Virtualisation des fonctions d'un Cloud Radio Access Network(C-RAN)

Tarek Rabia

THÈSE DE DOCTORAT DE
LA SORBONNE UNIVERSITÉ

Spécialité

Informatique

École doctorale Informatique, Télécommunications et Électronique de Paris

Présentée par

Tarek RABIA

Pour obtenir le grade de

DOCTEUR de la SORBONNE UNIVERSITE

Sujet de la thèse :

Virtualisation des Fonctions d'un Cloud Radio Access Network (C-RAN)

soutenue le 29 Janvier 2018
devant le jury composé de :

M. Rami LANGAR	Rapporteur	Professeur - Université Paris-Est Marne-la-Vallée
M. Nadjib ACHIR	Rapporteur	HDR - Université Paris 13 - France
Mme. Noëmie SIMONI	Examinateur	Professeur - École supérieure Télécom ParisTech
M. Prosper CHEMOUIL	Examinateur	Directeur de recherche - Orange Labs
M. Stefano SECCI	Examinateur	HDR - Université de Pierre et Marie Curie
M. Mathieu BOUET	Examinateur	HDR et chef d'équipe de recherche - THALES
M. Guy PUJOLLE	Directeur de thèse	Professeur - Université de Pierre et Marie Curie
M. Othmen BRAHAM	Encadrant	Docteur et directeur général - VirtuOR

Remerciements

Je souhaite remercier en premier lieu mon directeur de thèse, M.Guy Pujolle, Professeur des Universités pour m'avoir accueilli au sein de son équipe. Je lui suis également reconnaissant pour le temps conséquent qu'il m'a accordé, ses qualités pédagogiques et scientifiques, sa franchise et sa sympathie. J'ai beaucoup appris à ses côtés et je lui adresse ma gratitude pour tout cela.

J'adresse également mes remerciements à mon encadrant de thèse en entreprise, M.Othmen Braham, docteur et directeur technique de VirtuOR , pour son attention sur mes travaux, pour ses conseils avisés et son écoute qui ont été prépondérants pour la bonne réussite de cette thèse.

Je tiens à remercier tous les membres de l'équipe PHARE pour leur aide, leur soutien et pour leur sympathie durant mes trois années de thèse. Les moments que j'ai passé avec les thésards de l'équipe resteront gravés à jamais dans ma mémoire.

Enfin, je remercie toute ma famille, à commencer par mes parents qui m'ont continuellement soutenu durant ma thèse, je remercie également mes deux frères, mes cousins ainsi que mes amis pour leurs encouragements.

Abstract

Over the next five years, the new generation of mobile networks (5G) would face a significant growth of the data volume, exchanged between billions of connected objects and applications. Furthermore, the emergence of new technologies, such as Internet of Things (IoT), autonomous driving and augmented reality, imposes higher performance and quality of service (QoS) requirements. Meeting these requirements, while reducing the Capital and Operation Expenditures (CAPEX/OPEX), are the pursued goals of the mobile operators. Consequently, Telcos define a new radio access architecture, called Cloud Radio Access Network (C-RAN). The C-RAN principle is to centralize, within a pool, the processing unit of a radio interface, named BaseBand Unit (BBU). These two units are interconnected through a Fronthaul (FH) network. In this thesis, we design a new partially centralized C-RAN architecture that integrates a virtualization platform, based on a Xen environment, called Metamorphic Network (MNet). Through this architecture, we aim to : i) implement a pool in which physical resources (processors, memory, network ports, etc.) are shared between virtualized BBUs and other applications ; ii) establish an open FH network that can be used by multiple operators, service providers and third parties to deploy their services and *Apps* closer to the users for a better Quality of Experience (QoE) ; iii) exploit, through the FH, the existing Ethernet infrastructures to reduce CAPEX/OPEX ; and finally iv) provide the recommended network performance for the 5G. In the first contribution, we define a new Xen architecture for the MNet platform integrating the packet-processing framework, OpenDataPlane (ODP), within a privileged Xen domain, called *"Driver Domain (DD)"*. This new architecture accelerates the data packet processing within MNet, while avoiding the physical CPUs overuse by ODP. Thus, virtual CPU cores (vCPU) are allocated within DD and are used by ODP to accelerate the packet processing. This new Xen architecture improves the MNet platform by 15%. In the second contribution, we implement two network solutions within the FH. The first solution consist of deploying a layer 2 network protocol, Transparent Interconnection of Lots of Links (TRILL), to connect multiple elements of our C-RAN architecture. The second solution consists of implementing a Software Defined Network (SDN) model managed by Open Network Operating System (ONOS), a distributed SDN controller that is which is virtualized within BBU pool. Moreover, a network performance comparison is performed between these two solutions.

