



République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Electronique et d'Informatique

Département Informatique

Mémoire de Licence

Option

Génie des Télécommunications et Réseaux

Thème

Application du Codage Réseau pour le SDN

Proposé et Dirigé par :

Professeur MERAZKA Fatiha

Présenté par :

Melle. ACHAIBOU Radia Amina

Melle. HAMENNICHE Amira

Devant le jury composé de :

Mr F.FARAH

Président

Mr M.Hamidia

Membre

Binôme N°178/2018

REMERCIEMENTS

Après avoir rendu grâce à Dieu le tout Puissant et le Miséricordieux, nous tenons à exprimer nos respectueux remerciement à notre promotrice Mme MERAZKA Fatiha pour l'aide qu'elle a bien voulu nous accorder tout le long de notre travail, pour ses précieux conseils, ses orientations bienveillantes, son infatigable dévouement.

Nous tenons à remercier l'équipe IEEE Collabratec, en particulier Mr Jonas Hansen et Mr Péter Vingelmann, pour leur implication dans notre recherche. Ils nous ont donné des conseils avisés et ont été une grande ressource pour l'avancement de notre projet.

Nous remercions les membres du jury Mr FARAH et Mr Hamidia pour l'honneur d'avoir voulu examiner et évaluer ce travail et nous tenons à vous exprimer tout notre respect et notre estime.

Nous tenons à remercier nos parents pour leurs soutien moral et physique et leurs inéluctable patience et c'est grâce à eux que ce projet a pu voir le jour. Enfin, nous remercions tous ceux qui ont contribué de près ou de loin à la réalisation de ce mémoire.

Radia Amina ACHAIBOU
&
Amira HAMENNICHE

Table des matières

Table des figures	IV
Liste des tableaux	V
List des acronymes	VI
Introduction générale	1
I Généralités sur le SDN	4
I-1 Introduction	3
I-2 Définition	3
I-3 Architecture	4
I-3-1 Plan de données	4
I-3-2 Plan de contrôle	4
I-3-3 Plan d'application	4
I-3-4 Interfaces vers le sud (SBI)	5
I-3-5 Interfaces vers le Nord (NBI)	5
I-3-6 Interfaces Est/Ouest	6
I-4 Le protocole OpenFlow	6
I-5 Programmabilité dans le SDN	7
I-5-1 Le contrôleur	8
I-5-2 La virtualisation des fonctions réseaux (NFV)	8
I-5-3 Intelligence dans les équipements	9
I-6 L'orchestrateur	9
I-7 Applications SDN	10
I-7-1 Data center et Cloud computing	10
I-7-2 Multimédia et QOS	10
I-7-3 Les Réseaux sans fil	10
I-8 Problèmes vs Solutions	10
I-9 Conclusion	11
II Le Codage Réseau	13
II-1 Introduction :	12
II-2 Définition	12
II-3 Classification	13
II-3-1 Selon le flux	13
II-3-2 Selon la couche OSI	13
II-4 Les domaines d'application du Codage Réseau	14
II-4-1 Partage de fichier	14
II-4-2 Transmission Multicast	14
II-4-3 Les réseaux sans fil	15
II-4-4 Sécurité réseau	16
II-5 Exemples de codage réseau	16
II-5-1 Pourquoi le Codage réseau XOR	16
II-5-2 Principe de fonctionnement général	17
II-6 Conclusion	18

III Conception de la solution	20
III-1 Introduction	19
III-2 Architecture de la solution	19
III-2-1 L'hyperviseur KVM	20
III-2-2 Le contrôleur	20
III-2-3 OpenVswitch	21
III-3 Le codage appliqué	22
III-3-1 Fonctionnement général	22
III-3-2 Structure du paquet	23
III-3-3 Exemple de Codage Réseau basé sur XOR	24
III-3-4 Interface Tun	24
III-3-5 Diagramme schématisant la structure des applications	25
III-4 Conclusion	26
IV Simulation et résultats	28
IV-1 Introduction	27
IV-2 Environnement d'évaluation	27
IV-2-1 Installation de KVM	27
IV-2-2 Installation du contrôleur	28
IV-2-3 Installation du commutateur OpenVswitch	29
IV-3 Présentation de la topologie implémentée	31
IV-3-1 Explication du script de la topologie	32
IV-3-2 Validation de la topologie	35
IV-4 Analyse des résultats de la simulation	37
IV-4-1 Métrique de performance	37
IV-4-2 Iperf	37
IV-4-3 Génération des tests	37
IV-4-4 Résultats	41
IV-4-5 Conclusion	42
Conclusion général	43
Bibliographie	I
Annexes	IV

Table des figures

Figure I.1 : Architecture de SDN	4
Figure I.2 : Les APIs existantes dans le système SDN	5
Figure I.3 : Le commutateur OpenFlow et ses différentes parties	6
Figure I.4 : Modèles de programmabilité en SDN	7
Figure I.5 : Exemple d'utilisation d'un orchestrateur	9
Figure II.1 : Exemple classique de codage réseau	13
Figure II.2 : Le principe de codage réseau dans les réseaux sans fil	15
Figure II.3 : Exemple de la sécurité réseau en utilisant du codage réseau	16
Figure II.4 : Schéma d'un réseau Butterfly	18
Figure III.1 : Architecture de la solution	19
Figure III.2 : Position de l'hyperviseur dans une plateforme Linux	20
Figure III.3 : Fonctionnement interne d'Openvswitch	22
Figure III.4 : Structure du paquet xor	23
Figure III.5 : Exemple de codage/décodage XOR	24
Figure III.6 : Structure logicielle des applications implémentées	25
Figure IV.1 : Vérification de la virtualisation au niveau de la CPU	27
Figure IV.2 : Vérification de la version du processeur 64/32 bits	27
Figure IV.3 : Installation tmux	27
Figure IV.4 : vérification de l'installation de kvm au niveau du noyau Linux	27
Figure IV.5 : Ajout d'un utilisateur dans le groupe KVM	28
Figure IV.6 : Exécution du contrôleur OpenDaylight	29
Figure IV.7 : Démarrage d'OpenVswitch et d'OVSDB	30
Figure IV.8 : Terminal d'une VM	31
Figure IV.9 : Topologie de transmission à saut unique	32
Figure IV.10 :Préparation de l'environnement	32
Figure IV.11 :Configuration des interconnexions	33
Figure IV.12 :Configuration des équipements	33
Figure IV.13 :Création des interfaces virtuelles	34
Figure IV.14 :Démarrage des VMs	34
Figure IV.15 :Démarrage des OVS et configuration du partage	35
Figure IV.16 :Lancement de la topologie à saut unique	35
Figure IV.17 :Vérification de la liaison entre les deux machines virtuelles	36
Figure IV.18 :Topologie à saut unique vu par l'interface graphique d'ODL	36
Figure IV.19 :Initialisation de l'environnement de test	38
Figure IV.20 :La fonction de test run_iperf	38
Figure IV.21 :La fonction de test run_uncoded	39
Figure IV.22 :La fonction de test run_xor	39
Figure IV.23 :Générations des tests-1	40
Figure IV.24 :Générations des tests-2	40
Figure IV.25 :Performance de codage à saut unique avec un délai=10	41
Figure IV.26 :Taux de récupération avec un délai=10	42

Liste des tableaux

I.1	Tableau de comparaison de l'architecture traditionnel et l'architecture du SDN	11
III.1	Tableau de comparaison de quelques contrôleurs les plus connus	21

List des acronymes

<i>ALM</i>	: Application Layer Multicast
<i>API</i>	: Application Programming Interface
<i>CLI</i>	: Command Line Interface
<i>ETSI</i>	: European Telecommunications Standards Institute
<i>FEs</i>	: Forwarding Element
<i>FIB</i>	: Forwarding tables
<i>GUI</i>	: Graphical User Interface
<i>IDC</i>	: International Data Corporation
<i>KVM</i>	: Kernel-based Virtual Machine
<i>LNC</i>	: Linear Network Coding
<i>MTU</i>	: Maximum Transmission Unit
<i>NAT</i>	: Network Address Translation
<i>NBI</i>	: NorthBound Interface
<i>NC</i>	: Network Coding
<i>NFV</i>	: Network Function Virtualization
<i>NIB</i>	: Network Information Base
<i>NPU</i>	: Network Processing Units
<i>ODL</i>	: OpenDayLight
<i>ONF</i>	: Open Networking Fondation
<i>OS</i>	: Operating System
<i>OSI</i>	: Open Systems Interconnection
<i>OVS</i>	: Open vSwitch
<i>OVSDB</i>	: Open vSwitch Database
<i>PNC</i>	: Physical Layer Network Coding
<i>P2P</i>	: Peer to Peer
<i>RESTAPI</i>	: Representational State Transfer
<i>SBI</i>	: SouthBound Interface
<i>SDN</i>	: Software-Defined Networking
<i>SDWN</i>	: Software Defined Wireless Network
<i>SSL</i>	: Secure Sockets Layer
<i>TCP</i>	: Transmission Control Protocol
<i>Tmux</i>	: Terminal Multiplexer
<i>TUN</i>	: Tunnel
<i>TLS</i>	: Transport Layer Security
<i>VLAN</i>	: Virtual LAN
<i>VM</i>	: virtual machine
<i>VSDN</i>	: Video over Software-Defined Networking
<i>XOR</i>	: eXclusive OR

Introduction générale

Avec l'avènement des innovations technologiques récentes telles que la virtualisation des systèmes et le cloud computing, les limites actuelles des architectures réseaux deviennent de plus en plus problématiques pour les opérateurs et les administrateurs réseaux. En effet, depuis plusieurs années, il est communément admis que les architectures IP traditionnelles sont, d'une part, particulièrement complexes à configurer à cause de la nature distribuée des protocoles réseaux et, d'autre part, difficile à faire évoluer en raison du fort couplage qui existe entre le plan de contrôle et le plan de données des équipements d'interconnexion existants.

Le problème est que le manque d'innovation dans l'univers des réseaux a fait peser des contraintes importantes sur le déploiement des applications réseaux. Les constructeurs ont donc fini par se décider à tenter d'apporter aux réseaux le niveau de flexibilité et de simplicité dont ont besoin les entreprises et les fournisseurs de services. Les principaux domaines d'innovation au cours des dernières années sont à chercher du côté du contrôle centralisé, de la programmabilité, et de la virtualisation. Ces innovations sont regroupées sous l'appellation commune de Software-Defined Network.

Le paradigme SDN est une nouvelle approche qui a pour ambition de répondre à cette rigidité architecturale des réseaux IP actuels, notamment en les rendant plus programmables. Il fournit aussi des services innovants, notamment la sécurité, la gestion de la bande passante et l'ingénierie du trafic. En revanche, le SDN rencontre souvent des problèmes de pertes, de gigue et du délai au niveau du réseau qui restent des grands obstacles surtout dans les réseaux de virtualisation et les réseaux volumineux tel que la 5G et qui réduisent ses performances. Dans l'optique d'éradiquer ces problèmes (gigue, perte de paquet, délai ...) nous avons eu recours au codage réseau qui a montré un grand potentiel dans divers scénarios de communication en modifiant les principes d'acheminement des paquets des réseaux actuels. Il peut améliorer non seulement le débit, la latence et la fiabilité mais aussi le besoin de coordination dans de nombreux scénarios.

L'objectif de notre projet, est de combiner ces deux paradigmes. Pour cela nous avons proposé une stratégie de codage réseau que nous avons implémenté dans une architecture SDN.

Ce mémoire est organisé en 4 chapitres :

- **Chapitre 1** : nous le consacrerons à un état de l'art des réseaux SDN. Nous commencerons par définir les réseaux SDN puis nous décrirons son architecture composée des plans de données, de contrôle, d'application et les interfaces intermédiaires entre ces plans. Par la suite, nous décrirons les différents modèles de programmabilité en utilisant un contrôleur centralisé, la virtualisation des fonctions réseaux et l'intelligence dans les équipements.
- **Chapitre 2** : a pour objectif de présenter un état de l'art sur le paradigme de codage réseau. Nous décrirons d'abord cette discipline, puis nous présenterons les multiples approches et classifications : selon le flux et la couche OSI. Par la suite, nous discuterons les multiples exemples d'applications de codage réseau pair-à-pair et les réseaux sans fil.

- **Chapitre 3** : détaillera la conception de notre solution qui vise à mettre en place une stratégie de codage réseau dans une architecture SDN. Nous commencerons par définir les prérequis de cette solution et nous expliquerons, ensuite, l'algorithme de codage réseau choisi.
- **Chapitre 4** : est un manuel de mise en place d'une architecture SDN virtualisée supportant le codage réseau. Afin de montrer l'impact de l'intégration proposée, une série de mesures de performance sera réalisée à l'aide de l'outil standard iperf pour la mesure du rendement du réseau.

Nous terminerons ce mémoire par une conclusion générale et les perspectives des travaux futurs.

Chapitre I

Généralités sur le SDN

I-1 Introduction

Les réseaux informatiques sont une technologie complexe qui permet aux appareils terminaux de communiquer les uns avec les autres. Une infrastructure réseau typique comprend des routeurs, des commutateurs, des serveurs, des serveurs web, des pare-feux, des équilibreurs de charge, des systèmes de prévention d'intrusion et d'autres périphériques. L'efficacité, la fiabilité, la flexibilité et la robustesse sont les conditions requises pour le traitement et la gestion de l'énorme quantité de données envoyées sur le réseau. Cela a amené les entreprises de fabrication de dispositifs réseau à implémenter des protocoles complexes et gourmands en ressources qui permettent aux routeurs et commutateurs de communiquer entre eux par commutation de paquets et en créant une topologie de réseau à des fins de routage, ces solutions de routage ont des exigences élevées en matière de performance et d'évolutivité du réseau ce qui a rendu le contrôle inflexible.

La solution à ce dilemme est tirée de la technologie de virtualisation, plusieurs machines virtuelles exécutées sur un hyperviseur sont extraites du matériel de la machine physique et partagent les mêmes ressources. Ce paradigme est adopté par le Software Defined Networking, l'une des technologies les plus révolutionnaires étant une substitution aux réseaux traditionnels.

Ce chapitre a pour but de vulgariser la technologie SDN en se basant sur des recherches théoriques qui serviront notre solution. Nous présenterons le concept et l'architecture du SDN ainsi que le protocole OpenFlow et les différents modèles de programmation du réseau afin de mieux interagir avec les applications. Nous expliciterons le rôle du contrôleur au sein de cette architecture ainsi que l'intérêt qu'a suscité SDN auprès des chercheurs et des industriels dans le domaine des réseaux.

I-2 Définition

SDN signifie littéralement Software-Defined Networking, lancé par L'ONF (consortium d'entreprises à but non lucratif fondé en 2011 pour promouvoir SDN et normaliser ses protocoles) ; c'est un réseau défini par application qui permet l'innovation réseau basé sur quatre principes fondamentaux [1] :

- Les plans de contrôle et d'acheminement du réseau sont clairement découplés.
- Les décisions d'acheminement sont basées sur les flux plutôt que sur les destinations.
- La logique d'acheminement est extraite du matériel vers une couche logicielle programmable.
- Un élément appelé un contrôleur, est introduit pour coordonner les décisions d'acheminement à l'échelle du réseau.

Notons que l'ONF regroupe plus de 100 entreprises comportant les géants du web, les opérateurs de télécoms et les fabricants ce qui montre l'intérêt immense des industriels pour SDN.

I-3 Architecture

Dans cette section, nous présenterons l'architecture du SDN [RFC7276] et décrirons ses principales composantes [?]. La Figure I.1 schématise l'architecture conceptuelle du SDN.

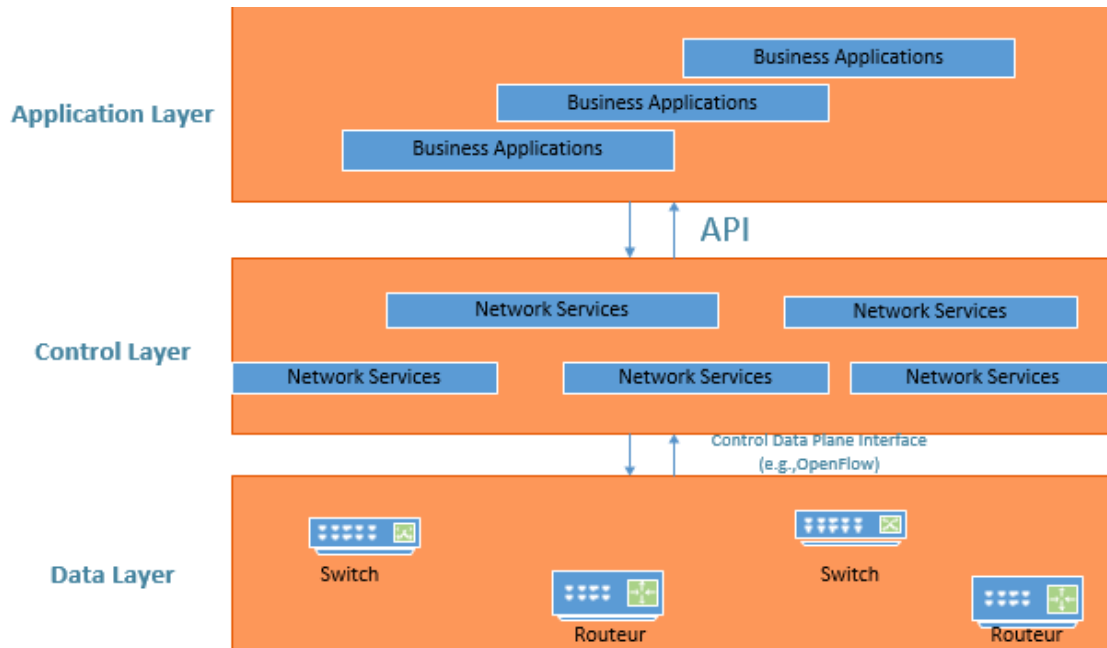


Figure I.1 – Architecture de SDN

I-3-1 Plan de données

C'est la couche la plus basse, elle contient les équipements de transmission FEs tels que les switchs physiques et virtuels. Son rôle principal est de transmettre les données en se basant sur des informations contenues dans des tables FIBs, surveiller les informations locales et collecter les statistiques.

I-3-2 Plan de contrôle

Le plan de contrôle se situe entre le plan de données et le plan d'application, constitué d'un ou de plusieurs logiciels de contrôle (Contrôleurs). Ces contrôleurs utilisent des interfaces Sud ouvertes pour contrôler le comportement des FEs et communiquent via des APIs Nord avec la couche supérieure pour superviser et gérer le réseau. Le plan fournit également des services pour que les applications utilisent le plan de données et les données collectées pour fournir d'autres fonctions au sein du réseau.

I-3-3 Plan d'application

Ce plan héberge les applications qui peuvent introduire de nouvelles fonctionnalités réseau, comme la sécurité, la configuration dynamique et la gestion. La couche d'application exploite la vue globale et distante du réseau offerte par le plan de contrôle pour fournir des directives appropriées à la couche de contrôle. Des exemples d'applications réseau seront présentés dans la section 7 de ce chapitre.

La Figure I.2 montre les différents APIs existantes dans le système SDN.

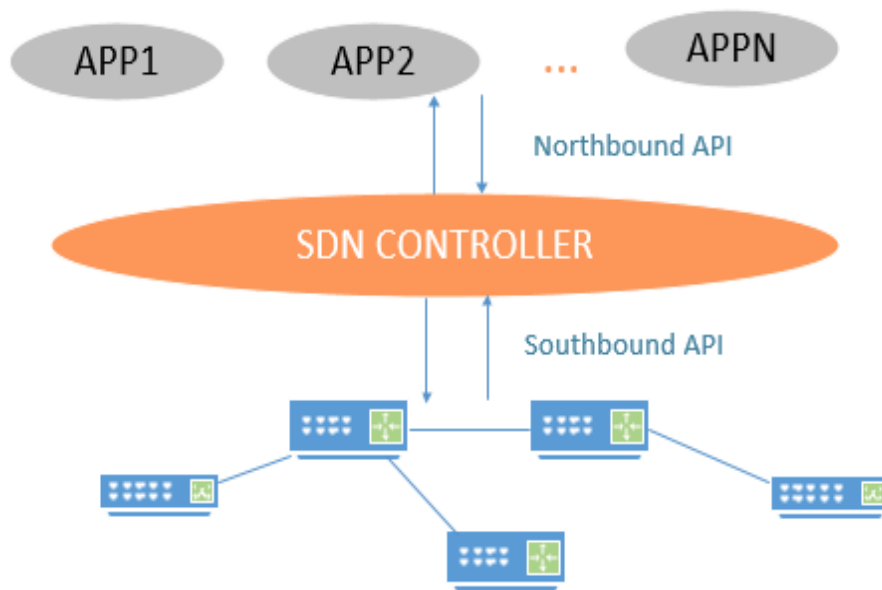


Figure I.2 – Les APIs existantes dans le système SDN

I-3-4 Interfaces vers le sud (SBI)

Ce sont des interfaces de communication permettant au contrôleur d'interagir avec les switches/routeurs du plan de données. Le protocole le plus utilisé et le plus déployé comme interface de Sud est Open-Flow, qui est un élément fondamental publié par l'ONF pour construire des solutions SDN, ce protocole est traité dans la section 4 [2].

I-3-5 Interfaces vers le Nord (NBI)

Les interfaces Nord servent à programmer les éléments de transmission en exploitant l'abstraction du réseau fourni par le plan de contrôle. Elles sont considérées davantage comme des APIs que comme protocole de programmation et de gestion du réseau [3]. Au moment de la rédaction de ce document, il n'existe aucun standard intervenant entre la couche de contrôle et celle d'application du côté d'ONF ou d'autres organisations. Selon l'ONF, plusieurs niveaux d'abstraction et différents cas d'utilisation peuvent être caractérisés, ce qui signifie qu'il peut y avoir plusieurs interfaces Nord pour servir tous les cas d'utilisation. Parmi les propositions des industriels, nous trouvons une API basée sur REST API pour fournir une interface programmable utilisable par les applications du commerce [?].

