM SIGCOMM Computer Communication Review*, t. 38, n° 2, p. 69-74, 2008.
- [13] J. D. CASE, M. FEDOR, M. L. SCHOFFSTALL et J. DAVIN, *Simple Network Management Protocol (SNMP)*, RFC 1157, Accessed : 2025-04-26, 1990. adresse : <https://www.rfc-editor.org/rfc/rfc1157>.

- [14] R. ENNS, *NETCONF Configuration Protocol*, RFC 4741, <https://tools.ietf.org/html/rfc4741>, Consulté en avril 2025, 2006.
- [15] M. B. AL-SOMAIDAI et E. B. YAHYA, « Survey of software components to emulate OpenFlow protocol as an SDN implementation », *Department of Electrical Engineering, Mosul University*, 2015.
- [16] A. TOOTOONCHIAN, S. KANDULA, S. BANERJEE, M. ZAHARIA et I. STOICA, « On controller performance in software-defined networks », in *Proceedings of the 2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, 2012. adresse : <https://www.usenix.org/conference/hot-ice12/workshop-program/presentation/tootoonchian>.
- [17] N. HANDIGOL, S. SEETHARAMAN, M. FLAJSLIK, N. MCKEOWN et R. JOHARI, « Plug-n-Serve : Load-Balancing Web Traffic Using OpenFlow », in *ACM SIGCOMM Demo*, Demo at ACM SIGCOMM, 2009.
- [18] OPENDAYLIGHT, *OpenDaylight Project*, Accessed 22 Feb 2019, 2019. adresse : <https://www.opendaylight.org/>.
- [19] P. AGRAWAL, « RAFT : A Recursive Algorithm for Fault Tolerance », in *Proceedings of the International Conference on Parallel Processing (ICPP)*, 1985, p. 814-821.
- [20] S. SEZER, S. SCOTT-HAYWARD, P.-K. CHOUHAN et al., « Are we ready for SDN ? Implementation challenges for software-defined networks », *IEEE Communications Magazine*, t. 51, n° 7, p. 36-43, 2013.
- [21] OPEN NETWORKING FOUNDATION (ONF), *OpenFlow Switch Specification, Version 1.4.0 (Wire Protocol 0x05)*, <https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.4.0.pdf>, Accessed 8 May 2025, oct. 2013.
- [22] MICROSOFT, *Fin du support pour Windows*, Consulté en mai 2025, 2024. adresse : <https://www.microsoft.com/fr-fr/windows/end-of-support?msocid=0ee0f6f7a9fb62141784e546a88c63bb>.
- [23] « Pourquoi et comment hiérarchiser un réseau ? » (2014), adresse : <https://giantsnet.blogspot.com/2014/03/reseau-hierarchique.html> (visité le 08/05/2025).
- [24] « Modélisation hiérarchique du réseau – Pourquoi et comment hiérarchiser ? » (2020), adresse : <https://fr.scribd.com/document/487866903/MODELISATION-HIERARCHIQUE-DU-RESEAU-Pourquoi-et-comment-hierarchiser-1> (visité le 08/05/2025).
- [25] W. ODOM, *CCNA 200-301 Official Cert Guide, Volume 1*, 2nd. Indianapolis, IN : Cisco Press, 2020, ISBN : 9780135792735.
- [26] H. WANG, « Application of the Stacking and LACP Technology in Communications Networks », *Journal of Physics : Conference Series*, t. 1769, n° 1, p. 012073, 2021. DOI : 10.1088/1742-6596/1769/1/012073. adresse : https://www.researchgate.net/publication/349538116_Application_of_the_Stacking_and_LACP_Technology_in_Communications_Networks.
- [27] CISCO SYSTEMS, *Illustration de l'empilage : modules de service Cisco EtherSwitch et commutateurs Catalyst 3750*, Référence interne Cisco 121915 – fournit une bande passante complète et des connexions redondantes, Cisco Systems, 2008. adresse : <https://www.cisco.com/c/en/us/products/collateral/switches/catalyst-3750-series-switches/white-paper-c11-670482.html>.
- [28] F. BURNEL. « Qu'est-ce qu'une DMZ en informatique ? » (2021), adresse : <https://www.it-connect.fr/informatique-cest-quoi-une-dmz/> (visité le 08/05/2025).
- [29] SUPERMICRO, *SSE-G3748-SONIC Datasheet*, https://www.supermicro.com/datasheet/datasheet_SSE-G3748-SONIC.pdf, Consulté le 22 mai 2025, 2023.

- [30] J. GOMEZ et K. KFOURY, « A survey on network simulators, emulators, and testbeds used for research and education », *Computer Networks*, t. 237, p. 110 054, 2023. DOI : 10.1016/j.comnet.2023.110054. adresse : <https://www.sciencedirect.com/science/article/pii/S1389128623004991>.
- [31] APPOSITE TECHNOLOGIES, *Network Emulation vs. Simulation*, Consulté le 22 mai 2025, 2025. adresse : <https://apposite-tech.com/network-emulation-vs-simulation/>.
- [32] B. LANTZ, B. HELLER et N. MCKEOWN, « A Network in a Laptop : Rapid Prototyping for Software-Defined Networks », in *Proc. of the 9th ACM Workshop on Hot Topics in Networks (HotNets-IX)*, Monterey, CA, USA, oct. 2010.
- [33] MININET PROJECT, *Mininet : An Instant Virtual Network on Your Laptop*, <http://mininet.org>, Consulté le 12 juin 2025.
- [34] A. BOUKAIS, « L'ingénierie du trafic dans un réseau MPLS sous SDN », Département de Télécommunications, Faculté de Génie Électrique, Mémoire de Master, Université des Sciences et de la Technologie Houari Boumediene, Alger, Algérie, juin 2023.
- [35] CISCO DEVNET, *OpenDaylight OpenFlow Manager (OFM) App*, GitHub repository : <https://github.com/CiscoDevNet/OpenDaylight-Openflow-App>, Consulté le 12 juin 2025.
- [36] DAVID BOMBAL, *ODL OFM install with GNS3*. adresse : <https://github.com/davidbombal/GNS3Talks/blob/master/ODL%20OFM%20%20install%20with%20GNS3.txt>.
- [37] WIRESHARK FOUNDATION, *Wireshark*, Logiciel disponible sur <https://www.wireshark.org/>, Consulté le 15 juin 2025.
- [38] A. K. MADDALA, « Modeling of OpenFlow-Based Software Defined Networks using Mininet », Description et interface de l'outil MiniEdit (pp. 15–19), Master's thesis, Iowa State University, Ames, Iowa, 2019, chap. 3.4.