Keywords

C-RAN, Xen, MNet, Partial centralization, NGFI, ODP, DPDK, OAI, TRILL, SDN, 5G, Functional splitting.

Table des matières

Table des figures

Liste des tableaux

Glossaire

3GPP 3rd Generation Partnership Project.

ADC Analog-to-Digital Converter.
AMC Adaptive Modulation and Coding.
API Application Programming Interface.
ARQ Automatic Repeat Request.
AS Access Stratum.

BBU BaseBand Unit.
BGP Border Gateway Protocol.

CAPEX Capital Expenditure.
CoMP Coordinated Multipoint.
CP Control Plane.
CPRI Common Public Radio Interface.
CPU Central Processing Unit.
C-RAN Cloud Radio Access Network.
CRC Cyclic Redundancy Check.
CSI Canal State Information.
CWDM Coarse Wavelength Division Multiplexing.

DAC Digital-to-Analog Converter.
DD Driver Domain.
DNA Direct NIC Access.
Dom0 Domain 0.

DomU Domain User.
DP Data Plane.
DPDK Intel Data Plane Development Kit.
DRB Designated Routing Bridge.
DWDM Dense Wavelength Division Multiplexing.

EAL Environment Abstraction Layer.
ECMP Equal Cost MultiPath.
EPC Evolved Packet Core.
ESADI End-Station Address Distribution Information.
ETSI European Telecommunications Standards Institute.

FAI Fournisseurs d'Accès Internet.
FDD Frequency Division Duplexing.
FEC Forward Error Correction.
FFO Fractional Frequency Offset.
FH Fronthaul.
FIFO First In First Out.
ForCES Forwarding and Control Element Separation.

GENEVE Generic Network Virtualization Encapsulation.
GPU Graphic Processing Unit.
GSMA Global System for Mobile Communications Association.

HARQ Hybrid Automatic Repeat Request.
HeNB Home eNodeB.
HMAC Keyed-Hash Message Authentication Code.
HSS Home Subscriber Server.
HVM Hardware-assisted Virtualized Machine.

I/O Input/Output.
IANA Internet Assigned Numbers Authority.
IETF Internet Engineering Task Force.
IMS IP Multimedia Subsystem.
IOCTL Input-Output Control.
IoT Internet of Things.

IQ In-phase/Quadrature.
IS-IS Intermediate System to Intermediate System.
ITU International Telecommunication Union.

KPI Key Performance Index.
KVM Kernel-based Virtual Machine.

LIP6 Laboratoire d'Informatique de Paris 6.
LTE-A Long Term Evolution Advanced.
LuaJIT Lua Just In Time.

MAC Media Access Control.
MEC Mobile Edge Computing.
MIMO Multiple-Input Multiple-Output.
MME Mobility Management Entity.
MMU Memory Management Unit.
MNet Metamorphic Network.
MP Management Plane.
MPLS MultiProtocol Label Switching.

NAS Non-Access Stratum.
NFV Network Function Virtualization.
NGFI Next Generation Fronthaul Interface.
NGMN Next Generation Mobile Networks.
NIC Network Interface Card.
NOS Network Operating System.
NTP Network Time Protocol.
NUMA Non Unifom Memory Access.

OAI Open Air Interface.
OAM Operations, Administration and Management.
OBSAI Open Base Station Architecture Initiative.
ODP OpenDataPlane.
ONF Open Networking Foundation.
ONOS Open Network Operating System.
OPEX Operational Expenditure.

OPNFV Open Platform for Networks Functions Virtualization.
ORI Open Radio Interface.
OS Operating System.
OSI Open Systems Interconnection.
OTN Optical Transport Network.
OVS OpenVSwitch.
OVSDB OpenVSwitch DataBase.