I-3-6 Interfaces Est/Ouest

Les interfaces côté Est/Ouest sont des interfaces de communication qui permettent généralement la communication entre les contrôleurs dans une architecture multi-contrôleurs pour synchroniser les états du réseau [?]. Ces architectures sont très récentes et aucun standard de communication inter-contrôleur n'est actuellement disponible.

I-4 Le protocol OpenFlow

Dans le fonctionnement actuel, chaque équipement d'un réseau opère de façon distribuée. Le SDN propose donc de créer un point central qui gère le plan de contrôle, tandis que les switches/routeurs physiques n'auraient plus qu'à s'occuper du plan de données.

Pour réaliser cela, l'ONF a publié un protocole standard pour transmettre des instructions qui permettent de programmer le plan de contrôle d'un équipement utilisé au niveau SBI dans l'architecture SDN qui est OpenFlow. Ces instructions (appelées flow entries dans OpenFlow) sont des règles avec un pattern (ip source ou destination, mac adresse, port TCP...) et une action correspondante (rejeter le paquet, transmettre sur le port x, ajouter une en-tête VLAN...). Ce Protocole permet un routage flexible des flux réseau sans interrompre les autres trafics. Cette possibilité a été obtenue grâce à la séparation du plan de données et de contrôle.

En utilisant le protocole OpenFlow, un contrôleur peut non seulement définir des règles de transfert en envoyant des entrées de flux, mais aussi apprendre les statistiques réseau, les détails matériels, les ports, l'état de connectivité et la topologie réseau des commutateurs Ethernet.

Comme le montre la Figure I.3 ci-après, un commutateur OpenFlow comprend au moins trois parties :

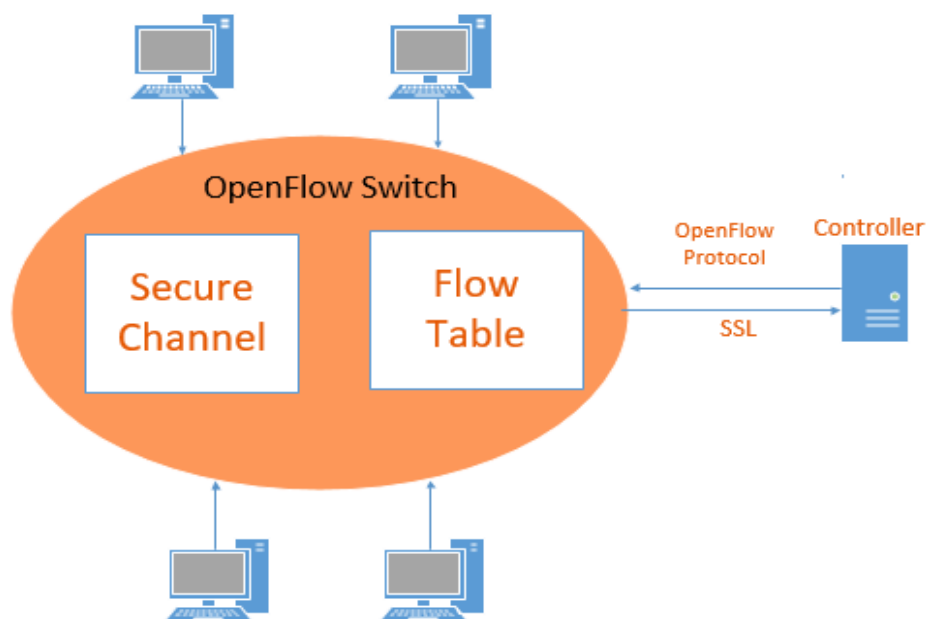


Figure I.3 – Le commutateur OpenFlow et ses différentes parties

1. **Une table de flux** : avec une action associée à chaque entrée de flux pour indiquer au commutateur comment traiter les flux.
2. **Un canal sécurisé** : est l'interface utilisée pour la connexion entre chaque commutateur OpenFlow et un contrôleur. Grâce à cette interface, le contrôleur peut recevoir des événements générés par le commutateur, envoyer des paquets hors du commutateur, configurer et gérer le commutateur. La connexion entre le commutateur et le contrôleur est cryptée avec le protocole TLS.
3. **Le protocole OpenFlow** : qui fournit un moyen ouvert et standard de communication avec un commutateur.

I-5 Programmabilité dans le SDN

Le SDN englobe toutes les solutions permettant une programmation du réseau, afin de mieux interagir avec les applications. Diverses solutions coexistent, adaptées selon les besoins des utilisateurs [1]. La Figure I.4 illustre les différents modèles de programmabilité.

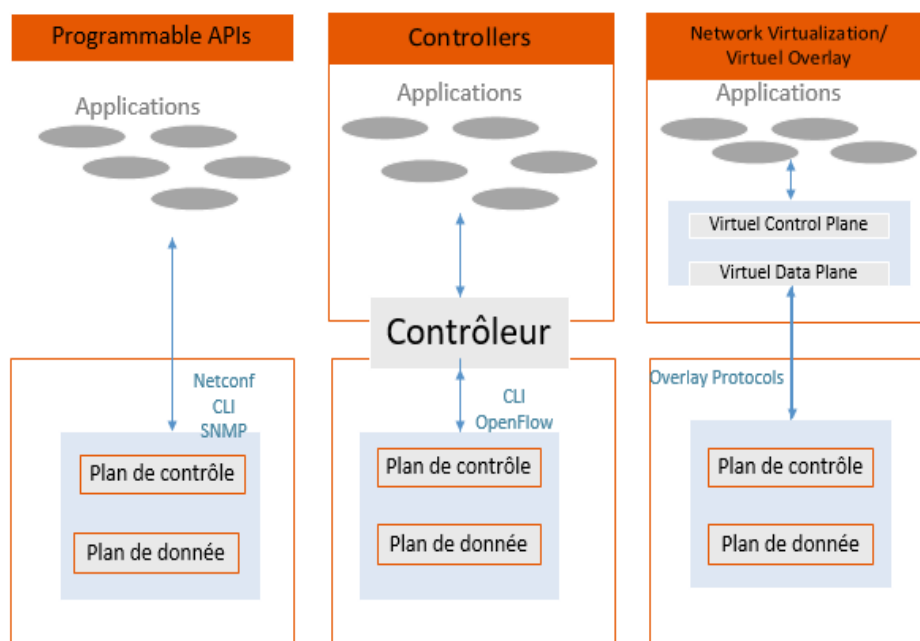


Figure I.4 – Modèles de programmabilité en SDN

- **Programmabilité individuelle de chaque équipement** : dans ce modèle une application interagit directement avec chaque équipement via des APIs. L'application est centralisée ou peut être localisée directement sur l'équipement réseau pour réaliser des tâches spécifiques.
- **Programmabilité via un contrôleur** : dans ce modèle, une application donne un ordre abstrait et global à un contrôleur, qui à son tour traduit cette requête en une suite d'ordres auprès des équipements du réseau concerné. Ce modèle est certainement le plus populaire puisqu'il permet de simplifier le réseau et masquer sa complexité.

- **Création d'un réseau virtuel au dessus du réseau physique** : dans ce modèle, les applications créent leur propre réseau «overlay», s'affranchissant des contraintes du réseau physique sous jacent. Ce dernier n'a pour mission que la simple connectivité entre les nœuds d'extrémité des tunnels et le réseau d'overlay qui assure l'intégralité des services.

On parle également de virtualisation des fonctions réseau NFV quand les routeurs, commutateurs, firewalls, etc sont des éléments virtualisés sur des serveurs.

I-5-1 Le contrôleur

Certaines solutions SDN reposent sur la mise en place de contrôleurs. Ces derniers ont pour mission de fournir une couche d'abstraction du réseau et de présenter ce dernier comme un système. Le contrôleur SDN permet d'implémenter rapidement un changement sur le réseau en traduisant une demande globale (par exemple : prioriser l'application X) en une suite d'opérations sur les équipements réseau (requêtes Netconf, ajouts d'états Openflow, configuration en CLI...). Les ordres sont donnés au contrôleur par une application via une API dite « Northbound ». Les éditeurs logiciels de contrôleurs publient la documentation de l'API afin de permettre d'interfacer des applications. Le contrôleur communique avec les équipements via une ou plusieurs API dites « Southbound ». Openflow se positionne comme une API sud agissant directement sur le plan de données.

Afin de pouvoir interagir avec le réseau, le contrôleur a besoin d'une vue précise du réseau. C'est ainsi que le concept de NIB a vu le jour. Cette NIB est construite au niveau du contrôleur et permet à ce dernier de savoir comment implémenter chaque ordre abstrait, trouver les équipements qui doivent être reconfigurés et s'assurer de la capacité de ces équipements à implémenter une directive.

I-5-2 La virtualisation des fonctions réseaux (NFV)

Lorsque les fournisseurs de services ont tenté d'accélérer le déploiement de nouveaux services réseau afin de faire progresser leurs plans de revenus et de croissance, ils ont constaté que les Appliance matérielles limitaient leur capacité à atteindre ces objectifs. Ils se sont tournés vers les technologies de virtualisation informatique standard et ont découvert que NFV permettait d'accélérer l'innovation et le provisionnement des services. Avec cela, plusieurs fournisseurs se sont regroupés et ont créé l'Institut européen des normes de télécommunications. La création de l'ETSI a abouti à la création des exigences de base et de l'architecture de NFV [4].

La virtualisation des fonctions réseau permet de tirer parti des techniques de virtualisation des technologies de l'information pour migrer des classes entières de fonctions réseau généralement hébergées sur du matériel propriétaire vers des plates-formes virtuelles basées sur des serveurs de calcul et de stockage généraux [5]. Chaque nœud de fonction virtuelle est connue comme la fonction de réseau virtualisé NFV, qui peut s'exécuter sur un seul ou un ensemble de VMs, au lieu d'avoir des dispositifs matériels personnalisés pour la fonction réseau proposée.

I-5-3 Intelligence dans les équipements

Programmer le réseau n'est possible que si chaque équipement offre une certaine dose de programmabilité. Or, les équipements réseau sont très divers, alors que certains ne doivent gérer que quelques kilobits par seconde pour des applications spécifiques (distributeurs de billets, capteurs divers...), d'autres routeurs sur les cœurs de réseau d'opérateurs doivent commuter plusieurs téraoctets par seconde.

Les différences entre les performances requises sur ces matériels imposent des architectures radicalement différentes. La plupart des constructeurs intègrent et/ou développent des NPU introduisant une dose de programmabilité tout en conservant les performances réseau attendues. Ces composants peuvent être reprogrammés pour implémenter les dernières innovations, comme par exemple un nouveau type d'encapsulation. Nous pouvons retenir que les approches SDN n'ont pas pour objectif d'éradiquer tout plan de contrôle sur les équipements réseau, au contraire elles cherchent à mieux les exploiter. Alors que, la programmation du réseau impose de nouvelles contraintes sur le plan de données des équipements réseau qui sont amenés à réaliser des actions plus complexes.

I-6 L'orchestrateur

Dans le cas de déploiement d'une application dans différents types d'environnement (WAN, LAN, DATACENTER ...) un seul contrôleur ne pourra pas programmer l'ensemble de la chaîne, ce qui a permis l'apparition de l'orchestrateur qui offre une programmation de bout en bout comme illustré dans la Figure I.5 [6]. Ce dernier reçoit un ordre depuis une application (par exemple un portail de fourniture de services) et effectue les actions nécessaires pour réaliser la tâche demandée.

L'orchestrateur doit être capable de valider chaque étape quand il réalise une action afin de revenir en arrière en cas d'échec. Il ne peut pas laisser le réseau dans un mode bancal où seule une partie aurait été configurée avec un nouveau service.

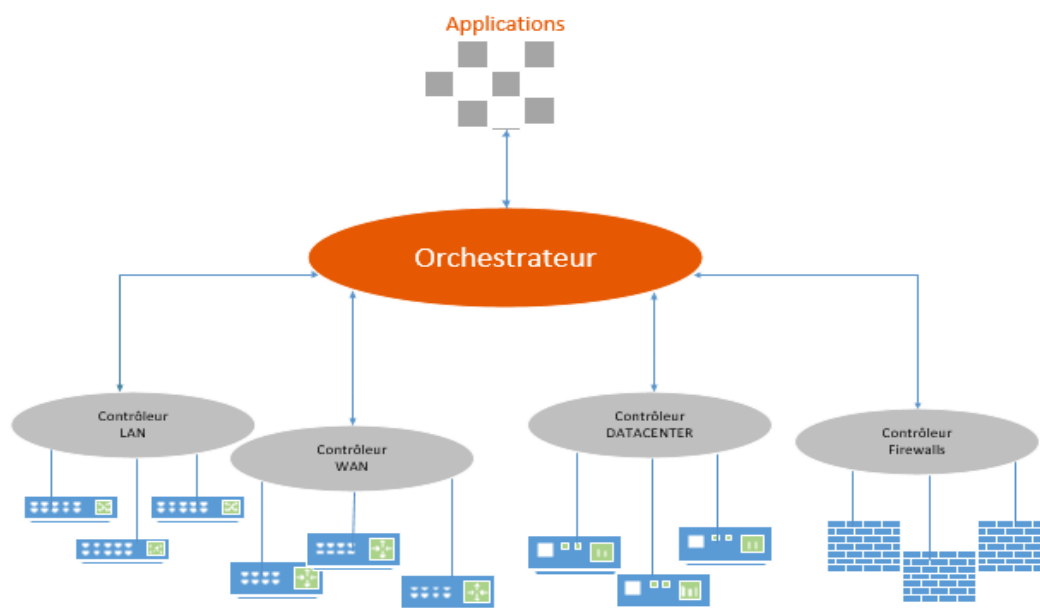


Figure I.5 – Exemple d'utilisation d'un orchestrateur

I-7 Applications SDN

Les réseaux SDN ont une large variété d'applications dans les environnements réseaux et permettent un contrôle personnalisé, ils simplifient également le développement et le déploiement de nouveaux services et protocoles réseaux. Selon IDC, le marché de SDN est composé des équipements d'infrastructure, des logiciels de contrôle et de virtualisation et des applications de réseau et de sécurité, il sera évalué à 12.5 milliards de dollars en 2020. Cette section présentera quelques exemples d'applications SDN .

I-7-1 Data center et Cloud computing

SDN est hautement considéré comme une des solutions les plus récentes, qui permet de configurer et de gérer facilement le Cloud et les centres de données. Le géant d'Internet Google a expérimenté les bénéfices du SDN dans la gestion et le contrôle de ses centres de données autour du globe [?].

I-7-2 Multimédia et QOS

L'architecture Internet d'aujourd'hui repose sur l'envoi des paquets sans prendre en considération le type des paquets et la qualité de la transmission. Les applications multimédias comme le streaming vidéo, la vidéo à la demande, la vidéo-conférence, etc nécessitent des ressources réseau stables et tolèrent les erreurs et les retards de transmission. En se basant sur la vue centralisée du réseau offerte par SDN, on peut sélectionner selon le débit, des chemins différents pour les divers flux du trafic, ce qui a permis la naissance d'une architecture réseau qui détermine la trajectoire optimale en utilisant une vue d'ensemble du réseau VSDN [?].

I-7-3 Les Réseaux sans fil

SDN A été aussi appliqué dans le domaine des réseaux sans fil, en particulier la 5G. En fait, selon le rapport d'Ericsson, le trafic mobile a augmenté de 60 % entre 2015 et le premier trimestre de l'année 2016. Gérer cette nouvelle masse de données est le plus grand défi des opérateurs, puisqu'une mauvaise gestion conduirait à une interférence entre les différents signaux sans fils. Le SDWN a été proposé comme solution d'intégration du SDN pour les réseaux sans fil, pour faciliter la gestion et supporter le déploiement de nouvelles applications dans l'infrastructure réseau.

I-8 Problèmes vs Solutions

Dans cette section, le Tableau I.1 met en avant les problèmes du réseau traditionnel qui ont poussé le SDN à voir le jour ainsi que les avantages qu'apporte ce dernier.

Problèmes	Solutions
Couplage du plan de contrôle et de données	Plan de contrôle et de données sont découplés
La complexité de gestion et de configuration	Directement programmable
Déploiement moins rapide	Programmation automatique
Dépendance aux fabricants	Normes ouvertes
Contrôle distribué	Centralement géré
Passage à l'échelle	Agile

Table I.1 – Tableau de comparaison de l'architecture traditionnel et l'architecture du SDN

I-9 Conclusion

Dans ce chapitre, nous avons présenté le concept et l'architecture de SDN et ses applications. Ces applications montrent l'intérêt qu'a suscité SDN auprès des chercheurs et des industriels dans le domaine des réseaux. Nous pensons que cet intérêt est dû à la simplification de l'architecture des équipements réseau et à l'abstraction du réseau. À travers les interfaces de communication fournies par le contrôleur, les développeurs peuvent communiquer avec le réseau et proposer de nouveaux services.

Alors que le concept de SDN offre plusieurs avantages comme la facilité de gestion des réseaux et l'introduction de nouvelles applications, il soulève cependant de nouveaux défis de performances surtout au niveau du plan de contrôle puisqu'il concentre toute la complexité et l'intelligence du réseau.

Chapitre II

Le Codage Réseau

II-1 Introduction :

Les systèmes en réseau apparaissent dans divers contextes de communication tels que les réseaux téléphoniques, l'Internet public, les réseaux pair-à-pair, les réseaux sans fil, ad-hoc et les réseaux de capteurs. De tels systèmes deviennent essentiels à notre mode de vie. Au cours du dernier demi-siècle, un important effort de recherche a été consacré à l'exploitation et à la gestion des réseaux, une prémisses fondamentale et inhérente derrière le fonctionnement de tous les réseaux de communication réside aujourd'hui dans la façon dont l'information est traitée. Qu'il s'agisse de paquets sur Internet ou de signaux dans un réseau téléphonique, si les flux d'informations indépendants proviennent de différentes sources, alors ils sont conservés séparément.

Aujourd'hui, le routage, le stockage de données, contrôle d'erreur et généralement toutes les fonctions de réseau fonctionnent sur ce principe. Ce n'est que récemment, avec l'avènement du codage réseau, que l'observation simple mais importante a été faite dans les réseaux de communication, nous pouvons permettre aux nœuds non seulement de transmettre mais aussi de traiter les flux d'informations indépendants entrants. Ce nouveau paradigme a émergé au tournant du millénaire et a immédiatement suscité un intérêt considérable dans les communautés de recherche.

Ce chapitre consiste en une recherche bibliographique sur le codage réseau. Nous définissons cette discipline et procédons à discuter les techniques de sa conception, puis nous citons les différentes classifications : selon le flux et les couche OSI. En outre, nous citons les divers domaines d'application de cette discipline et nous terminons ce chapitre par présenter la stratégie de codage réseau OU exclusif.

II-2 Définition

Network Coding NC est une nouvelle discipline dans laquelle les données transmises sont codées à la source, et décodées uniquement à la destination [?]. Cette nouvelle approche augmente le débit du réseau, réduit considérablement la latence, rend le réseau plus robuste et offre une meilleurs adaptabilité et une plus grande sécurité contre l'écoute clandestine, le piratage et d'autres formes d'attaques.

NC rompt avec le mécanisme actuel de stockage et retransmission (store-and-forward), dans lequel les paquets sont copiés puis transférés vers leur destination [7]. Les nœuds qui appliquent les fonctions de codage et recodage sur les flux de données sont appelés soit des nœuds sources, soit des nœuds intermédiaires, en fonction de leur position dans l'itinéraire réel entre deux dispositifs communicants. Implicitement, le nœud qui va appliquer la fonction inverse (décodage) pour récupérer le flux d'origine s'appelle le nœud récepteur. Par conséquent, chaque chemin contient toujours une source et une destination et peut contenir zéro ou plusieurs nœuds intermédiaires comme le montre la Figure 1, qui est un exemple classique implémentant le codage réseau.

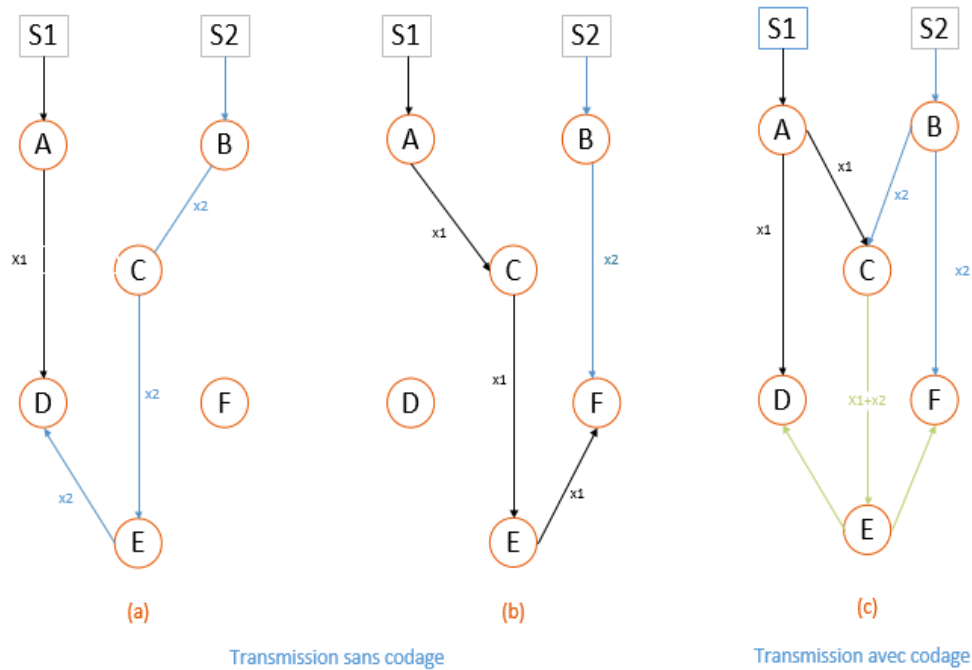


Figure II.1 – Exemple classique de codage réseau

II-3 Classification

II-3-1 Selon le flux

Dans cette classification on distingue 2 type de codage réseau [8].