PCI Peripheral Component Interconnect.
PDCP Packet Data Compression Protocol.
PHY Couche Physique.
PMD Poll Mode Driver.
POF Protocol-Oblivious Forwarding.
ppm Part Per Million.
PTN Packet Transport Network.
PTP Precision Time Protocol.
PV Paravirtualization.

QoE Quality of Experience.
QoS Quality of Service.

RAB Radio Access Bearer.
RAN Radio Access Network.
RAU Radio Aggregation Unit.
RB Routing Bridge.
RCC Radio Cloud Center.
RE Radio Equipment.
REC Radio Equipment Control.
RF Radio Frequency.
RLC Radio Link Control.
RoE Radio over Ethernet.
RoHC Robust Header Compression.
RRC Radio Resource Control.
RRH/RRU Remote Radio Head/Unit.
RRM Radio Resource Management.

RRS Remote Radio System.
RTT Round Trip Time.

SDK Software Development Kit.
SDN Software Defined Network.
SDR Software Defined Radio.
SISO Single-Input Single-Output.
SoC System-On-Chip.
SPGW Serving/PDN Gateway.
SRC Sample Rate Conversion.
SR-IOV Single-Root Input/Output Virtualization.
SyncE Synchronous Ethernet.

TCP Transmission Control Protocol.
TLS Transport Layer Security.
TRILL Transparent Interconnection of Lots of Links.

UDP User Datagram Protocol.
UE User Equipment.

VLAN Virtual Local Area Network.
VM Virtual Machine.
VMM Virtual Machine Monitor.
VNF Virtual Network Function.
VNI VXLAN Network Identifier.
VTEP VXLAN Tunnel End Point.
VXLAN Virtual Extensible LAN.

WDM Wavelength Division Multiplexing.
WNC Wireless Network Cloud.
WWW World Wide Web.

XCI Xhaul Control Infrastructure.
XFE Xhaul packet Forwarding Element.

Chapitre 1

Introduction

Sommaire

Depuis sa popularisation au début des années 90, grâce à la naissance de World Wide Web (WWW), l'internet n'a cessé de se développer au fil des années pour devenir, de nos jours, omniprésent avec près de 3.9 milliards d'utilisateurs répartis dans le monde en 2017, soit 51.7% de la population mondiale [1]. Alors qu'il s'opérait principalement par les voies filaires et satellitaires, l'accès à internet a évolué durant ces trois dernières décennies, en s'orientant de plus en plus vers la voie sans-fil, grâce à la succession des différentes générations de réseau mobiles (2.5G/3G/4G), à l'évolution de la technologie Wi-Fi et à la démocratisation des équipements nomades tels que, les smartphones, tablettes, navigateurs GPS, etc. De ce fait, selon l'édition 2017 du rapport de l'économie mobile de la Global System for Mobile Communications Association (GSMA), le nombre de personnes ayant souscrit à une offre de téléphonie mobile, devrait atteindre les 5 milliards à la fin de cette année [2]. De plus, les équipements mobiles actuels génèrent et échangent de plus en plus de données, à travers les réseaux sociaux, les applications de partage/streaming vidéo, etc. Le volume de ces données devrait par ailleurs dépasser les 11 Exaoctets (Eo) par mois à la fin de cette année [3]. Cette tendance devrait continuer lors de la prochaine décennie (figure 1.1), en raison de l'orientation du contenu internet vers le format vidéo et