- **Intra-session** : Le codage doit être fait stratégiquement afin de garantir que chaque récepteur peut décoder les paquets sources désirés, les nœuds ne peuvent pas combiner toutes leurs entrées aléatoirement car ils peuvent ne pas avoir la capacité d'entrée suffisante pour décoder tous les processus sources combinés. Ce type de codage est appelé codage de réseau intra-session, puisqu'il consiste principalement à coder les données d'un même flux entre elles en les combinant.

- **Inter-session** : Il consiste principalement à coder les données de différents flux, ce codage opère au niveau des nœuds intermédiaires du réseau. Il réduit la surcharge du réseau, il est utilisé principalement dans les réseaux sans fils.

II-3-2 Selon la couche OSI

- **Couche application** : Il est prouvé que le codage réseau peut atteindre une capacité de multidiffusion et donc améliorer de manière significative le débit d'un réseau. ALM est un candidat idéal pour appliquer ce codage au niveau de la couche d'application pour deux raisons : premièrement, ALM est construit sur des réseaux peer-to-peer dont la topologie peut être arbitraire, il est donc facile d'adapter la topologie pour faciliter le codage réseau ; en second lieu, les nœuds dans ALM sont hôtes finaux suffisamment puissants pour effectuer un codage complexe et opérations de décodage [9].

- **Couche réseau** : Pour améliorer le débit du trafic unicast dans les réseaux multi-sauts sans fil on exploite la nature de diffusion du support à travers le codage réseau qui participe principalement au routage. Notre formulation théorique fournit une méthode pour calculer les routes source-destination et en utilisant les meilleures possibilités de codage à partir des disponibles afin de maximiser la débit [10].
- **Couche Physique** : Le concept de codage réseau physique PNC a été proposé en 2006 pour une application dans les réseaux sans fil. Depuis lors, il s'est développé en un sous-champ de codage réseau avec de larges implications. L'idée de base de la PNC est d'exploiter le mélange de signaux qui se produit naturellement lorsque des ondes électromagnétiques se superposent les unes aux autres [11]. En particulier, au niveau d'un récepteur, les transmissions simultanées de plusieurs émetteurs aboutissent à la réception d'une somme pondérée des signaux. Cette somme pondérée est une forme d'opération de codage de réseau par elle-même.

II-4 Les domaines d'application du Codage Réseau

Dans cette section, nous traitons quelques applications du codage réseau :

II-4-1 Partage de fichier

Le partage de fichier peer-to-peer ou un fichier d'une certaine taille, au niveau d'un nœud dit "serveur", est découpé en blocs, les nœuds désirant acquérir le fichier, téléchargent les blocs qui le constitue du "serveur" ou les autres nœuds sous forme de combinaisons linéaires. À ce niveau, le codage réseau permet aussi de réduire le temps de transfert et augmente la fiabilité [12].

II-4-2 Transmission Multicast

Dans la transmission des données à plusieurs récepteurs et dans le cas d'une perte des paquets, NC permet de contrôler les erreurs au niveau des récepteurs ce qui assure la fiabilité du réseau [13].

Exemple : Supposons que nous voulons transmettre de manière fiable des données sur le réseau à plusieurs récepteurs. Le problème qu'il va y avoir est que les transmissions unicast fournissent une mauvaise utilisation de la disponibilité des ressources réseaux. Si tous les récepteurs sont intéressés par le même contenu, nous pouvons efficacement utiliser le canal sans fil avec des transmissions de diffusion. Sous l'idéal conditions de canal, tous les paquets de diffusion sont livrés à tous les nœuds simultanément. Dans les réseaux sans fil de la vie réelle les pertes de paquets sont fréquentes, donc une retransmission est nécessaire pour assurer la fiabilité. Nous avons besoin de corriger les pertes de paquets liées aux récepteurs. La plus simple des solutions est que le nœud récepteur demande à la source originale tous les éléments manquants de paquets. Cela implique que chaque paquet perdu est transmis à nouveau, et si les paquets perdus qui ne sont pas les mêmes dans les différents récepteurs, la plupart des retransmissions ne sont pas utiles pour l'ensemble des récepteurs car certains d'entre eux ont déjà reçu ces paquets en premier lieu.

En utilisant le codage réseau, nous pouvons simultanément bénéficier d'une seule retransmission en

envoyant un paquet codé au lieu de choisir un paquet original spécifique. Un paquet codé porte des informations qui peuvent potentiellement corriger différentes erreurs à différents nœuds simultanément.

II-4-3 Les réseaux sans fil

Dans un environnement sans fil, le codage réseau peut être utilisé pour offrir des avantages en termes de bande passante sans fil et du délai [14] [15].

Exemple : Considérons un réseau ad-hoc sans fil où les périphériques A et C veulent échanger les fichiers binaires x_1 et x_2 en utilisant le périphérique B comme relais. Nous supposons que le temps est négligé et qu'un périphérique peut transmettre ou recevoir un fichier pendant un intervalle de temps (communication semi-duplex).

La Figure II.2 représente à gauche l'approche standard : les nœuds A et C envoient leurs fichiers au relais B, qui à son tour transmet chaque fichier à la destination correspondante. L'approche de codage de réseau tire partie de la capacité naturelle des canaux sans fil pour la radiodiffusion, à donner des avantages en termes d'utilisation des ressources, comme illustré à la Figure II.2. En particulier, le nœud C reçoit à la fois les fichiers x_1 et x_2 , et les envoie par bits pour créer le fichier $x_1 + x_2$, qu'il diffuse ensuite aux deux récepteurs en utilisant une transmission commune. Le nœud A a x_1 et peut donc décoder x_2 . Le nœud C a x_2 et peut donc décoder x_1 .

Cette approche offre des avantages en termes d'efficacité énergétique (le nœud B transmet une fois au lieu de deux fois) et de délai (la transmission est terminée après trois timeslots au lieu de quatre).

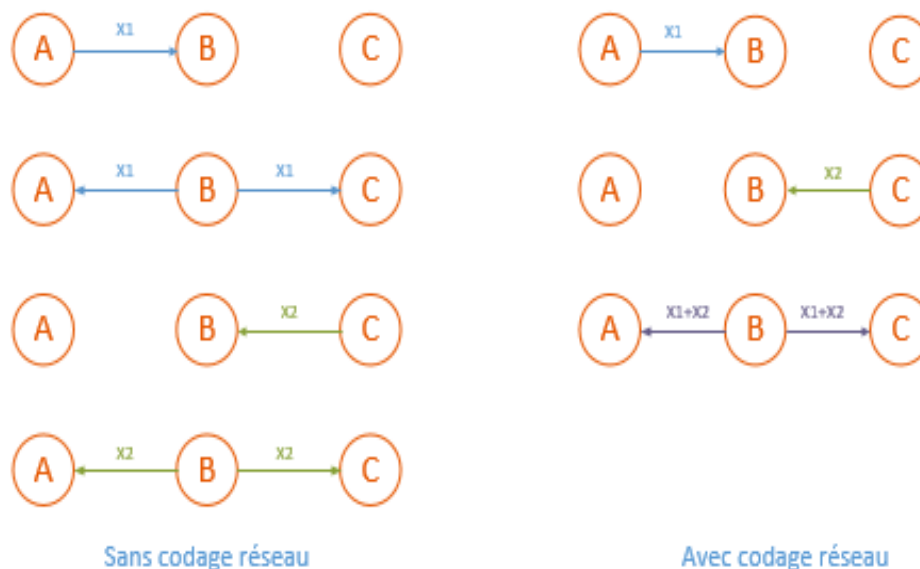


Figure II.2 – Le principe de codage réseau dans les réseaux sans fil

II-4-4 Sécurité réseau

L'envoi des combinaisons linéaires de paquets au lieu d'envoyer des données non codées est un moyen naturel de tirer parti de la diversité des trajets multiples pour la sécurité contre les attaques par écoute clandestine, ainsi les systèmes qui nécessitent seulement une protection contre de telles attaques simples, peut l'obtenir "gratuitement" sans des mécanismes supplémentaire de sécurité [16] [17]. Exemple : Considérons le noeud A qui envoie des informations au noeud D par deux chemins ABD et ACD dans la Figure II.3. Supposons qu'un adversaire (autre personne) peut intercepter un seul chemin et n'a pas accès à la voie complémentaire.

Si les symboles indépendants x_1 et x_2 sont envoyés non codés, l'adversaire peut en intercepter un. Si au contraire on a des combinaisons linéaires, des symboles sont envoyés à travers les différentes routes, l'adversaire ne peut décoder aucune partie des données (par exemple, il récupère $x_1 + x_2$, la probabilité de deviner correctement x_1 est égal à 50%, idem pour x_2).

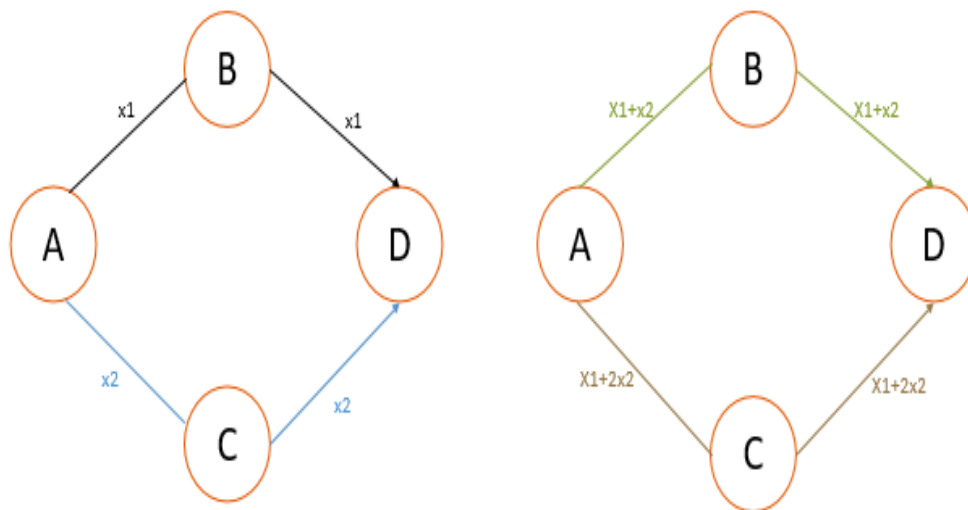


Figure II.3 – Exemple de la sécurité réseau en utilisant du codage réseau

II-5 Exemples de codage réseau

De nos jours, NC a beaucoup évolué depuis sa formalisation et il existe différentes implémentations du concept mathématique parmi eux le codage réseau XOR :

II-5-1 Pourquoi le Codage réseau XOR

XOR est un code qui permet d'améliorer les transmissions dans différentes implémentations sur différentes couches du réseau. À des fins d'explication, cette section décrit pourquoi une implémentation du codage au niveau de la couche TCP/IP est nécessaire.

Les communications point à point fonctionnent selon des règles très strictes où des protocoles tels que TCP/IP est utilisé pour transmettre efficacement des informations. Les communications effectuées dans le cadre de ces protocoles sont basées sur des paquets ou des fragments. Dans une communication TCP, les paquets sont transmis séquentiellement à partir d'un nœud source et seulement acceptés à la destination s'ils arrivent dans l'ordre. Chaque paquet est reconnu par le destinataire. Si, toutefois, l'expéditeur ne reçoit pas l'accusé de réception dans un délai raisonnable ou s'il reçoit l'accusé de réception d'un paquet ultérieur avant de recevoir celui qu'il attend, il considère que le paquet non acquitté est perdu et l'envoie à nouveau. L'expéditeur peut envoyer le paquet n°83 lorsqu'il apprend que le paquet n°21 n'est pas arrivé. Il renvoie ensuite le paquet n°21 et tous les paquets suivants.

Dans les transmissions TCP, XOR améliore l'efficacité de la communication en utilisant la combinaison des informations de plusieurs paquets en un seul paquet. Le même nombre de paquets doit être envoyé et reçu, mais les paquets sont maintenant interchangeables, de sorte que si le paquet n°21 n'arrive pas, le récepteur peut utiliser le paquet n°22 comme substitut, évitant ainsi de devoir renvoyer les paquets n°22 à n°83. Cette polyvalence des paquets de XOR réduit considérablement les réémissions de paquets et la signalisation associée nécessaire pour suivre les «paquets perdus».

II-5-2 Principe de fonctionnement général

En mode XOR, les nœuds sources transmettent (n) paquets non codés d'origine. Les nœuds intermédiaires effectuent le codage en appliquant un XOR logique sur les paquets entrants (à partir d'une source ou d'un flux unique ou multiple) [18]. Le décodage est effectué sur les nœuds de destination. Afin de décoder, ces nœuds doivent préalablement recevoir et stocker (n - 1) paquets d'origine du paquet codé.

Pour illustrer le concept, considérons un seul nœud source S et deux nœuds de destination X et Y. Les nœuds intermédiaires C, D, E et V ne font que multidiffuser les paquets entrants vers les ports restants. Le nœud source S a deux paquets, A et B, qui doivent être multidiffusés vers X et Y, au débit le plus élevé possible. De plus, chaque bord dirigé représente un lien réseau et ne peut transmettre qu'un seul paquet par unité de temps. D'après la Figure II.4 (a), si aucun codage n'est autorisé, il y a un "goulot d'étranglement" au nœud V.

Soit le paquet A ou B est transmis à travers le lien sortant. Pour délivrer les deux paquets A et B, aux deux destinations X et Y, le nœud V nécessite deux unités de temps. La première pour transmettre le paquet A, la seconde pour transmettre le paquet B, ou vice-versa. Cette solution de routage possible est illustrée à la Figure II.4 (b) et (c).

D'après la Figure II.4 (d), si le codage est autorisé, il peut alors effectuer un simple OU exclusif sur les paquets A et B et transmettre le résultat. Ceci permet de surmonter la congestion du nœud V, puisque X obtient A et reconstruit B à partir de $A \oplus (A \oplus B) = B$ et Y obtient B et A = $B \oplus (A \oplus B)$, en une seule unité de temps.

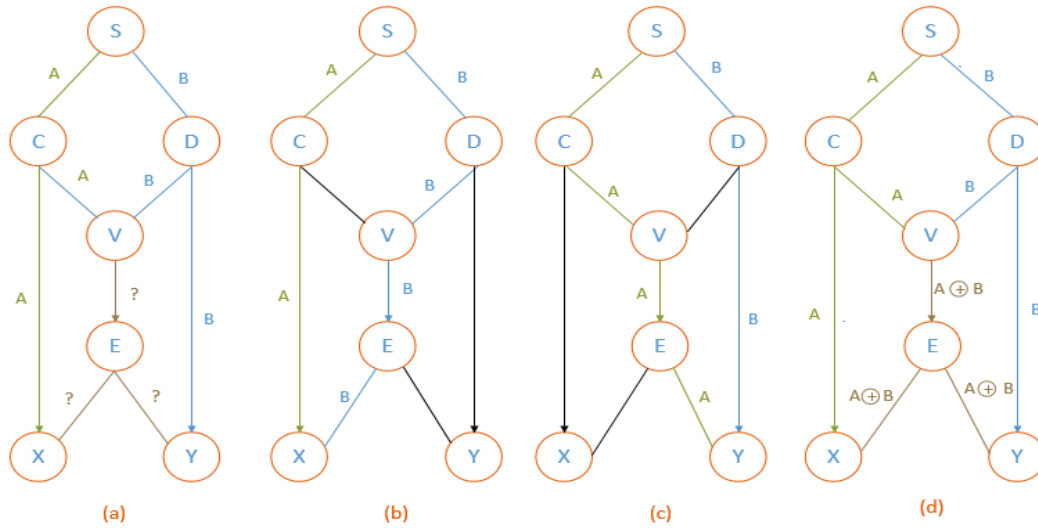


Figure II.4 – Schéma d'un réseau Butterfly

II-6 Conclusion

Ce chapitre est le résultat d'une recherche théorique sur le Codage Réseau. Cette nouvelle technologie permet d'augmenter la vitesse de la communication réseau et de réduire également la latence en introduisant le mécanisme de calcul et retransmission, au lieu du mécanisme de stockage et retransmission sur les périphériques réseau. Nous avons listé les différentes classifications et domaine d'application et enfin nous avons abordé la stratégie de codage OU exclusive que nous allons implémenter dans notre solution.

Chapitre III

Conception de la solution

III-1 Introduction

Ce chapitre se concentre sur la mise en œuvre du codage réseau. Nous proposons un SDNC qui est un codage réseau complet dans une architecture SDN. Nous présentons les détails de cette implémentation avec les critères sur lesquels nous nous sommes basées dans le choix des modules à installer. Nous mettons en avant l'algorithme "xor_redundancy" utilisé ainsi qu'une schématisation de l'ensemble des scripts et les fonctions qui contribuent au bon fonctionnement de la solution proposée.

III-2 Architecture de la solution

Notre solution repose sur un réseau de virtualisation distribué, ces réseaux sont sensibles à la charge, aux pertes et aux latances car ils génèrent un trafic important. Pour remédier à ces problèmes, nous avons intégré un mécanisme de codage réseau dans une topologie SDN à saut unique au niveau des noeuds finaux (la Figure III.1 schématise cette architecture).

Dans le déploiement de cette solution, plusieurs composants et outils ont été nécessaire à installer et à configurer :

- Nous avons opté pour une virtualisation des noeuds source et destination.
- Nous avons eu recours à un gestionnaire de machine virtuelle qui est un programme qui permet à plusieurs systèmes d'exploitation de partager un seul hôte matériel (une seule machine physique) qui est l'hyperviseur KVM.
- Nous avons installé un contrôleur adéquat, ainsi qu'un commutateur multicouches virtualisé distribué OVS compatible avec OpenFlow.

Une telle conception permettra de minimiser le trafic réseau et d'assurer la réception des flux.

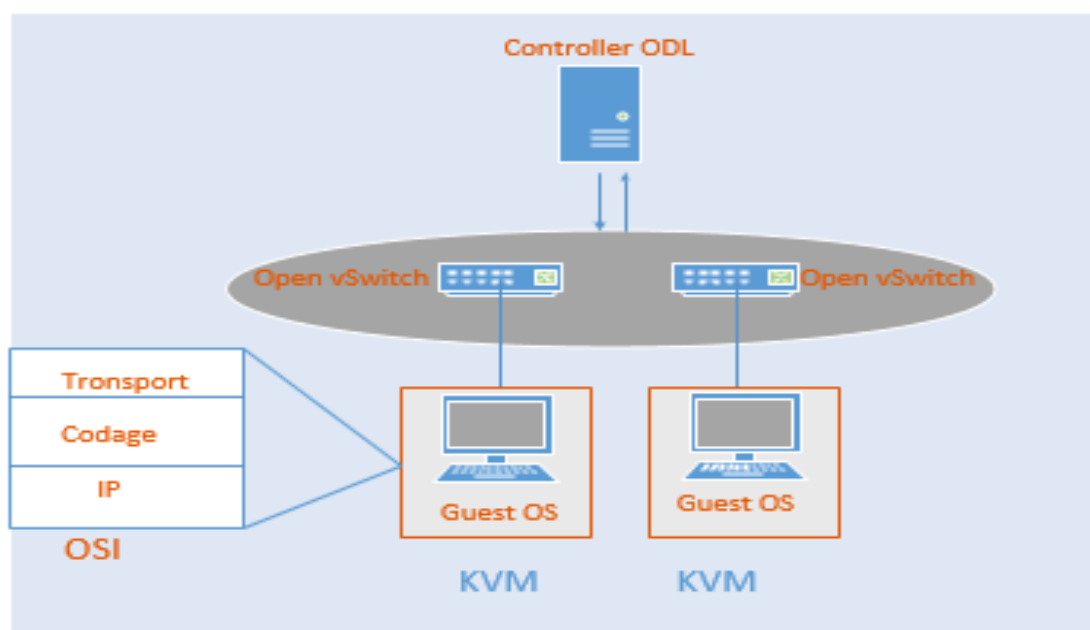


Figure III.1 – Architecture de la solution

III-2-1 L'hyperviseur KVM

KVM est le module qui transforme le noyau Linux en Hyperviseur, il est intégré dans le noyau Linux depuis la version 2.6.20. Il fonctionne originellement sur les processeurs à architectures x86 disposant des instructions de Virtualisation Intel VT ou AMD-V.

La Figure III.2 est un cas de figure de deux hôtes invités installés sur une machine physique (Host OS), qui embarque le logiciel hyperviseur qui interprète les pilotes de périphériques virtuels (Guest OS) et offre l'accès au processeur (CPU) et à la mémoire RAM aux différents hôtes invités.

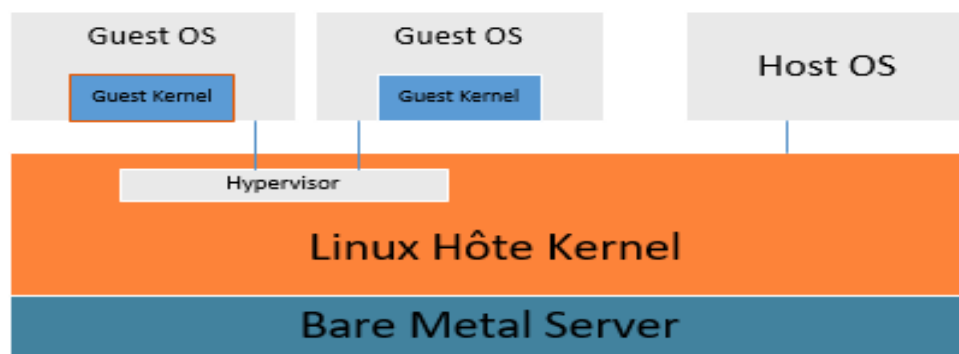


Figure III.2 – Position de l'hyperviseur dans une plateforme Linux

III-2-2 Le contrôleur

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

OpenDylight

« Il y a plusieurs contrôleurs Open Source sur le marché mais seuls deux sont actifs aujourd'hui : OpenDaylight et Open Networking Operating System (Onos) », indique Curt Beckmann (Président fondateur du groupe de travail Abstractions (FAWG) de l'ONF-mars-2015) [19][20].

Nous avons choisi Opendaylight car il prend en charge la programmabilité du réseau via plusieurs protocoles orientés vers le sud comme OVSDB et openflow, de plus il est peu volumineux et peu gourmand en performances. L'architecture OpenDaylight se compose de trois couches [21] :

1. Plugins et protocoles Southbound.
2. Adaptations de service et fonctions de réseau.
3. APIs et applications Northbound.

Le tableau III.1 souligne les spécifications des différents contrôleurs.

	ONOS	ODL	NOX	POX	RYU
langage de programmation	Java	Java	C++	Python	Python
GUI	Web Based	Web Based	Python+QT4	Python+QT4	Yes
Documentation	Bien	Très Bien	Mauvaise	Mauvaise	Moyenne
Modularité	Élevé	Élevé	Faible	Faible	Moyenne
Distribution / Centralisation	D	D	C	C	C
Plateforme	Linux,MacoS,Win	Linux,MacoS,Win	Linux	Linux,MacoS,Win	Linux
Productivité	Normal	Normal	Haute	Haute	haute
SBI	OF1.0 , 1.3 Netconf	OF 1.0, 1.3,1.4 Netconf/Yang ovsdb bgp snmp	OF1.0	OF1.0	OF1.0,1.2, 1.3, 1.4 Netconf/ ofconfig
NBI	REST API	REST API	REST API	REST API	REST API

Table III.1 – Tableau de comparaison de quelques contrôleurs les plus connus

III-2-3 OpenVswitch

Afin de basculer le trafic entre différentes machines virtuelles (VM) sur le même hôte physique dans une architecture SDN, nous avons besoin d'un commutateur Openvswitch [RFC 7047] qui est un commutateur logiciel open-source virtuel intégré dans le noyau Linux. Nous avons opté pour l'intégration d'OpenVswitch dans notre solution car [22] :

- Il fournit deux protocoles ouverts qui sont spécialement conçus pour la gestion à distance dans un réseau à environnement virtualisé qui sont : OpenFlow, qui expose l'état de transfert basé sur le flux et le protocole de gestion OVSDB : l'interface qui est utilisée pour effectuer la gestion et les opérations de configuration sur l'instance OVS, qui expose l'état du port du commutateur [23].
- OpenVswitch nous permet de gérer et assister le trafic entre deux machines virtuelles dans le même hôte.
- Il peut actuellement fonctionner sur n'importe quelle virtualisation sur la plate-forme Linux, y compris KVM, VirtualBox, Xen ainsi que la majeure partie du code est écrite en plate-forme C et est facilement portée vers d'autres environnements.

OpenVswitch est composé de la partie noyau et la partie utilisateur qui est composée :

- D'un démon de configuration qui lit régulièrement sa base de configuration ovsdb-server.
- D'un démon qui contrôle et gère les switchs virtuels ovs-vswitchd.

- D'une partie utilisateur qui fournit les commandes systèmes (ovs-*) pour interagir avec tout ça.

Le schéma de la Figure III.3 représente le fonctionnement interne utilisé par l'hyperviseur de notre implémentation.

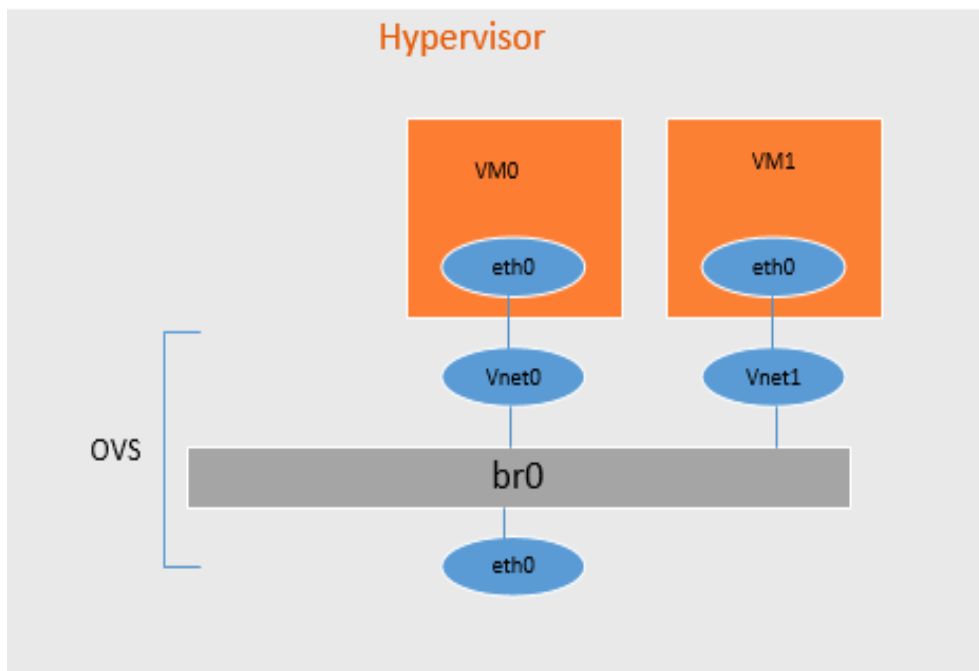


Figure III.3 – Fonctionnement interne d'Openvswitch

Le switch virtuel (appelé aussi bridge) sera nommé "br0". L'interface réseau physique "eth0" sera intégralement gérée par OpenVswitch. Elle sera donc considérée comme un port de ce switch. Ici c'est la grande force d'OpenVswitch d'utiliser directement des trames ethernet pour les échanges. Ainsi quand le trafic sort par une carte réseau physique, c'est transparent. Cela permet de relier facilement la partie réseau virtuel au monde physique.

III-3 Le codage appliqué

Dans cette partie, nous détaillerons la manière dont opère le codage mis en place dans la solution. Nous avons opté pour la mise en place d'une stratégie de codage XOR. Cette stratégie consiste en un codage au niveau des noeuds sources. Le codage est implémenté sur les noeuds, entre les couches de transport et réseau du modèle OSI, afin qu'il agisse de façon transparente vis-à-vis des couches supérieures. Ceci se fera grâce à une interface virtuelle TUN dont nous détaillerons les caractéristiques et le fonctionnement plus loin dans cette section.

Les objectifs principaux de cette approche seront de minimiser la charge et d'assurer une haute tolérance aux pertes

III-3-1 Fonctionnement général

Dans les communications TCP/IP si le premier paquet de données envoyé P_1 est perdu ou délivré trop lentement, aucun accusé de réception n'est reçu dans le temps et le paquet P_1 est à nouveau transmis.

Ce processus se répète jusqu'à ce que le paquet P_1 soit reçu et reconnu.

Une alternative XOR du même protocole calcule les "paquets codés" P_1 et P_2 et envoie ces derniers à la place des "paquets de données" non codés, ce processus se répète 3 ou 5 fois selon le taux de redondance appliqué dans notre implémentation. Dans le cas d'une redondance de 3 paquets nous avons :

$$\begin{aligned} \text{paquet codé 1} &= P_1 + P_2 \\ \text{paquet codé 2} &= P_1 + P_2 + P_3 \\ \text{paquet codé 3} &= P_2 + P_3 + P_4 \\ \text{paquet codé 4} &= P_3 + P_4 + P_5 \end{aligned} \quad (\text{III.1})$$

Si le récepteur reçoit le paquet de P_1 et le paquet codé 1, il peut facilement obtenir la paquet P_2 . Le schéma codé est similaire au schéma standard quand aucun paquet n'est perdu. Un avantage du codage devient évident sous la perte de paquets introduite.

III-3-2 Structure du paquet

La structure du paquet englobe plusieurs champs d'informations. Certains champs contiennent des informations de contrôle (l'en-tête) et d'autres contiennent le message à transmettre (charge utile).

Chaque paquet possède 5 champs comme le présente la Figure III.4.

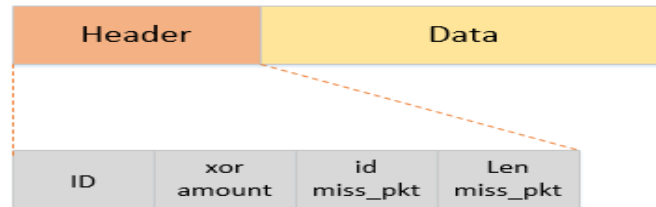


Figure III.4 – Structure du paquet xor

- ID : identifie le packet, l'identifiant du paquet est composé de l'identifiant du noeud et le numéro de la séquence. Par exemple, lorsqu'un paquet est produit par le noeud source n_1 son identifiant sera " n_1x " où x est le numéro du paquet généré par le noeud.
- xor_amount : est le taux de redondance.
- id_miss_pkt : est l'identificateur du paquet perdu. Cette valeur indique si le paquet contient des informations d'un paquet original (s'il est égale à 0) ou bien d'un paquet de redondance (pour toute valeur supérieur à 0).
- len_miss_pkt : est la taille du paquet perdu (0 si c'est un paquet original et non pas un packet de redondance).
- Data : est l'information utile.

Dans le processus de decodage, $\langle \text{id_miss_pkt}, \text{len_miss_pkt} \rangle$ sera crucial pour connaître l'information du paquet perdu.

III-3-3 Exemple de Codage Réseau basé sur XOR

XOR est utile dans de nombreux scénarios différents, par exemple dans la communication point à point car il peut éliminer les pertes de paquets sur un lien avec perte. La Figure III.5 est un exemple d'une transmission d'un noeud à un autre. Dans l'exemple, le lien a une probabilité de perte de paquets de 10% et le codeur ajoute une redondance de 3 pour compenser les pertes.

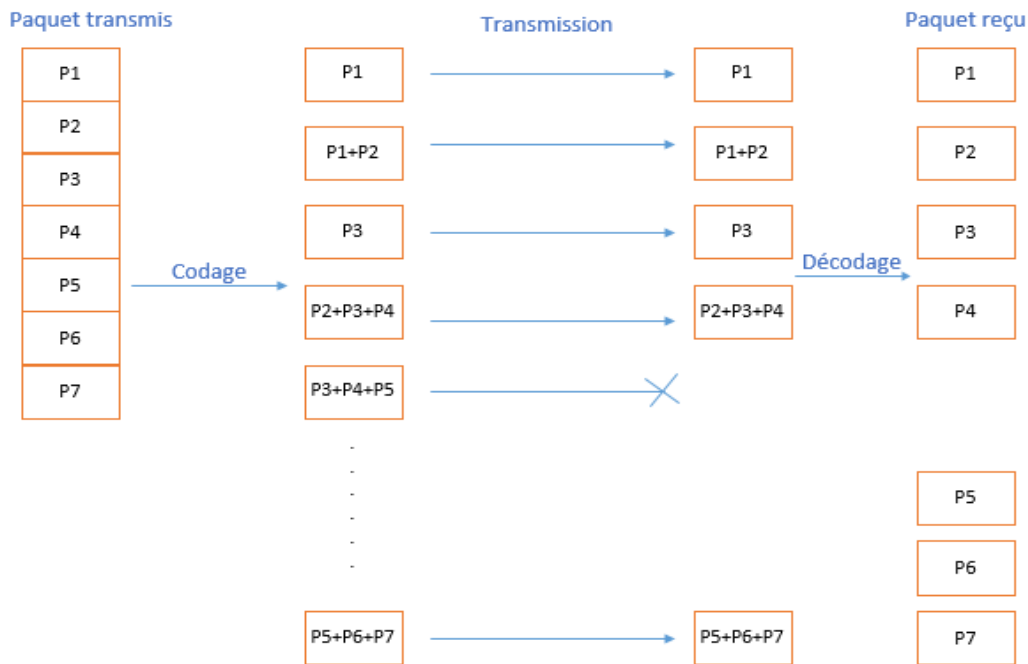


Figure III.5 – Exemple de codage/décodage XOR

III-3-4 Interface Tun

Dans notre implémentation, la communication entre le réseau du système hôte et les machines virtuelles nécessite l'utilisation des interfaces TUN/TAP du noyau Linux.

TUN (à savoir TUNnel réseau) simule un périphérique de couche réseau et il fonctionne avec les paquets de couche 3 comme les paquets IP. TAP simule un dispositif de couche liaison et fonctionne avec des paquets de couche 2 comme des trames Ethernet. TUN est utilisé avec le routage, tandis que TAP est utilisé pour créer un pont réseau [24].

Nous avons implémenté les interfaces TUNs au niveau des noeuds finaux, ces interfaces représentent les couches de codage qui forment un tunnel entre elles et échangent les paquets en utilisant UDP. Nous avons fixé le MTU de l'interface à 1350 octets, afin de le garder bien au-dessous du MTU de l'interface réseau normale (1500 octets) pour éviter la fragmentation des paquets au niveau de la couche IP.

L'interface TUN est configurée avec une adresse ip, quand une application est exécutée au niveau du noeud source un socket s'ouvre et les paquets sont transmis à cette adresse, le système (OS) trans-

mettra les paquets à la couche de codage, la couche de codage va ensuite traiter les paquets et les transmet via l'interface réseau normale à l'autre bout du tunnel.

À la reception, les paquets sont reçus par le nœud sur l'interface réseau et puis ils sont transmis à la couche de codage afin de décoder les paquets reçus.

III-3-5 Diagramme schématisant la structure des applications

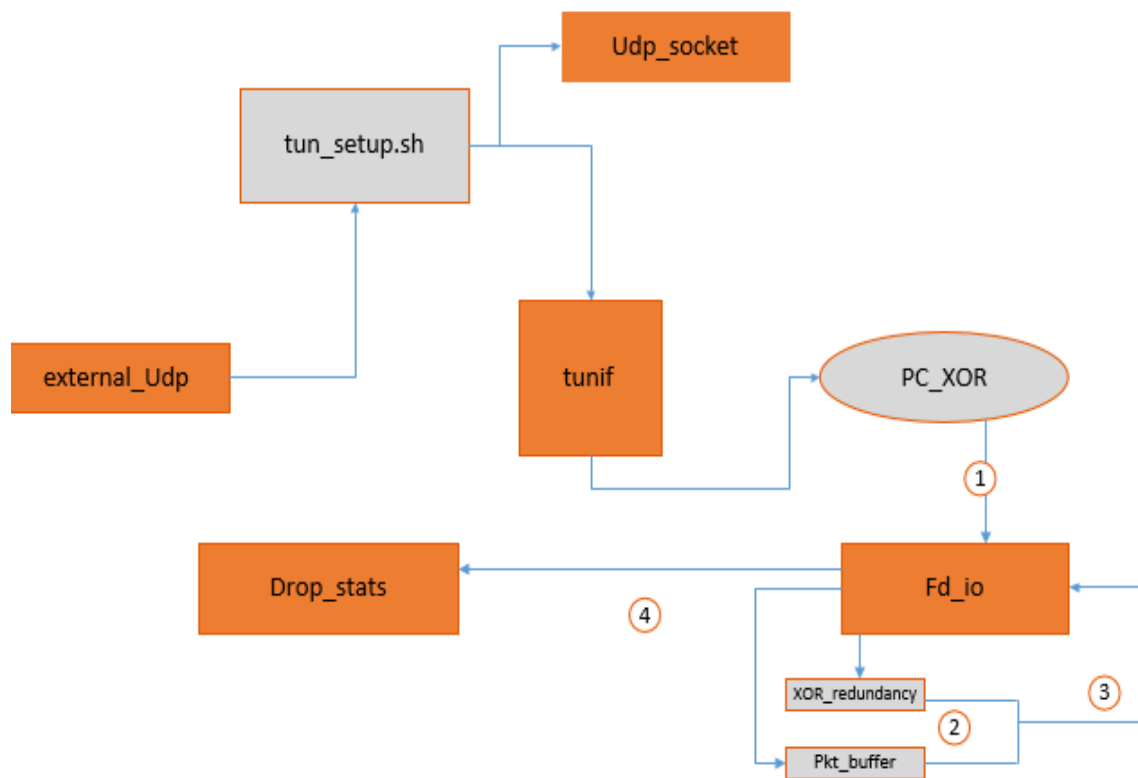


Figure III.6 – Structure logicielle des applications implémentées

Nous avons eu recours à plusieurs scripts et fonctions afin de mettre en oeuvre la stratégie de codage réseau "XOR_Redundancy" :

- External_udp : ce script gère les échanges entre l'interface physique et les interfaces virtuelles des nœuds finaux.
- Tun_setup : ce script nous permet de créer l'interface tun et de lui associer un nom et une adresse.
- Udp_socket : permet la création d'un tunnel entre les deux nœuds virtuelles.
- Tunif : associe l'interface tun créée précédemment à l'application de codage/décodage.
- Pc_xor : ce script permet de lancer l'application du codage en spécifiant le nom de l'interface Tun, le taux de redondance, le nom du fichier qui contient les états de codage/décodage générés par le script "Drop_stats" afin de mesurer les performances et le taux de récupération des paquets. Ce processus de codage se fait à l'aide de :
 - Fd_io : transfert des paquets et gestion des entrées/sorties vers la couche de codage.

- Xor_fixed_redundancy : les fonctions de base de ce script sont :
 - Read_pkt : génération de l'ID du packet, sa taille et son type.
 - Write_pkt : cette fonction permet de combiner le packet original avec le nouveau packet de redondance en spécifiant son ID et sa taille en utilisant "new_xor_info.push_back(id,len)".

III-4 Conclusion

Dans ce chapitre, nous avons proposé une nouvelle méthode d'intégration du codage réseau dans l'architecture SDN en utilisant le "Xor_Redundancy". Nous avons décrit l'ensemble des modules nécessaires pour une mise en œuvre conjointe de ces deux concepts afin de fournir une utilisation de la stratégie SDN-NC qui est transparente aux protocoles de bout en bout.

Chapitre IV

Simulation et résultats

IV-1 Introduction

Afin de mettre en oeuvre notre solution proposée dans le chapitre précédent, nous avons dédié ce chapitre aux détails de l'intégration concrète de la stratégie du codage réseau. Nous commençons par une mise en place de l'environnement d'évaluation de notre solution et les outils utilisés, puis nous présentons le script shell qui configure et crée la topologie de façon automatique. Nous terminons ce chapitre par une analyse des résultats en se basant sur différentes métriques de performance.

IV-2 Environnement d'évaluation

IV-2-1 Instalation de KVM

- Installation des outils KVM

Avant de procéder à l'installation, nous confirmons que notre système prend en charge KVM. Comme mentionné précédemment, il nécessite un support de virtualisation matérielle dans la CPU. La Figure IV.1 montre que le système est configuré et prêt pour l'installation de KVM.

```
achaibou@RADIA-C:~$ egrep -c '(vmx|svm)' /proc/cpuinfo
4
```

Figure IV.1 – Vérification de la virtualisation au niveau de la CPU

L'installation de KVM nécessite un processeur de 64 bits, pour voir si notre processeur est 64 bits, nous exécutons la commande de la Figure IV.2.

```
achaibou@RADIA-C:~$ egrep -c 'lm' /proc/cpuinfo
4
```

Figure IV.2 – Vérification de la version du processeur 64/32 bits

Nous installons un multiplexeur de terminaux : Tmux qui nous permet d'exploiter plusieurs terminaux au sein d'un seul et même affichage à l'aide de la commande suivante :

```
achaibou@RADIA-C:~$ sudo apt-get install tmux
```

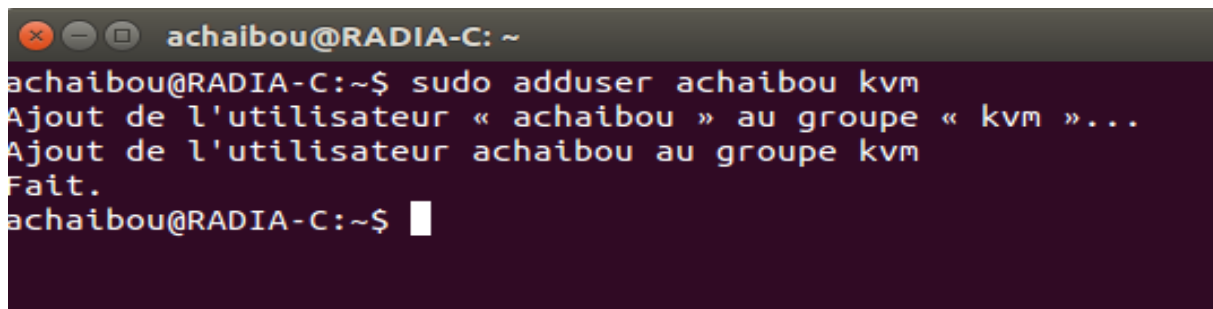
Figure IV.3 – Installation tmux

Vérifions que notre kvm peut être utilisé en exécutant la commande suivante :

```
achaibou@RADIA-C:~$ sudo kvm-ok
INFO: /dev/kvm exists
KVM acceleration can be used
```

Figure IV.4 – vérification de l'installation de kvm au niveau du noyau Linux

Nous ajoutons le nom d'utilisateur Ubuntu dans le groupe kvm en tapant :



```
achaibou@RADIA-C: ~  
achaibou@RADIA-C:~$ sudo adduser achaibou kvm  
Ajout de l'utilisateur « achaibou » au groupe « kvm »...  
Ajout de l'utilisateur achaibou au groupe kvm  
Fait.  
achaibou@RADIA-C:~$
```

Figure IV.5 – Ajout d'un utilisateur dans le groupe KVM

Après cela, nous devons se reconnecter pour que le nom d'utilisateur devient un membre effectif du groupe KVM. Les membres de ce groupe peuvent exécuter des machines virtuelles.