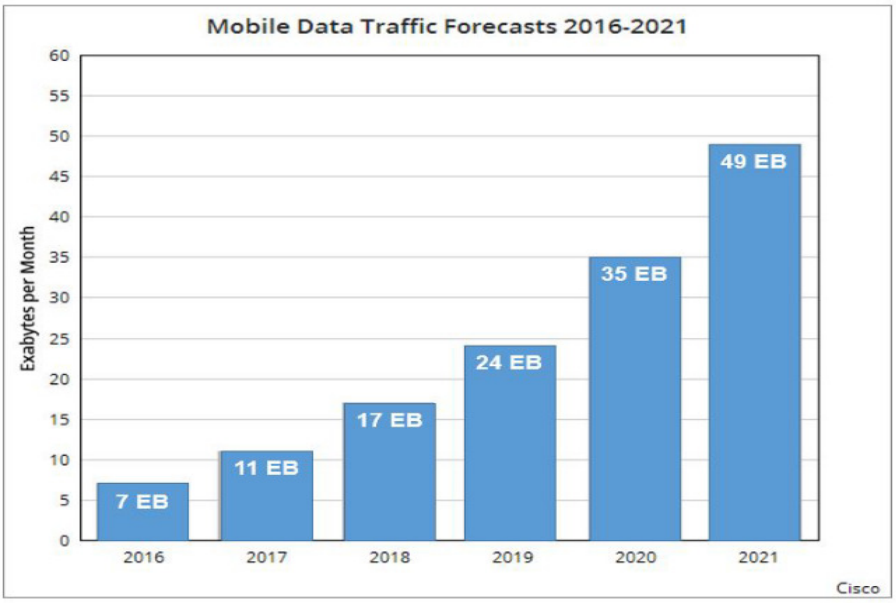

FIGURE 1.1 – Prévisions 2016-2021 du trafic mondial de données mobiles.

en raison de l'apparition de nouvelles solutions IT génératrices et consommatrices d'une grande quantité de données (Internet of Things (IoT), Cloud gaming, etc.). D'autre part, certaines technologies « critiques » requièrent la présence d'une infrastructure réseau fiable et performante, pouvant fournir une très faible latence de transport, un haut débit de transmission et une disponibilité accrue (99,999%). Citons comme exemple la télémédecine et particulièrement la chirurgie assistée, dont la mise en pratique exige une forte disponibilité et une faible latence de la connexion, en raison de son fonctionnement en temps réel.

Bien que l'architecture Long Term Evolution Advanced (LTE-A), actuellement déployée, permet de répondre aux besoins cités, grâce notamment à un délai de transmission réduit à 50 ms ou encore à des débits théoriques descendants/montants (downlink/uplink) de 1 Gbit/s et 500 Mbits/s respectivement, sur des bandes passantes de 100 MHz [4], les réseaux cellulaires devront rapidement évoluer pour s'adapter à la croissance du trafic, aux exigences émises, en terme de qualité de service (Quality of Service (QoS)), par les fournisseurs de contenus/services ainsi qu'aux besoins des utilisateurs, en termes d'accès internet et de qualité d'expérience (Quality of Experience (QoE)). La nouvelle génération (5G) est justement définie pour atteindre ces différents objectifs, avec les contraintes d'atténuation de la consommation d'énergie et de réduction des dépenses d'investissement/exploitation (Capital Expenditure (CAPEX) et Operational Expenditure (OPEX)) [5]. Par conséquent, les réseaux cellulaires 5G devront fournir un débit de données de 10 Gbit/s, une latence inférieure à 1 ms, un nombre d'équipements connectés supérieur à 10000 par cellule, etc. (figure 1.2). Pour cela, les opérateurs et équipementiers télécom semblent s'orienter vers l'adoption de nouvelles technologies de type Network Function Virtualization (NFV) [6], dans les parties Evolved Packet Core (EPC) et Radio Access Network (RAN) de l'infrastructure réseau, qui permettraient de fournir plus de flexibilité dans l'infrastructure du réseau, de réduire les CAPEX/OPEX et de diminuer le temps de déploiement des services