IV-2-2 Installation du contrôleur

Nous avons installé le contrôleur OpenDaylight sur une machine ubuntu 14.04 LTS. Ce contrôleur est programmé en Java. Nous avons installé l'environnement d'exécution Java à l'aide des commandes suivantes :

```
$ sudo apt-get update
```

```
$ sudo apt-get install default-jre-headless
```

Définir la variable d'environnement JAVA_HOME, en éditant le fichier bashrc :

```
$ nano /.bashrc
```

Ajout de la ligne suivante au fichier bashrc :

```
Export JAVA_HOME=/usr/lib/jvm/default-java
```

Ensuite, nous exécutons le fichier :

```
$ source /.bashrc
```

Téléchargement d'ODL :

```
$ wget https://nexus.opendaylight.org/content/groups/public/org/.opendaylight/integration/  
distribution-karaf/0.4.0-Beryllium/distribution-karaf-0.4.0-Beryllium.tar.gz
```

Installez OpenDaylight en extrayant le fichier tar :

```
$ tar -xvf distribution-karaf-0.4.0-Beryllium.tar.gz
```

Pour lancer OpenDaylight, nous exécutons la commande karaf dans le dossier de distribution du package. La Figure IV.6 illustre le résultat obtenu.



Pour compiler les programmes de l'espace utilisateur dans la distribution Open vSwitch, nous avons besoin des logiciels suivants :

- Nous clonons dans un répertoire nommé "ovs" le code source Open vSwitch et son référentiel Git, avec la commande :

Configurons le package en exécutant le script de configuration :

```
$ ./configure
```

- **Compilation**

Exécutons "\$ make install" pour installer les exécutable et les pages de manuel dans le système en cours d'exécution, par défaut sous /usr/local.

Il est possible que nous ayons déjà installé un module Open vSwitch sur notre machine (dans un répertoire différent). Pour charger le module noyau Open vSwitch que nous avons construit à partir de ce référentiel, nous devons créer un fichier "depmod.d" qui préfère les modules noyau nouvellement installés sur les modules noyau de Linux.

Procédons au chargement des modules du noyau dont nous avons besoin avec la commande :

```
$ /sbin/modprobe openvswitch
```

- **Démarrage**

Sur les systèmes Unix, le démarrage de la suite de démons Open vSwitch est un processus simple. Open vSwitch inclut un script shell, et des helpers, appelés ovs-ctl qui automatisent la plupart des tâches de démarrage et d'arrêt de ovssdb-server, et ovs-vswitchd.

Après l'installation, les démons peuvent être démarrés en utilisant l'utilitaire ovs-ctl. Cela prendra soin de configurer les conditions initiales et de démarrer les démons dans le bon ordre.

En plus d'utiliser le script automatisé pour démarrer Open vSwitch, nous pouvons démarrer manuellement les différents démons. Avant de démarrer ovs-vswitchd lui-même, nous devons démarrer sa base de données de configuration, ovssdb-server comme illustré dans la Figure IV.7.

```
root@RADIA-C:/home/achaibou/ovs# export PATH=$PATH:/usr/local/share/openvswitch/scripts
root@RADIA-C:/home/achaibou/ovs# ovs-ctl start
* Starting ovssdb-server
* system ID not configured, please use --system-id
* Configuring Open vSwitch system IDs
* Inserting openvswitch module
* Starting ovs-vswitchd
* Enabling remote OVSSDB managers
root@RADIA-C:/home/achaibou/ovs# mkdir -p /usr/local/var/run/openvswitch
root@RADIA-C:/home/achaibou/ovs# ovssdb-server --remote=punix:/usr/local/var/run/openvswitch/db.sock \
> --remote=db:Open_vSwitch,Open_vSwitch,manager_options \
> --private-key=db:Open_vSwitch,SSL,private_key \
> --certificate=db:Open_vSwitch,SSL,certificate \
> --bootstrap-ca-cert=db:Open_vSwitch,SSL,ca_cert \
> --pidfile --detach --log-file
2018-04-08T23:35:48Z|00001|vlog|INFO|opened log file /usr/local/var/log/openvswitch/ovssdb-server.log
ovssdb-server: /usr/local/var/run/openvswitch/ovssdb-server.pid: already running as pid 2873, aborting
root@RADIA-C:/home/achaibou/ovs# ovs-vsctl --no-wait init
```

Figure IV.7 – Démarrage d'OpenVswitch et d'OVSSDB

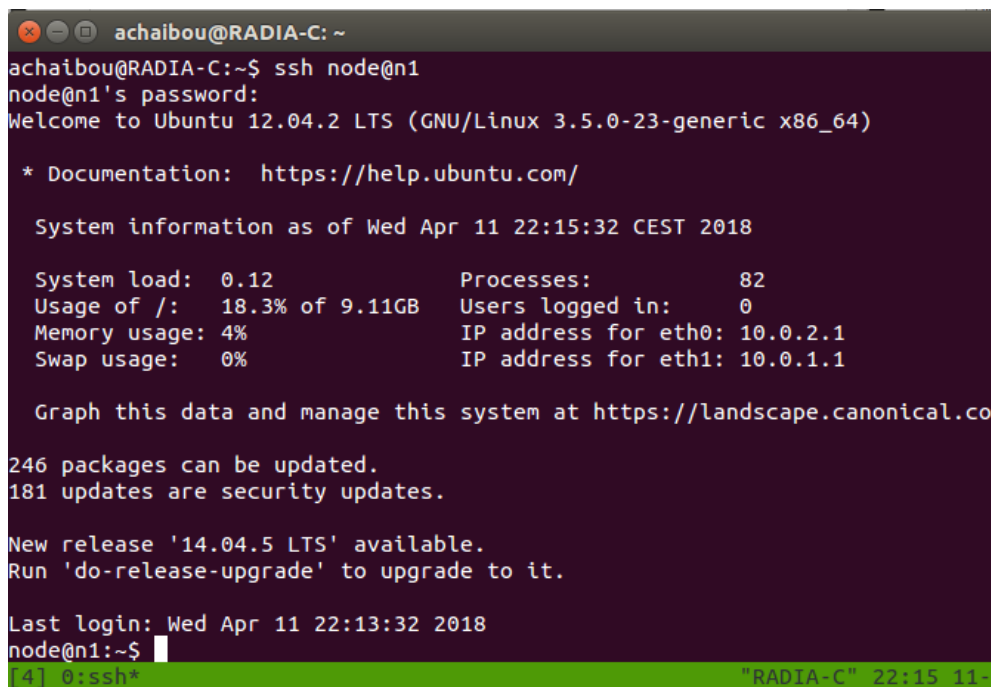
- **Validation**

À ce stade, nous pouvons utiliser ovs-vsctl pour configurer les ponts et d'autres fonctionnalités d'Open vSwitch. Par exemple, pour créer un pont nommé br0 et y ajouter les ports eth0 et vif1.0 :

```
$ ovs-vsctl add-br br0
$ ovs-vsctl add-port br0 eth0
$ ovs-vsctl add-port br0 vif1.0
```

Pour notre solution nous avons installé une image ubuntu 12.04 LTS, 64 bits qui jouera le rôle d'un hyperviseur, nous rajoutons dans le fichier /etc/hosts de la machine physique le nom et les adresses ip des deux noeuds source et destination n_2 et n_1 respectivement.

Après cette étape, l'hôte devrait être capable d'accéder au noeud n_1 et n_2 via ssh à partir d'une fenêtre Tmux avec la commande : "ssh node@n1" ou "ssh node@n2". La Figure IV.8 montre le résultat obtenu.



```
achaibou@RADIA-C: ~
achaibou@RADIA-C:~$ ssh node@n1
node@n1's password:
Welcome to Ubuntu 12.04.2 LTS (GNU/Linux 3.5.0-23-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

System information as of Wed Apr 11 22:15:32 CEST 2018

System load:  0.12               Processes:           82
Usage of /:   18.3% of 9.11GB    Users logged in:    0
Memory usage: 4%                IP address for eth0: 10.0.2.1
Swap usage:  0%                IP address for eth1: 10.0.1.1

Graph this data and manage this system at https://landscape.canonical.com

246 packages can be updated.
181 updates are security updates.

New release '14.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Wed Apr 11 22:13:32 2018
node@n1:~$
```

Figure IV.8 – Terminal d'une VM

IV-3 Présentation de la topologie implémentée

Dans cette section, nous présentons et expliquons le script shell de notre topologie.

Nous avons choisi une topologie à saut unique qui consiste en deux nœuds virtuels, chacun connecté à des commutateurs virtuels individuels. Ensuite, ces commutateurs sont connectés avec une connexion Ethernet virtuelle, sur laquelle le retard et les pertes sont introduits. La Figure IV.9 représente cette topologie.

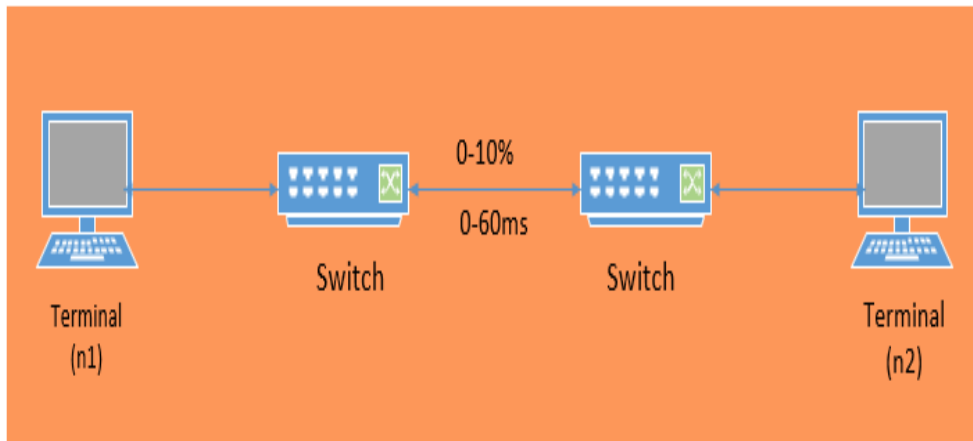


Figure IV.9 – Topologie de transmission à saut unique

IV-3-1 Explication du script de la topologie

Les étapes suivies lors de l'écriture du script **topology.sh** qui fait référence à notre topologie à un seul saut :

1. Préparation de l'environnement.
 - Définir l'image ubuntu qui sert à virtualiser les noeuds finaux.
 - Spécifier le répertoire de stockage des scripts au niveau des noeuds finaux.
 - Exiger l'exécution du script depuis une fenêtre Tmux.

La Figure IV.10 représente cette partie de script.

```
#!/bin/bash
sub=10.0.2.0/24
#wan=wlan0
img=ubuntu64-1.img
share=$PWD/../../
wan=eth0
if [ $TERM != "screen" ]; then
    echo Don't we want to be inside a screen?
    exit 1
fi
```

Figure IV.10 – Préparation de l'environnement

2. Configuration de l'interconnexion.
 - Configuration de l'interconnexion NAT.
 - Création de l'interconnexion entre les VMs et l'hyperviseur.

La Figure IV.11 représente cette partie de script.

```
sudo ip_forward 1
sudo iptables -t nat -A POSTROUTING -o $wan -j MASQUERADE
for j in $(seq 1 2); do
sudo ip tuntap add tap$j mode tap
sudo ip address add dev tap$j 10.0.2.10$j/24
sudo ip link set dev tap$j up
sudo ip route add to 10.0.2.$j dev tap$j
sudo iptables -A FORWARD -i tap$j -j ACCEPT
sudo ip tuntap add of_tap$j mode tap
```

Figure IV.11 – Configuration des interconnexions

3. Configuration des équipements.

- Configuration des équipements intermédiaires ovs.
- Connexion des noeuds finaux aux équipements intermédiaires.
- Création des images virtuelles.

La Figure IV.12 représente cette partie de script.

```
sudo ovs-vsctl add-br br$j
sudo ovs-vsctl set bridge br$j other-config:hwaddr=fe:fe:00:0a:ff:0$j
protocols=OpenFlow10,OpenFlow11,OpenFlow12,OpenFlow13
sudo ovs-vsctl add-port br$j of_tap$j
sudo ip link set dev of_tap$j up
sudo ip link set dev br$j up
qemu-img create -f qcow2 -b $img num$.img
```

Figure IV.12 – Configuration des équipements

4. Création des interfaces virtuelles.

- création des interfaces virtuelle entre les périphériques intermédiaires.

La Figure IV.13 représente cette partie de script.


```

sudo ip link add veth1 type veth peer name veth2
sudo ip link set veth1 up
sudo ip link set veth2 up
sudo ovs-vsctl add-port br1 veth1
sudo ovs-vsctl add-port br2 veth2
echo Opening KVM

```

Figure IV.13 – Création des interfaces virtuelles

5. Démarrage des VMs qui représente les noeuds finaux.

- Définir un type de réseau spécifique et une carte d'interface réseau pour les VM.
- Assigner une adresse mac à chaque noeud final.

La Figure IV.14 représente cette partie du script.

```

tmux new-window -d -t 1 "kvm \
  -smp 1 \
  -drive file=num1.img,if=virtio \
  -m 1024 \
  -nographic \
  -netdev tap,ifname=tap1,script=no,downscript=no,id=lan0 \
  -device virtio-net-pci,netdev=lan0 \
  -net nic,macaddr=fe:fe:00:0a:01:01 \
  -net tap,ifname=of_tap1,script=no \
  -fsdev local,security_model=passthrough,id=fsdev0,path=$share \
  -device virtio-9p-pci,id=fs0,fsdev=fsdev0,mount_tag=hostshare"
tmux new-window -d -t 2 "kvm \
  -smp 1 \
  -drive file=num2.img,if=virtio \
  -m 1024 \
  -nographic \
  -netdev tap,ifname=tap2,script=no,downscript=no,id=lan0 \
  -device virtio-net-pci,netdev=lan0 \
  -net nic,macaddr=fe:fe:00:0a:02:01 \
  -net tap,ifname=of_tap2,script=no \
  -fsdev local,security_model=passthrough,id=fsdev0,path=$share \
  -device virtio-9p-pci,id=fs0,fsdev=fsdev0,mount_tag=hostshare"
echo "Setting switch to listen for controller on local interface"

```

Figure IV.14 – Démarrage des VMs

6. Démarrage des OVS et configuration du partage.

- Configuration des ovs pour l'écoute du contrôleur ODL sur le port 6633.
- Configuration du partage.
- Configuration des interfaces Tun.

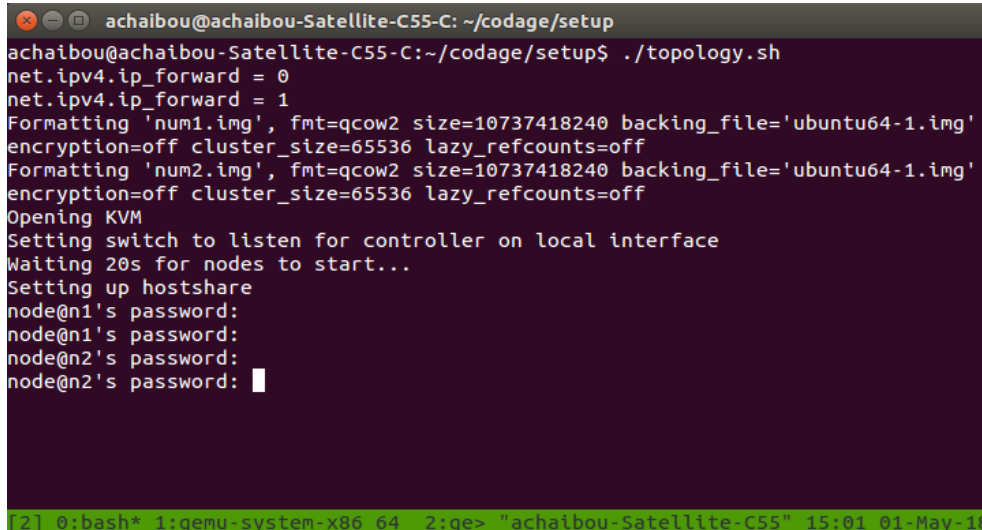
La Figure IV.15 représente cette partie de script.

```
sudo ovs-vsctl set-controller br1 tcp:0.0.0.0:6633
sudo ovs-vsctl set-controller br2 tcp:0.0.0.0:6633
echo "Waiting 20s for nodes to start..."
sleep 20
echo "Setting up hostshare"
for j in $(seq 1 2); do
ssh node@n$j "sudo mount hostshare"
ssh node@n$j "~/share/tun/scripts/tun_setup.sh ncif 10.1.1.$j"
```

Figure IV.15 – Démarrage des OVS et configuration du partage

IV-3-2 Validation de la topologie

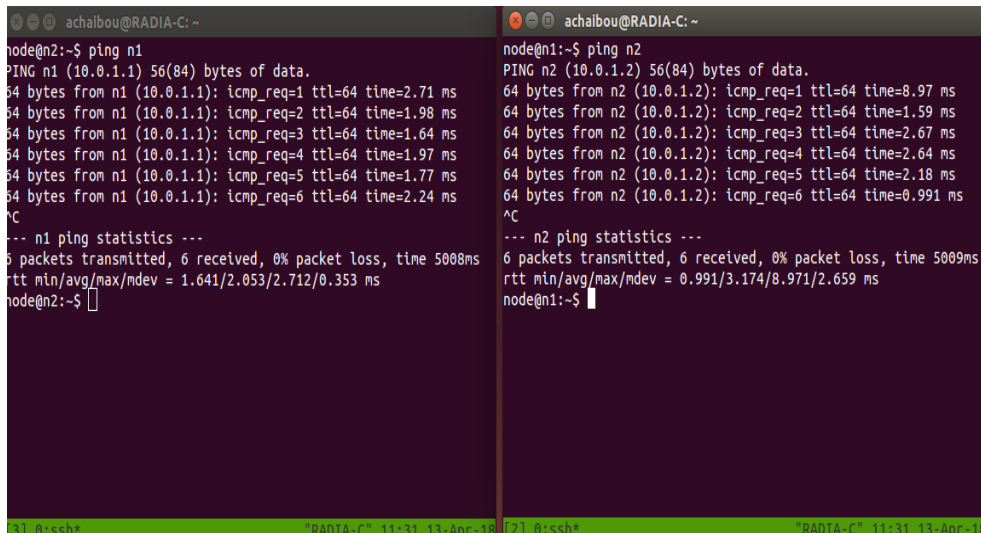
Avant d'exécuter le script "topology.sh", nous démarrons le contrôleur **OpenDayLight** et la base de donnée **OVSDb** et ensuite, nous exécutons le script "topology.sh" dans une fenêtre **tmux** comme le montre la Figure ci-après.



```
achaibou@achaibou-Satellite-C55-C: ~/codage/setup
achaibou@achaibou-Satellite-C55-C:~/codage/setup$ ./topology.sh
net.ipv4.ip_forward = 0
net.ipv4.ip_forward = 1
Formatting 'num1.img', fmt=qcow2 size=10737418240 backing_file='ubuntu64-1.img'
encryption=off cluster_size=65536 lazy_refcounts=off
Formatting 'num2.img', fmt=qcow2 size=10737418240 backing_file='ubuntu64-1.img'
encryption=off cluster_size=65536 lazy_refcounts=off
Opening KVM
Setting switch to listen for controller on local interface
Waiting 20s for nodes to start...
Setting up hostshare
node@n1's password:
node@n1's password:
node@n2's password:
node@n2's password: [ ]
```

Figure IV.16 – Lancement de la topologie à saut unique

Afin de vérifier que la liaison entre les deux machines virtuelles est correctement établie, nous lançons des pings de n_1 vers n_2 et vice versa. La Figure IV.17 affirme que la liaison entre les deux noeuds est correctement établie.



```

achalbou@RADIA-C: ~
node@n2:~$ ping n1
PING n1 (10.0.1.1) 56(84) bytes of data:
64 bytes from n1 (10.0.1.1): icmp_req=1 ttl=64 time=2.71 ms
64 bytes from n1 (10.0.1.1): icmp_req=2 ttl=64 time=1.98 ms
64 bytes from n1 (10.0.1.1): icmp_req=3 ttl=64 time=1.64 ms
64 bytes from n1 (10.0.1.1): icmp_req=4 ttl=64 time=1.97 ms
64 bytes from n1 (10.0.1.1): icmp_req=5 ttl=64 time=1.77 ms
64 bytes from n1 (10.0.1.1): icmp_req=6 ttl=64 time=2.24 ms
^C
--- n1 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5008ms
rtt min/avg/max/mdev = 1.641/2.053/2.712/0.353 ms
node@n2:~$

achalbou@RADIA-C: ~
node@n1:~$ ping n2
PING n2 (10.0.1.2) 56(84) bytes of data:
64 bytes from n2 (10.0.1.2): icmp_req=1 ttl=64 time=8.97 ms
64 bytes from n2 (10.0.1.2): icmp_req=2 ttl=64 time=1.59 ms
64 bytes from n2 (10.0.1.2): icmp_req=3 ttl=64 time=2.67 ms
64 bytes from n2 (10.0.1.2): icmp_req=4 ttl=64 time=2.64 ms
64 bytes from n2 (10.0.1.2): icmp_req=5 ttl=64 time=2.18 ms
64 bytes from n2 (10.0.1.2): icmp_req=6 ttl=64 time=0.991 ms
^C
--- n2 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5009ms
rtt min/avg/max/mdev = 0.991/3.174/8.971/2.659 ms
node@n1:~$
  
```

Figure IV.17 – Vérification de la liaison entre les deux machines virtuelles

Nous ouvrons un navigateur sur notre système hôte et nous saisissons l'adresse de bouclage 127.0.0.1 ainsi que le numéro du port propre à l'application qui est 8181, l'URL est comme suit : <http://127.0.0.1:8181/index.html>. L'interface graphique de la Figure IV.18 permet de visualiser la topologie.

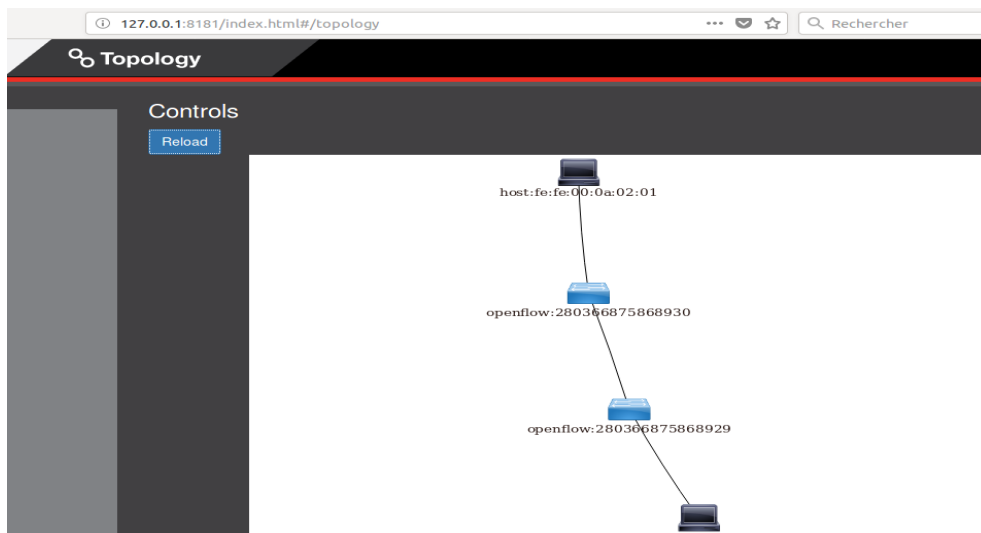


Figure IV.18 – Topologie à saut unique vu par l'interface graphique d'ODL

IV-4 Analyse des résultats de la simulation

Dans cette section, nous discutons les résultats de simulation pour le scénario proposé (topologie à saut unique). Pour chaque métrique de performance nous comparons les résultats obtenus en utilisant le codage avec ceux obtenus sans application du codage réseau en utilisant l'outil iperf.

IV-4-1 Métrique de performance

Nous considérons deux métriques pour tester les performances du codage appliqué dans un réseau SDN distribué :

- Le Débit (Mbit/s) : Une mesure de la quantité de données numériques transmises par unité de temps.
- Taux de récupération : C'est le taux effectif ou estimé de paquets reçus par le destinataire.

Nous avons utilisé ces métriques afin de montrer que la combinaison des deux techniques de routage SDN et le codage réseau est faisable et apporte plus de robustesse, d'efficacité et une meilleure récupération des paquets au niveau du récepteur.

IV-4-2 Iperf

Iperf est un outil de mesure de la bande passante et de la qualité d'un lien réseau. Un lien réseau est délimité par deux hôtes exécutants Iperf (source et destination), pour chaque test, il donne la bande passante, les pertes de paquets et d'autres informations selon le besoin.

Fonctionnalités d'iperf

Iperf utilise plusieurs protocoles, les plus connus : TCP et UDP

- **TCP**
 - Mesure de la bande passante (débit) sur une ou plusieurs connexions TCP.
 - Taille de la MTU réelle.
 - Modification de la taille de la fenêtre TCP [RFC2414] [RFC2581].
- **UDP**
 - Le client peut créer des flux UDP avec un débit spécifié.
 - Mesure la perte de paquets (en nombre de paquets et en pourcentage).
 - Mesure de la Gigue (variation de latence).
 - Perte de datagramme : peut être mesurée avec un test Iperf UDP.

IV-4-3 Génération des tests

Pour notre solution nous avons généré un script qui nous permettra d'avoir tous les tests iperf en changeant la probabilité d'erreur et le délai de transmission :

- **Etape 1 :Initialisation**

- Spécification des adresses Ip source et destination ainsi que l'adresse de l'interface de codage au niveau du serveur.
- Détermination du répertoire de codage des nœuds virtuelles ainsi que le répertoire qui contiendra les fichiers des résultats.
- Détermination du débit max de la liaison ainsi que la durée de transmission des paquets.
- Spécification du codeur utilisé, pour notre cas "XOR".

La Figure IV.19 représente cette partie du script.

```
LOGDIR=../results/logs/
SERVER="node@n2"
CLIENT="node@n1"
CLIENT_IP=10.0.1.1
SERVER_IP=10.0.1.2
CODED_SERVER_IP=10.1.1.2
DEV1=veth1
DEV2=veth2
MAXRATE=20mbit
DURATION=60
CODERDIR=/home/node/share/tun/build/apps/singlepath/
CLOG=/home/node/share/results/logs/
XOR_REDUNDANCY="3 5"
```

Figure IV.19 – Initialisation de l'environnement de test

- **Etape 2 : Les fonctions utilisées** Dans cette étapes nous avons écrit trois fonctions :

- **run_iperf** : cette fonction permet de générer un paquet de flux à partir du noeud serveur et mettre le résultat de la transmission du paquet dans un fichier. La Figure IV.20 représente le code de cette fonction.

```
function run_iperf {
if [ $# != 2 ]; then
echo "run_iperf argument mismatch. Arguments needed: \"server_ip
iperf_logfile\"
exit 1
fi
sshpass -p 'node' ssh $CLIENT "iperf -c $1 -t $DURATION -y C" >> $2
}
```

Figure IV.20 – La fonction de test run_iperf

- **run_uncoded** : Cette fonction permet de générer un paquet de flux non codé, son code est représenté par la Figure IV.21.

```
function run_uncoded {
if [ $# != 2 ]; then
echo "run_uncoded argument mismatch. Arguments needed: \"
server_ip iperf_logfile\"
exit 1
fi
run_iperf $1 $2
}
```

Figure IV.21 – La fonction de test run_uncoded

- **run_xor** : Cette fonction permet de générer un paquet de flux codé. Pour cela, nous activons la nouvelle couche de codage ajoutée entre la couche de transport et la couche réseau en lançant l'application "PC_XOR" qui jouera le rôle d'un codeur au niveau du noeud n_2 et d'un décodeur au niveau du noeud n_1 . La Figure IV.22 représente cette partie du script.

```
function run_xor {
if [ $# != 4 ]; then
echo "run_xor argument mismatch. Arguments needed:
\"coding_server_ip iperf_logfile
coding_logfile redundancy\"
exit 1
fi
## Setup
CODER=pc_xor
tmux new-window -dk -t 4 "sshpass -p 'node' ssh $CLIENT \"\$CODERDIR\$CODER
ncif $4 /dev/null $CLIENT_IP $SERVER_IP\"
tmux new-window -dk -t 5 "sshpass -p 'node' ssh $SERVER \"\$CODERDIR\$CODER
ncif $4 $3 $SERVER_IP $CLIENT_IP\"
}
```

Figure IV.22 – La fonction de test run_xor

- Etape 3 : Génération des paquets

Durant cette étape (Figures IV.23 et IV.24) :

- Nous avons fixé le débit max de chaque interface de sortie.
- Nous avons opté pour des probabilités d'erreurs différentes afin de mieux visualiser l'intérêt du codage ainsi que trois niveaux de retard.
- Génération de 10 paquets pour chaque test.
- Ajout d'une quantité fixe de retard et d'erreur à tous les paquets sortant de l'Ethernet des nœuds.
- Lancement du test iperf en spécifiant l'adresse ip de l'interface de codage et les fichiers qui contiendront les résultats ainsi que le taux d'erreur et le délai.

Les Figures IV.23 et IV.24 représentent cette partie du script.

```

for i in $(seq 1 10); do
for ERROR_RATE in 0 0.5 1 2 3 4 5 6 8 10 do
echo Setting error rate to $ERROR_RATE percent loss
sudo tc qdisc add dev $DEV1 root netem rate $MAXRATE loss $ERROR_RATE
sudo tc qdisc add dev $DEV2 root netem rate $MAXRATE loss $ERROR_RATE

echo Running uncoded iperf - $DURATION seconds
IPERF_LOGFILE=iperf_singlehop_uncoded_delay_0_error_$ERROR_RATE.dat
run_uncoded $SERVER_IP $LOGDIR$IPERF_LOGFILE
sleep 1
for RED in $XOR_REDUNDANCY; do
echo Running xor iperf - $DURATION seconds
LOG=singlehop_xor_delay_0_error_$ERROR_RATE\_redundancy_$RED.dat
IPERF_LOGFILE=iperf_$LOG
CODING_LOGFILE=recovery_$LOG
run_xor $CODED_SERVER_IP $LOGDIR$IPERF_LOGFILE $CLOG$CODING_LOGFILE $RED
sleep 1
done

```

Figure IV.23 – Générations des tests-1

```

for DELAY in 0 10 20 30 60 ; do
  DELAY_VAR=$(expr $DELAY / 5)
echo Setting delay to $DELAY milliseconds with variance "$DELAY_VAR" milliseconds
sudo tc qdisc add dev $DEV1 root netem \
  rate $MAXRATE delay "$DELAY"ms "$DELAY_VAR"ms distribution normal loss $ERROR_RATE
sudo tc qdisc add dev $DEV2 root netem \
  rate $MAXRATE delay "$DELAY"ms "$DELAY_VAR"ms distribution normal loss $ERROR_RATE
echo Running uncoded iperf - $DURATION seconds
IPERF_LOGFILE=iperf_singlehop_uncoded_delay_$DELAY\_error_$ERROR_RATE.dat
run_uncoded $SERVER_IP $LOGDIR$IPERF_LOGFILE
sleep 1
for RED in $XOR_REDUNDANCY; do
echo Running xor iperf - $DURATION seconds
LOG=singlehop_xor_delay_$DELAY\_error_$ERROR_RATE\_redundancy_$RED.dat
IPERF_LOGFILE=iperf_$LOG
CODING_LOGFILE=recovery_$LOG
run_xor $CODED_SERVER_IP $LOGDIR$IPERF_LOGFILE $CLOG$CODING_LOGFILE $RED

```

Figure IV.24 – Générations des tests-2

IV-4-4 Résultats

Le scénario à saut unique (décrit dans la section 3 de ce chapitre) consiste en deux nœuds virtuels, chacun connecté à des commutateurs virtuels individuels. Ces commutateurs sont par la suite connectés avec une connexion Ethernet virtuelle, sur laquelle le retard et les pertes sont introduits en utilisant le script expliqué précédemment. Ce script génère des fichiers contenant les résultats des tests iperf, nous avons représenté ces résultats en utilisant des graphes afin de mesurer les performances de cette implémentation.

- **Analyse du débit**

Pour tester les performances du codage utilisé et montrer l'efficacité de ce dernier comparant à la transmission des paquets non codés, nous avons établi une correspondance entre le débit et les différentes probabilités d'erreur comme indiqué dans la Figure IV.25.

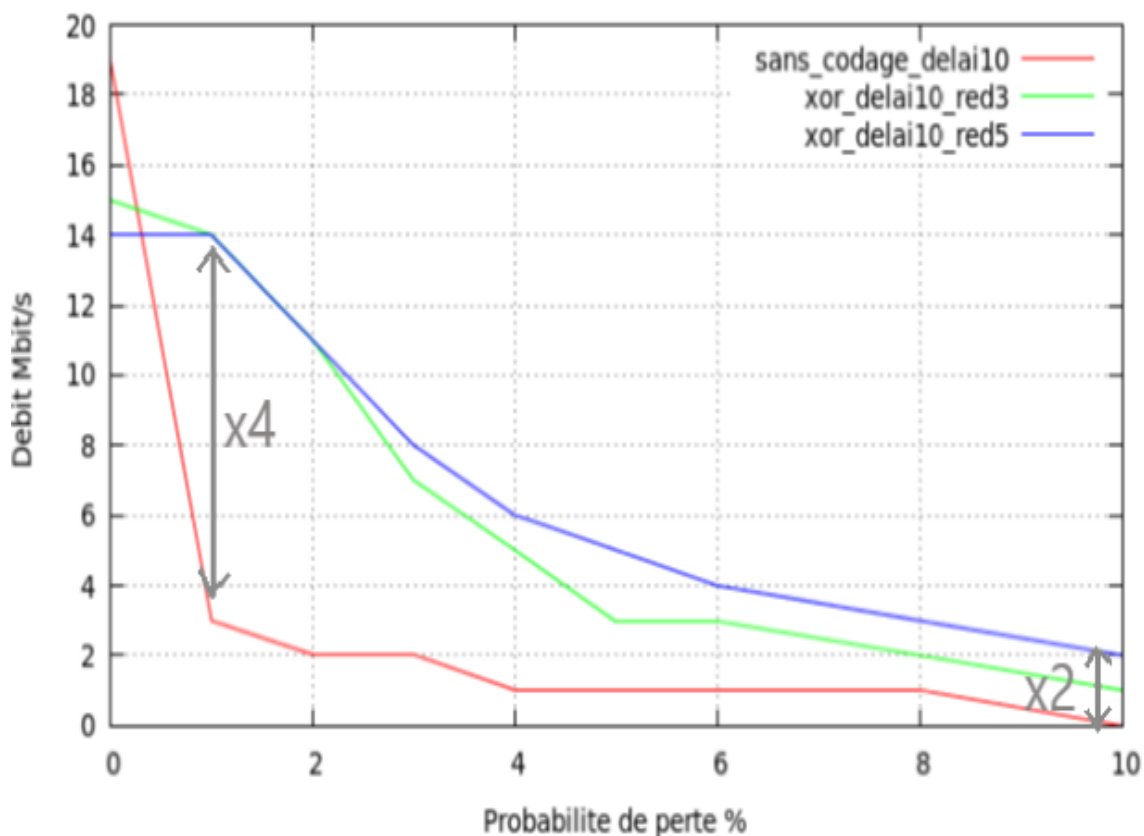


Figure IV.25 – Performance de codage à saut unique avec un délai=10

Dans ce graphe le débit d'un flux TCP non codé est comparé aux connexions TCP effectuées par l'approche de codage déployée en utilisant un codeur XOR. Les débits atteignables pour les données codées en utilisant un codage "xor avec redondance 3" et un codage "xor avec redondance 5" sont respectivement 1 Mbit/s et 2Mbit/s à une probabilité de perte de 10 %. En revanche, ce débit s'annule pour un flux non codé pour une même probabilité d'erreur. Le gain obtenu est équivalent à 4 fois les performances d'un flux non codé à une probabilité de perte de paquet de 1 % et de 2 fois à une probabilité de perte de 10%. Nous constatons que l'utilisation de la redondance freine la décroissance du débit en fonction des probabilités d'erreur.

- Analyse du taux de récupération

La Figure IV.26 représente le taux de récupération des paquets en fonction des probabilités de perte en utilisant le codage XOR avec redondances 3 et 5.

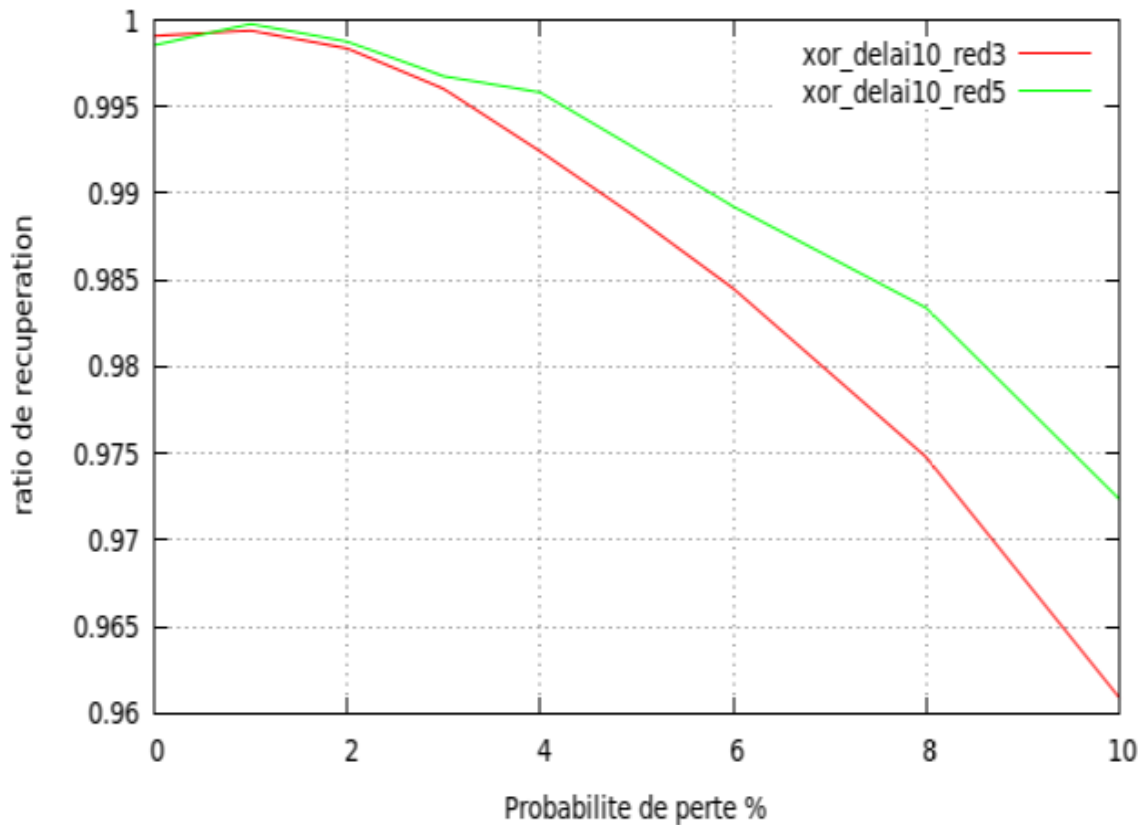


Figure IV.26 – Taux de récupération avec un délai=10

La récupération de données (paquets) dans l'approche de codage développée est révélée car le taux de récupération qui peut se produire sur le lien des différents effacement reste toujours supérieur à 0.955 et donc proche de la récupération totale des paquets transmis malgré le fait que le taux d'erreur atteint les 10 %.

IV-4-5 Conclusion

Dans ce chapitre nous avons implémenté la stratégie de codage XOR dans une architecture SDN. Le concept, l'implémentation et l'évaluation de notre solution ont été détaillés en se basant sur une plateforme préalablement établie. Les résultats de simulations étaient comparés à ceux d'une architecture exempte du codage réseau. Pour l'évaluation de comportement du codage utilisé nous avons choisi deux métriques de performance, à savoir le débit et le taux de récupération des paquets. Les résultats obtenus montrent l'efficacité et la robustesse qu'apporte ce codage en termes de bonne récupération de paquets en présence des délais et des erreurs introduites dans les liens de transmissions.

Conclusion générale

Dans le cadre de notre projet de fin d'étude sanctionnant un cursus de deux années de Licence Génie des Télécommunications et Réseaux, il nous a été donné de travailler sur l'implémentation d'une stratégie de codage réseau pour le SDN. Le présent document a récapitulé le travail effectué dans le cadre de ce projet. Pour finaliser cette étude, nous allons exposer une synthèse de notre parcours :

La première partie regroupe des généralités sur SDN, son architecture, ses principes et ses services innovants, notamment le routage multidiffusion, la sécurité, le contrôle d'accès, la gestion de la bande passante, l'ingénierie de trafic et la QoS.

Notre étude bibliographique a été dirigée dans le deuxième chapitre sur le codage réseau. Nous avons effectué une étude générale des différents aspects de codage réseau et ses approches. Nous avons donné quelques exemples d'applications de cette discipline dans le monde des réseaux et nous avons terminé par une présentation de codage réseau Ou exclusive.

Après l'étude bibliographique nécessaire sur les deux paradigmes, à savoir le SDN et le codage réseau, il nous a été indispensable de détailler dans le troisième chapitre la conception de notre solution qui vise à mettre en place une stratégie de codage réseau "xor_redundancy" dans une architecture SDN. Nous avons détaillé l'algorithme mis en oeuvre et nous avons déterminé la couche protocolaire chargée de l'implémentation.

Dans le quatrième chapitre, nous avons expliqué la manière dont nous avons intégré concrètement la stratégie de codage au sein du réseau SDN. Nous avons décrit l'ensemble architectural pour une mise en oeuvre conjointe de ces deux concepts et démontrer leur potentiel dans une topologie clé, à savoir, topologie à saut unique. Nous avons terminé notre étude par une comparaison des performances du potentiel TCP avec et sans utilisation de la stratégie SDN-NC pour stabiliser les liens réseau. Autrement dit, nous avons fourni une utilisation de SDN et NC qui est transparente aux protocoles de bout en bout.

L'objectif fixé, qui consistait en la réunion des deux concepts de réseautage pertinents nommément le codage réseau et le réseau défini par logiciel, est atteint. Nous avons montré leur potentiel commun d'opérer ensemble. Les résultats des tests montrent que des gains considérable de 2 fois à 4 fois sont réalisables avec cette approche de codage en termes de débit, permettant ainsi une transmission fiable de paquets en minimisant les pertes dûes aux aléas et perturbations.

En somme, ce projet nous a permis de toucher à plusieurs aspects du monde de l'informatique (Réseau, SDN, Vistualisation ...). Il nous a présenté un très bon moyen pour la mise en pratique des connaissances acquises durant ces dernières années de formations et c'est avéré être une bonne opportunité pour enrichir nos connaissances.

Comme perspective, nous proposons les solutions suivantes :

- Tester notre solution sur les larges plate-formes de test à savoir des topologies SDN à saut multiple nécessitant un recodage au niveau des nœuds intermédiaires.
- Implémenter de nouveaux services comme le codage réseau linéaire aléatoire, vu l'aspect modulaire de notre architecture.
- Intégrer la stratégie de codage voulu au niveau du plan de contrôle à l'aide d'un module qui gère les états des liens fournis par le plan de données.

Bibliographie

- [1] J. Rexford N. Feamster and E. Zegura. The road to sdn : an intellectual history of programmable networks. *ACM Mag*, 11(12), 2013.
- [2] D. Kreutz, F.M.V. Ramos, C.E. Rothenberg, S. Azodolmolky, S. Uhlig, and P.E. Verissimo. “software-defined networking : A comprehensive survey”. *Proceedings of the IEEE*, 103(1) :pp. 14–76, 2015.
- [3] D. M. Dumitriu Midokura. Using southbound apis to build an sdn solution. *OPENDAYLIGHT SUMMIT*, 5 Feb 2014.
- [4] Ricard Vilalta, Arturo Mayoral, and Ramon Casellas. Sdn/nfv orchestration of multi-technology and multi-domain networks in cloud/fog architectures for 5g services, pp. 1-3. 2016.
- [5] S. Jain, A. Kumar, and S. Mandal. ”b4 : Experience with a globally-deployed software defined wan”. *ACM SIGCOMM Computer Communication Review*, 43(04) :pp. 3–14, 2013.
- [6] R. Ahlswede, Ning Cai, S. Y. R. Li, and R. W. Yeung. ”network information flow”. *IEEE Transactions on Information Theory*, 46(04) :pp. 1204–1216, Jul 2000.
- [7] A. Jiang. Network coding for joint storage and transmission with minimum cost. *IEEE International Symposium on Information Theory*, pp. 1359-1363, July 2006.
- [8] Hülya Seferoğlu and Athina Markopoulou. ”intra and inter-session network coding”. *University of California, Irvine*.
- [9] Min Yang. ”network coding for application layer multicast”. *Stony Brook University*, 2009.
- [10] S. Sengupta, S. Rayanchu, and S. Banerjee. Network coding-aware routing in wireless networks. *IEEE/ACM Transactions on networking*, 18(4) :pp. 1158–1170, Aug 2010.
- [11] Zhang Shengli, Soung Chang Liew, and Lu Lu. Physical-layer network coding : Tutorial, survey, and beyond. 2012.
- [12] Dah Ming Chiu, R. W. Yeung, Jiaqing Huang, and Bin Fan. Can network coding help in p2p networks? *International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, pp. 1-5, April 2006.
- [13] C.K. Yeo, B.S. Lee, and M.H. Er. ”a survey of application level multicast techniques”. *IEEE Network*, 27(15), pp. 1547-1568, 22 September 2004.

- [14] S. Katti, D. Katabi, Wenjun Hu, and Rahul Hariharan. The importance of being opportunistic : Practical network coding for wireless environments. *In Proc. 43rd Allerton Conference on Communication, Control, and Computing, Monticello, IL*, September 2005.
- [15] Yunnan Wu and Philip A. Chou and S. Y. Kung. Information exchange in wireless networks with network coding and physical-layer broadcast. *Technical Report MSR-TR-2004-78, Microsoft Research*, August 2004.
- [16] Qiyan Wang, Long Vu, Klara Nahrstedt, and Himanshu Khurana. Malicious nodes identification scheme in network-coding-based peer-to-peer streaming. *IEEE Infocom*, pp. 1-5, March 2010.
- [17] Ning Cai and R. W. Yeung. Secure network coding. *Proceedings IEEE International Symposium on Information Theory*, 2002.
- [18] Efficient scheme for data transfer using xor network coding. *International Journal of Engineering Research and General Science*, vol 2 ; Issue 6 :pp. 2091–2730, October-November,2014.
- [19] Opendaylight. [Online]. Available : <http://www.opendaylight.org/>.
- [20] Onos. [Online]. Available :<https://github.com/opennetworkinglab/onos>.
- [21] S. Lawson. Network heavy hitters to pool sdn efforts in.opendaylight project. *Network World*, 8 April 2013.
- [22] Production quality, multi-layer open vswitch. [Online]. Available : <http://openvswitch.org/>.
- [23] Ed. Pfaff and B. Davie. The open vswitch database management protocol. *Internet Draft, Internet Engineering, Task Force*, Octobre 2013.
- [24] Tun/tap interface tutorial. [Online]. Available : <http://backreference.org/2010/03/26/tuntap-interfacetutorial/>, 2013.
- [25] D. L. Tennenhouse and D. J. Wetherall. Towards an active network architecture. *Proceedings DARPA Active Networks Conference and Exposition*, pp. 2-15, 2002.
- [26] B. Schwartz, A. W. Jackson, W. T. Strayer, Wenyi Zhou, R. D. Rockwell, and C. Partridge. “smart packets for active networks”. *IEEE Second Conference on Open Architectures and Network Programming Proceedings OPENARCH '99*, 1999.
- [27] D. J. Wetherall, J. V. Guttag, and D. L. Tennenhouse. “ants : a toolkit for building and dynamically deploying network protocols”. *IEEE Open Architectures and Network Programming*, pp. 117–129, Apr 1998.
- [28] A.T. Campbell, H.G. De Meer, M.E. Kounavis, K. Miki, B.J. Vicente, and D. Villela. “a survey of programmable networks”. *ACM SIGCOMM Computer Communication Review*, 29(2) :pp. 7–23, April 1999.

- [29] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. “ethane : Taking control of the enterprise”. *Proceedings of the Conference on Applications, Technologies, Architectures and Protocols for Computer Communications*, pp. 1–12, 2007.

ANNEXES

Historique des réseaux SDN

Avant l'apparition des réseaux SDN tels que nous les connaissons aujourd'hui, plusieurs idées et travaux ont été proposés auparavant, notamment la programmation du réseau et la séparation des plans de contrôle et de données. Nous donnons dans cette section un bref aperçu de ces travaux, qui peuvent être considérés comme des ancêtres de SDN. La première idée de programmation de réseaux a été développée en 1996 sous le nom de « réseaux actifs » (AN, Active Network) [25]. Ces réseaux injectent des programmes parmi les données du paquet. Quand un noeud du réseau reçoit ces paquets, il extrait et exécute les programmes à partir des données du paquet et déclenche par conséquent des actions de transmission, de modification ou de suppression du paquet. Avec cette approche, de nouveaux mécanismes de services et de routage du réseau peuvent être implémentés sans modification des équipements de transmission. Plusieurs études ont été menées sur les ANs, en particulier sur les paquets intelligents, ANTS et SwitchWare [26] [27]. Puisque les paquets des ANs peuvent transporter des programmes malveillants, une alternative aux ANs appelée « réseaux programmables » (PN) a été proposée en 1999 [28]. Les PN injectent des programmes à l'intérieur des noeuds du réseau. Ces noeuds exécutent les programmes uniquement après une phase de signalisation et de vérification, pour renforcer la sécurité. Les ANs et les PN ont cherché à introduire la programmabilité dans les réseaux à travers des paquets et des switchs programmables. Ces approches n'ont pas réduit la complexité de l'infrastructure réseau. D'autres part, certains projets ont essayé de séparer le plan de contrôle du plan de données pour simplifier l'architecture réseau et fournir une abstraction sur l'infrastructure physique. Par exemple, le projet de recherche dénommé DCAN (Devolved Control of ATM Networks) de l'université de Cambridge dont l'objectif était de développer l'infrastructure réseau de manière à ce que les fonctions de contrôle et de gestion de plusieurs équipements (les switchs ATM dans le cas du DCAN) soient découplées de l'équipement physique et déléguées aux entités externes telles que le gestionnaire.

Le début des réseaux SDN a commencé avec le projet Ethane, lancé en 2006 à l'université de Stanford. En effet, le projet Ethane définit une nouvelle architecture pour les réseaux d'entreprises [29]. L'objectif d'Ethane était d'avoir un contrôleur centralisé pour gérer les règles et la sécurité dans le réseau. Ethane utilise deux composantes : un contrôleur pour décider si un paquet doit être transmis et un switch Ethane composé de table et d'une chaîne de communication entre les deux. Ethane était une source d'inspiration pour un nouveau concept appelé aujourd'hui « réseaux programmés par logiciel » (SDN, Software-Defined Networking).

Définition

Selon l'ONF : "Le logiciel SDN est une architecture émergente qui est dynamique, gérable, rentable et adaptable, ce qui la rend idéale pour la haute bande passante et la nature dynamique des applications actuelles. Cette architecture découple les fonctions de contrôle et d'acheminement du réseau permettant de contrôler directement le contrôle du réseau et d'extraire l'infrastructure sous-jacente pour les

applications et les services de réseau”.

Le but du SDN est de fournir des interfaces ouvertes permettant le développement de logiciels pouvant contrôler la connectivité fournie par un ensemble de ressources réseau, l’inspection et la modification possibles du trafic qui peut être effectué dans le réseau, ainsi que la simplification de l’administration du réseau et de rendre la consommation des ressources réseaux par des applications plus flexibles.

La programmation des équipements

Les APIs :

La programmation des équipements réseau nécessite sur ces derniers la capacité de recevoir des directives de l’extérieur. Pour cela des interfaces de programmation sont nécessaires. Il existe de nombreuses APIs, standards ou propriétaires, pouvant agir sur différents éléments de l’équipements (plan de données, plan de contrôle...) il est communément admis qu’une seule API universelle ne suffira pas, les équipements modernes implémentent souvent plusieurs APIs.

Quelques APIs les plus communément supportées sur les équipements réseau :

- **Netconf/YANG :**

Le protocole de configuration réseau (Netconf) [RFC6632] appartient au standard IETF, est un protocole de gestion de réseau [RFC6632]. Netconf fournit des mécanismes installer, manipuler et supprimer la configuration du réseau. Ce protocole utilise un codage de données basé sur XML pour les données de configuration ainsi que les messages de protocole.

- **CLI :**

L’accès en ligne de commande aux équipements via telnet/ssh est une API qui est souvent la plus utilisée pour programmer/configurer le réseau.

- **SNMP :**

Le protocole SNMP (Simple Network Management Protocol) appartient au standard IETF, est un protocole de gestion qui est actuellement à sa troisième révision (SNMPv3) [RFC3417][RFC3412][RFC3414]. Il consiste en un ensemble de normes pour la gestion de réseau, y compris un protocole de couche d’application, un schéma de base de données et un ensemble d’objets de données. SNMP expose les données de gestion (objets gérés) sous la forme de variables sur les systèmes gérés, qui décrivent la configuration du système. Ces variables peuvent ensuite être interrogées et définies en gérant les applications.

- **OpenFlow :**

C’est un protocole SDN southbound proposé pour être un protocole standard dans l’industrie et utilisé pour transporter une fonction de logique de contrôle d’un commutateur à un contrôleur.

- Protocole vSwitch Data Base ouvert(OVSDB) :

Protocole de gestion utilisé par le commutateur logiciel Open vSwitch open source.

Les tables de flux

Sont constituées d'entrées de flux qui définissent comment les commutateurs doivent se comporter avec un flux de trafic provenant de/vers différentes interfaces physiques et virtuelles. Les tables d'un commutateur OpenFlow sont numérotées à partir de 0.

Premièrement, les paquets entrants sont comparés aux entrées de flux dans la table 0.

Lorsqu'une entrée de flux est trouvée, les instructions incluses dans cette entrée seront exécutées sur le paquet. Les instructions peuvent envoyer explicitement le paquet à une autre table de flux, répétant le processus encore et encore conformément au jeu d'instructions correspondant à ce paquet.

Si une table de flux pour un paquet est manquante, plusieurs instructions peuvent être définies pour cette situation. Les options incluent la suppression du paquet, sa transmission à un autre flux ou son envoi à un contrôleur via le canal de contrôle, comme illustré dans la Figure 1.

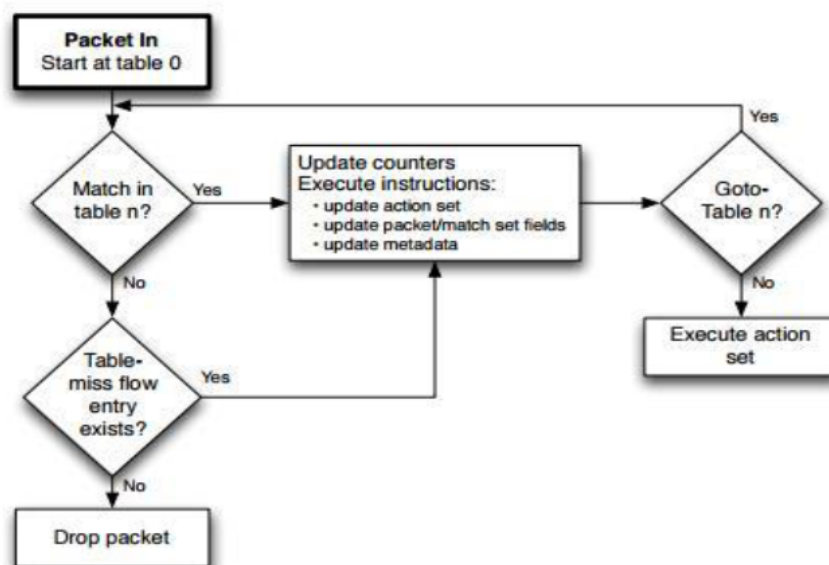


Figure 1 - Organigramme de paquets

Chaque table de flux du commutateur contient un ensemble d'entrées de flux qui présentent les règles d'acheminement des paquets. La Figure 2 schématise la structure d'une entrée OpenFlow.

- **Match Fields** : Champs de correspondance qui définissent le modèle de flux de paquets à travers l'instanciation des champs d'en-tête allant de la couche Ethernet à la couche transport.
- **Counters** : également appelés compteurs, collectent les informations d'une entrée d'un flux particulier. Les informations collectées par le compteur sont : le nombre de paquets reçus, le nombre d'octets reçus et la durée d'un flux particulier.
- **Actions** : Actions à appliquer aux paquets qui correspondent à l'entrée de flux.

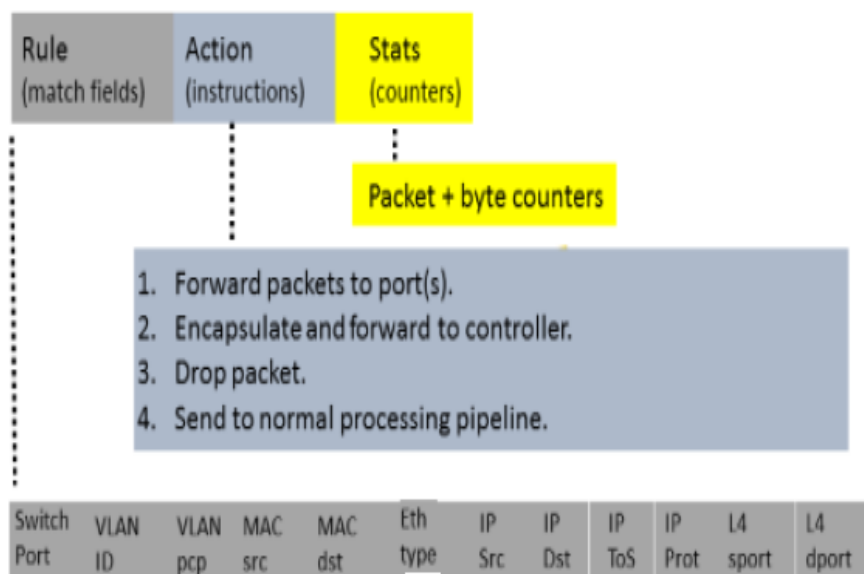


Figure 2 - Structure d'une entrée OpenFlow

```
#!/bin/bash
img=ubuntu64-1.img
share=$PWD/../
wan=eth0

# Make sure we are inside a screen
if [ $TERM != "screen" ]; then
    echo Don't we want to be inside a screen?
    exit 1
fi

./of_stop.sh

# Setup NAT configuration de l'interconnexion
sudo ip_forward 1
sudo iptables -t nat -A POSTROUTING -o $wan -j MASQUERADE

for j in $(seq 1 2); do
#configuration de l'interconnexion entre les vms et l'hyperviseur
    sudo ip tuntap add tap$j mode tap
    sudo ip address add dev tap$j 10.0.2.10$j/24
    sudo ip link set dev tap$j up
    sudo ip route add to 10.0.2.$j dev tap$j
    sudo iptables -A FORWARD -i tap$j -j ACCEPT

    sudo ip tuntap add of_tap$j mode tap

    ### OVS setup creation et configuration des périphérique intermédiaire ovs
    sudo ovs-vsctl add-br br$j
    sudo ovs-vsctl set bridge br$j

    ### Connet device to bridge
    sudo ovs-vsctl add-port br$j of_tap$j

    sudo ip link set dev of_tap$j up

    sudo ip link set dev br$j up

# # Create image
    qemu-img create -f qcow2 -b $img num$j.img
done

# Create veth interface creation des interfaces entre périphérique intermédiaire
sudo ip link add veth1 type veth peer name veth2
sudo ip link set veth1 up
sudo ip link set veth2 up
```

Figure 3 - Script de la topologie à un seul saut partie-1

```
sudo ovs-vsctl add-port br1 veth1
sudo ovs-vsctl add-port br2 veth2
#starting the vms
echo Opening KVM

tmux new-window -d -t 1 "kvm \
-smp 1 \
-drive file=num1.img,if=virtio \
-m 1024 \
-nographic \
-netdev tap,ifname=tap1,script=no,downscript=no,id=lan0 \
-device virtio-net-pci,netdev=lan0 \
-net nic,macaddr=fe:fe:00:0a:01:01 \
-net tap,ifname=of_tap1,script=no \
-fsdev local,security_model=passthrough,id=fsdev0,path=$share \
-device virtio-9p-pci,id=fs0,fsdev=fsdev0,mount_tag=hostshare"

tmux new-window -d -t 2 "kvm \
-smp 1 \
-drive file=num2.img,if=virtio \
-m 1024 \
-nographic \
-netdev tap,ifname=tap2,script=no,downscript=no,id=lan0 \
-device virtio-net-pci,netdev=lan0 \
-net nic,macaddr=fe:fe:00:0a:02:01 \
-net tap,ifname=of_tap2,script=no \
-fsdev local,security_model=passthrough,id=fsdev0,path=$share \
-device virtio-9p-pci,id=fs0,fsdev=fsdev0,mount_tag=hostshare"

echo "Setting switch to listen for controller on local interface"

### OSV setup #configuration of the switch to listen to the controller
sudo ovs-vsctl set-controller br1 tcp:0.0.0.0:6633
sudo ovs-vsctl set-controller br2 tcp:0.0.0.0:6633

echo "Waiting 20s for nodes to start..."
sleep 20
echo "Setting up hostshare"

## mount hostshare
for j in $(seq 1 2); do
#configuration du partage
ssh node@n$j "sudo mount hostshare"
#configuration des interfaces tun
ssh node@n$j "~/share/tun/scripts/tun_setup.sh ncif 10.1.1.$j"
done
```

Figure 4 - Script de la topologie à un seul saut partie-2

```
#!/bin/bash
#enc: utf8

# General parameters
LOGDIR=./results/logs/
SERVER="node@n2"
CLIENT="node@n1"
CLIENT_IP=10.0.1.1
SERVER_IP=10.0.1.2
CODED_SERVER_IP=10.1.1.2
DEV1=veth1
DEV2=veth2

MAXRATE=20mbit

# Iperf duration
DURATION=60

# Coder directory on the virtual machines
CODERDIR=/home/node/share/tun/build/apps/singlepath/

# Coder logdir
CLOG=/home/node/share/results/logs/

XOR_REDUNDANCY="3 5"

# Coders in use
# CODERS="pc_xor"

function poll_recovery_stats {
    if [ $# != 0 ]; then
        echo "poll_recovery_stats argument mismatch"
        exit 1
    fi
}
```

Figure 5 - run_single_hop.sh -1

```

    sshpass -p 'node' ssh $SERVER "echo \"print_decode_stat\" > /dev/
udp/127.0.0.1/54321"
    sshpass -p 'node' ssh $CLIENT "echo \"print_decode_stat\" > /dev/
udp/127.0.0.1/54321"
    sleep 1
}
function run_iperf {
    if [ $# != 2 ]; then
        echo "run_iperf argument mismatch. Arguments needed: \"server_ip
iperf_logfile\"
        exit 1
    fi
    sshpass -p 'node' ssh $CLIENT "iperf -c $1 -t $DURATION -y C" >> $2
}

function run_uncoded {
    if [ $# != 2 ]; then
        echo "run_uncoded argument mismatch. Arguments needed: \"server_ip
iperf_logfile\"
        exit 1
    fi
    run_iperf $1 $2
}

function run_xor {
    if [ $# != 4 ]; then
        echo "run_xor argument mismatch. Arguments needed: \"coding_server_ip
iperf_logfile coding_logfile redundancy\"
        exit 1
    fi

    CODER=pc_xor
    | tmux new-window -dk -t 4 "sshpass -p 'node' ssh $CLIENT \"${CODERDIR}$CODER
ncif $4 /dev/null $CLIENT_IP $SERVER_IP\"
    tmux new-window -dk -t 5 "sshpass -p 'node' ssh $SERVER \"${CODERDIR}$CODER
ncif $4 $3 $SERVER_IP $CLIENT_IP\"

```

Figure 6 - run_single_hop.sh -2

```

run_iperf $1 $2
poll_recovery_stats
sshpass -p 'node' ssh $CLIENT "killall $CODER"
sshpass -p 'node' ssh $SERVER "killall $CODER"
sleep 1
}
if [ $TERM != "screen" ]; then
    echo Don't we want to be inside a screen?
    exit 1
fi
# Cleaning up:
sudo tc qdisc del dev $DEV1 root
sudo tc qdisc del dev $DEV2 root
sudo tc qdisc del dev of_tap1 root
sudo tc qdisc del dev of_tap2 root
sudo tc qdisc add dev of_tap1 root netem rate $MAXRATE
sudo tc qdisc add dev of_tap2 root netem rate $MAXRATE
#Make sure an iperf server is running on the server node
for i in $(seq 1 10); do
    for ERROR_RATE in 0 0.5 1 2 3 4 5 6 8 10
    do
        echo Setting error rate to $ERROR_RATE percent loss

        # Set error and maxrate
        sudo tc qdisc add dev $DEV1 root netem rate $MAXRATE loss $ERROR_RATE
        sudo tc qdisc add dev $DEV2 root netem rate $MAXRATE loss $ERROR_RATE

        # Run without delay:
        echo Running uncoded iperf - $DURATION seconds
        IPERF_LOGFILE=iperf_singlehop_uncoded_delay_0_error_$ERROR_RATE.dat
        run_uncoded $SERVER_IP $LOGDIR$IPERF_LOGFILE

        sleep 1

        for RED in $XOR_REDUNDANCY; do
            echo Running xor iperf - $DURATION seconds

```

Figure 7 - run_single_hop.sh -3


```

CODING_LOGFILE=recovery_$LOG
run_xor $CODED_SERVER_IP $LOGDIR$IPERF_LOGFILE $CLOG$CODING_LOGFILE $RED
sleep 1
done
# Remove loss on packets
# echo Removing delay rule
sudo tc qdisc del dev $DEV1 root
sudo tc qdisc del dev $DEV2 root

| for DELAY in 0 10 20 30 60 ; do
    DELAY_VAR=$(expr $DELAY / 5)
    echo Setting delay to $DELAY milliseconds with variance "$DELAY_VAR" milliseconds
    sudo tc qdisc add dev $DEV1 root netem \
        rate $MAXRATE delay "$DELAY"ms "$DELAY_VAR"ms distribution normal loss $ERROR_RATE
    sudo tc qdisc add dev $DEV2 root netem \
        rate $MAXRATE delay "$DELAY"ms "$DELAY_VAR"ms distribution normal loss $ERROR_RATE
    echo Running uncoded iperf - $DURATION seconds
    IPERF_LOGFILE=iperf_singlehop_uncoded_delay_$DELAY\_error_$ERROR_RATE.dat
    run_uncoded $SERVER_IP $LOGDIR$IPERF_LOGFILE
    sleep 1
    for RED in $XOR_REDUNDANCY; do
        echo Running xor iperf - $DURATION seconds
        LOG=singlehop_xor_delay_$DELAY\_error_$ERROR_RATE\_redundancy_$RED.dat
        IPERF_LOGFILE=iperf_$LOG
        CODING_LOGFILE=recovery_$LOG
        run_xor $CODED_SERVER_IP $LOGDIR$IPERF_LOGFILE $CLOG$CODING_LOGFILE $RED
        sleep 1
    done

    sudo tc qdisc del dev $DEV1 root
    sudo tc qdisc del dev $DEV2 root

```

Figure 8 - run_single_hop.sh -4

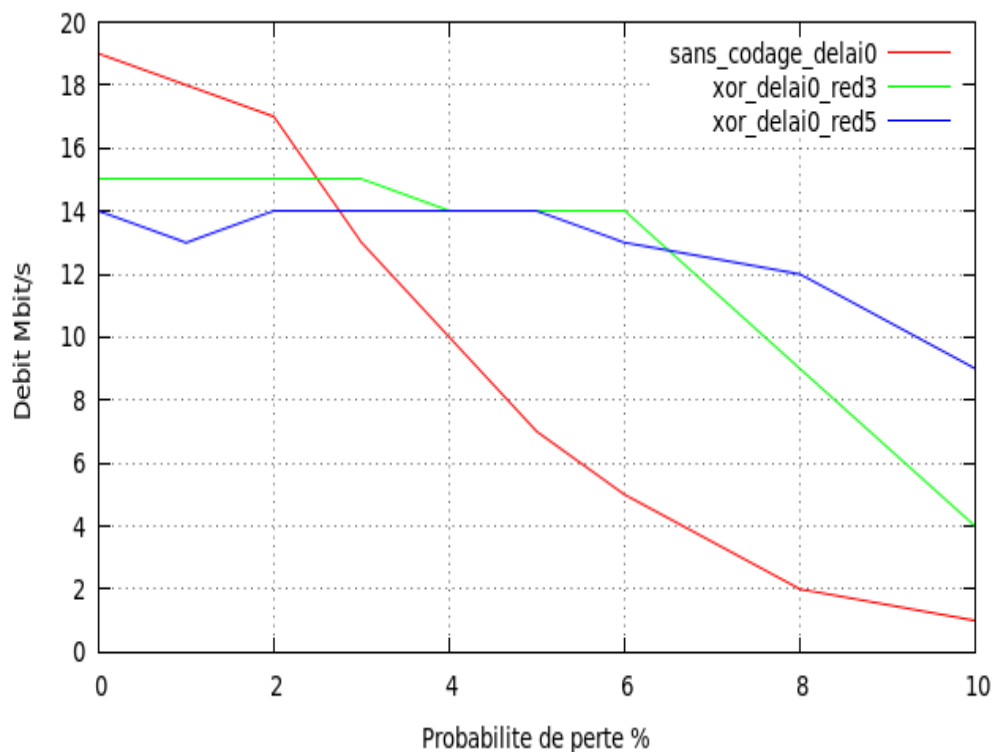


Figure 9 - Performance de codage délai=0

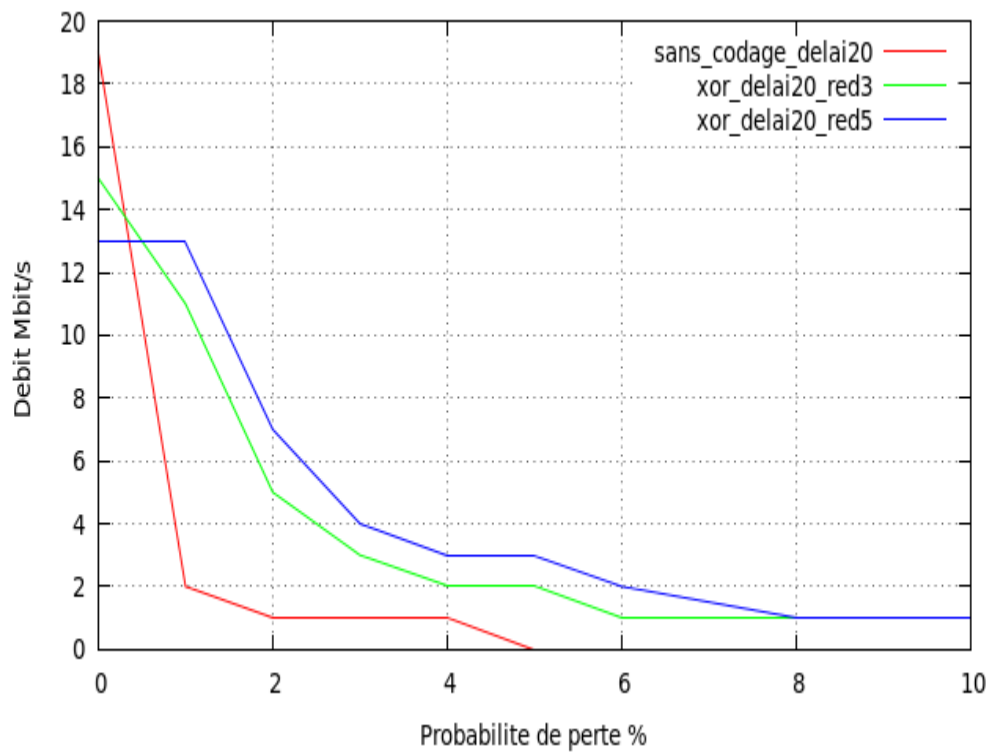


Figure 10 - Performance de codage délai=20

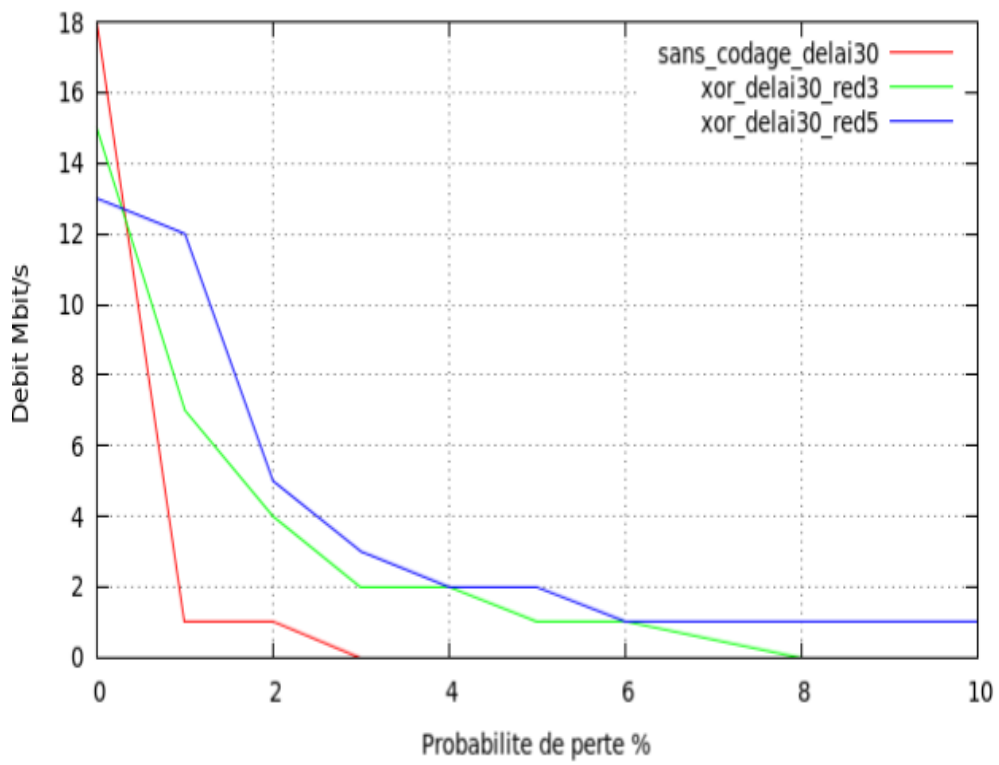


Figure 11 - Performance de codage délai=30

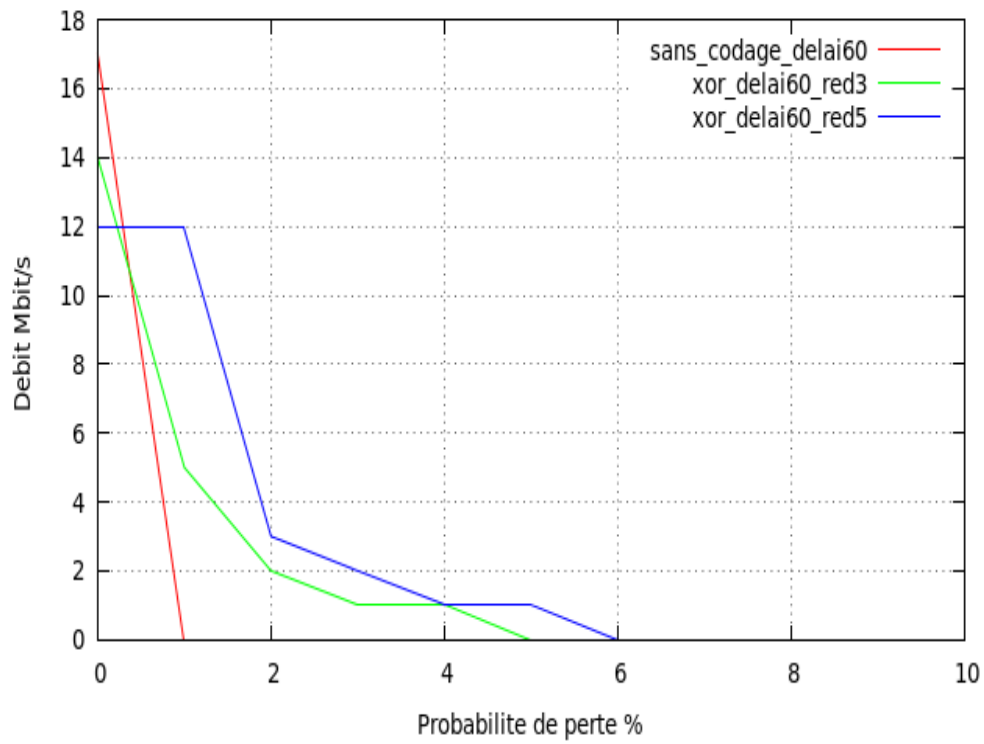


Figure 12 - Performance de codage délai=60

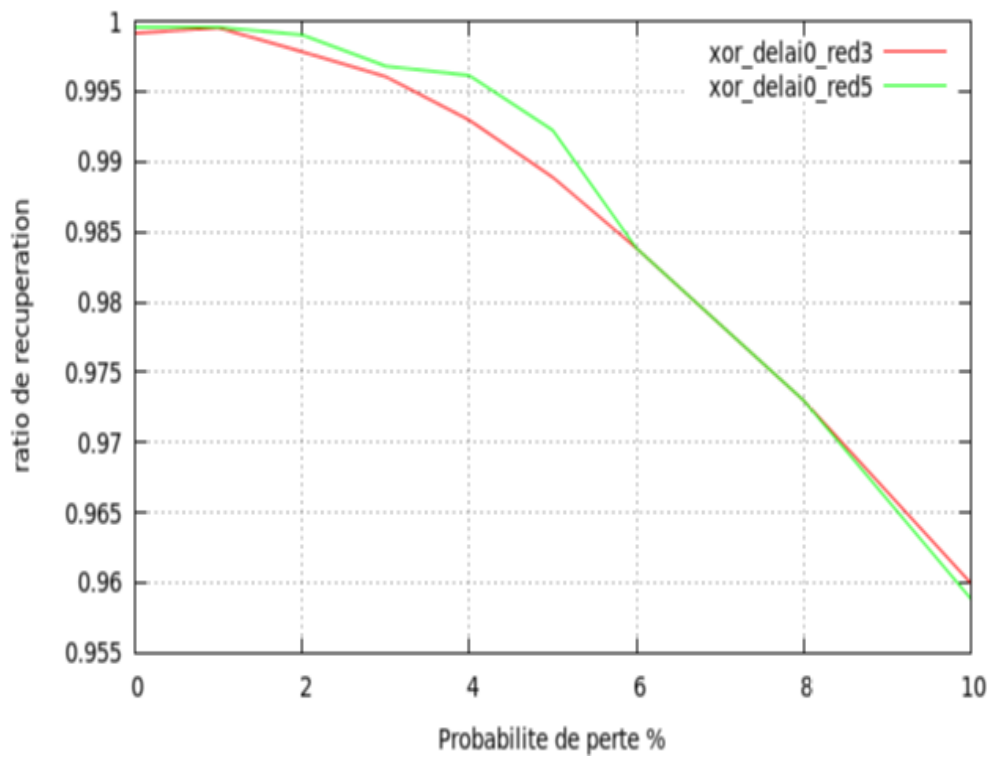


Figure 13 - ratio de récupération délai=0

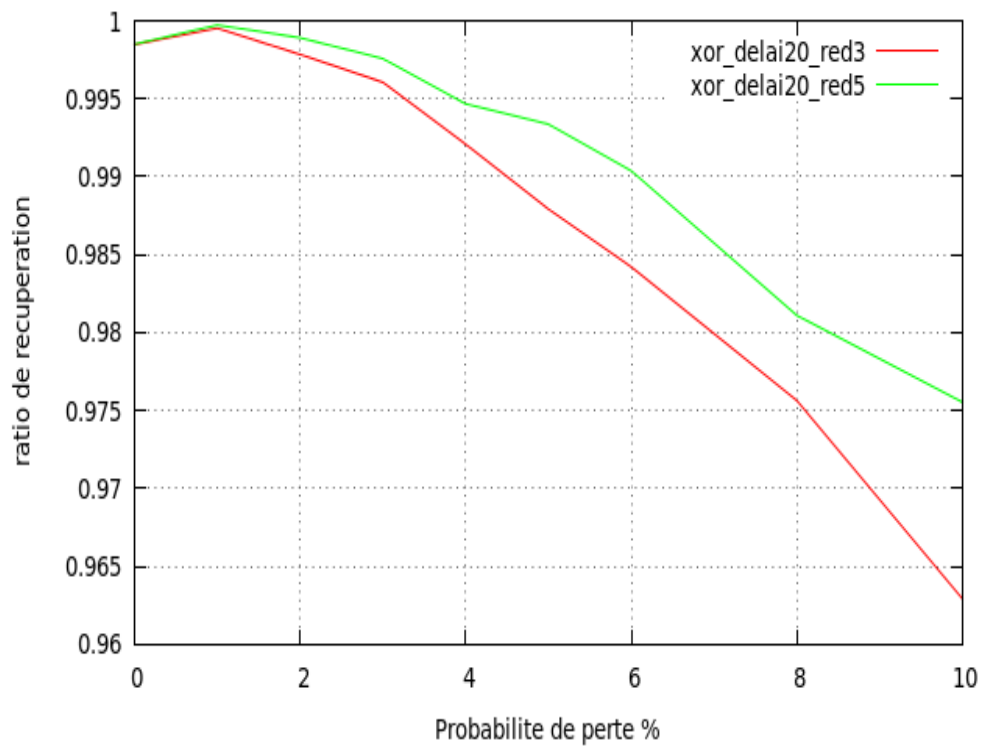


Figure 14 - ratio de récupération délai=20

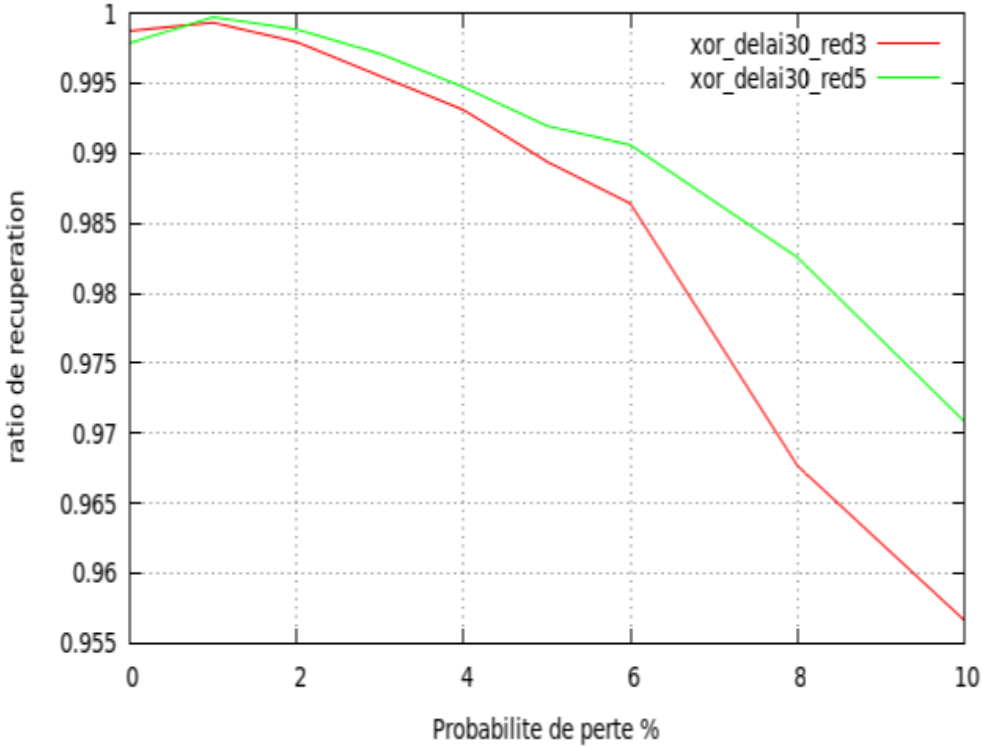


Figure 15 - ratio de récupération délai=30

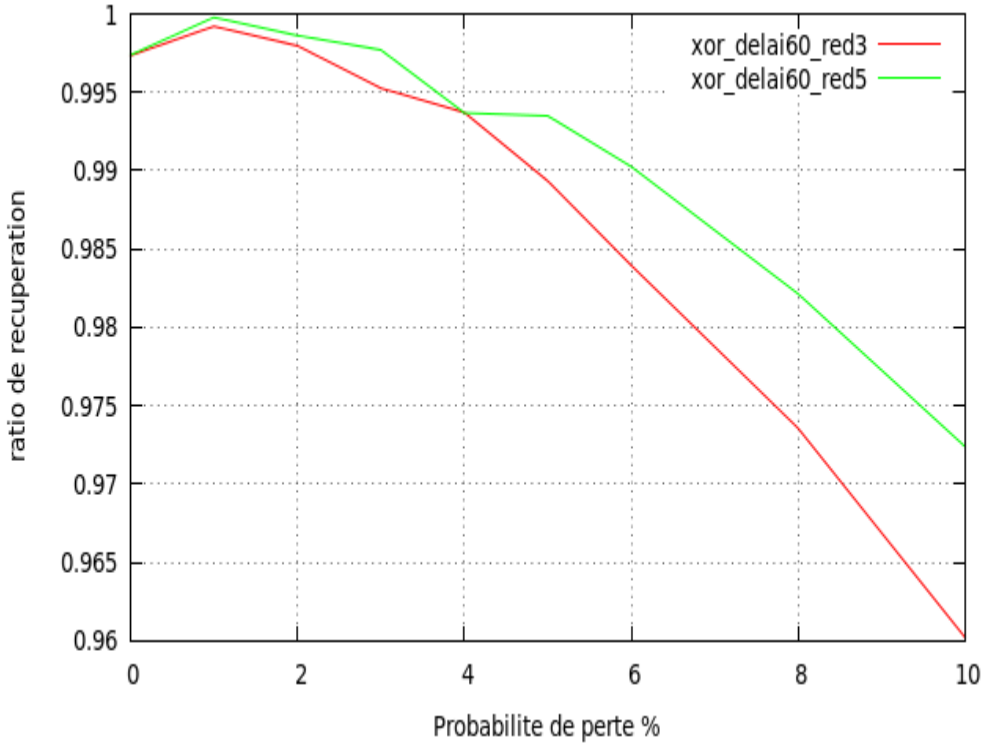


Figure 16 - ratio de récupération délai=60