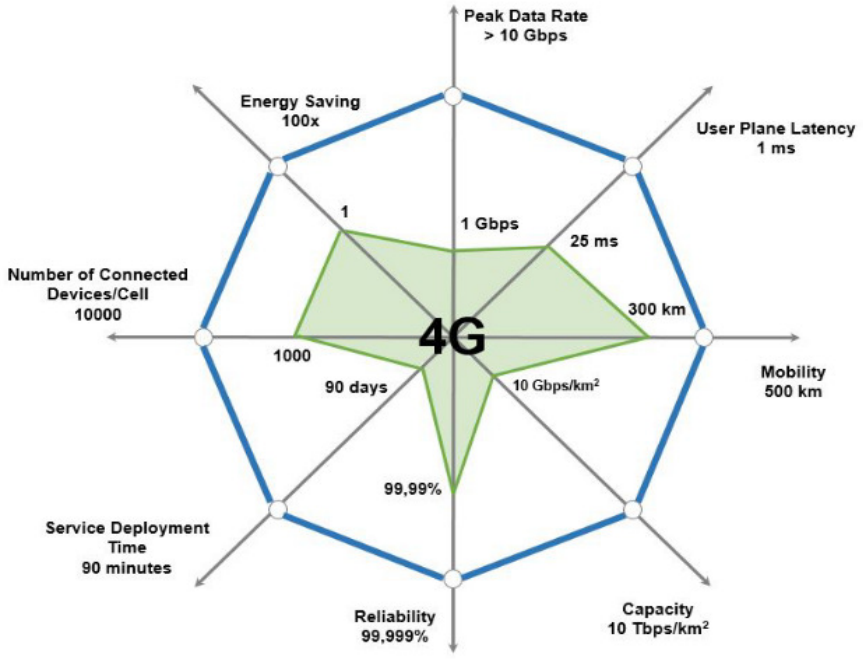

FIGURE 1.2 – Vision globale des objectifs fixés par la 5G à l'horizon 2020.

télécom. Par ailleurs, une nouvelle architecture d'accès radio a été proposée, nommée Cloud Radio Access Network (C-RAN) [7], avec pour objectif de fournir un meilleur traitement des signaux radio, un partage des ressources de calcul/radio optimisé et une efficience énergétique au niveau des stations de base.

Dans cette étude, les deux solutions mentionnées seront associées, pour concevoir une architecture C-RAN, composée d'éléments virtualisés qui seront intégrés dans la plateforme Metamorphic Network (MNet) développée par VirtuOR [8]. Le travail exposé dans ce document fera partie du projet FUI Elastic Network. Nous décrirons dans cette introduction, les motivations et les objectifs de la thèse (Section 1.1) et les détails du projet Elastic Network (Section 1.2). Nous présenterons ensuite les différentes contributions apportées (Section 1.3) et enfin le plan de structuration de ce manuscrit (Section 1.4).

1.1 Motivations et objectifs de la thèse

L'architecture C-RAN redéfinit l'accès radio, grâce à son principe de séparation de l'interface LTE en deux unités, l'unité radio appelée Remote Radio Head/Unit (RRH/RRU) qui constitue la cellule et qui embarque entre autres des fonctions d'amplification et de conversion du signal (analogique-numérique (Analog-to-Digital Converter (ADC)), numérique-analogique (Digital-to-Analog Converter (DAC))) et l'unité de traitement appelée BaseBand Unit (BBU) qui représente l'élément « intelligent »

du réseau d'accès regroupant les fonctions des couches protocolaires (L1/L2/L3) de l'interface radio. Les différentes BBU sont centralisées et virtualisées au sein d'un « Cloud », nommé *pool BBU*, se trouvant à une certaine distance des RRU. Les deux unités du C-RAN sont alors interconnectées à travers un réseau Fronthaul (FH).

L'architecture centralisée du C-RAN présente de nombreux avantages, tels que le partage optimisé des ressources de calcul entre les BBU virtualisées, la consommation d'énergie réduite dans les stations de base ainsi que la gestion améliorée de la mobilité des User Equipment (UE). En contrepartie, la connexion RRH-BBU requiert l'utilisation d'une large bande passante, la garantie d'une très faible latence dans le FH et la synchronisation des horloges entre les deux unités du C-RAN. Fournir de telles conditions implique nécessairement le déploiement de liaisons de fibre optiques point-à-point entre les RRH et les BBU, la mise en place de nouveaux équipements dans l'infrastructure réseau et l'implémentation de certaines techniques de multiplexage telles que Wavelength Division Multiplexing (WDM). Cela peut représenter un coût important pour les opérateurs qui devront alors remplacer certaines parties de leur réseau cellulaire. Par conséquent, trouver un compromis entre des performances optimales et un coût de déploiement réduit, représente le principal défi dans la conception d'une architecture C-RAN.

A travers ce travail de thèse, nous tenterons de relever ce défi, en proposant une nouvelle architecture C-RAN, à faible coût de déploiement, qui pourrait exploiter les équipements ainsi que les liaisons Ethernet, existant dans l'infrastructure d'un réseau cellulaire traditionnel. Pour cela, notre travail sera basé sur la virtualisation des éléments d'un C-RAN au sein d'une plateforme MNet améliorée. Ainsi, les objectifs de notre étude sont les suivants :

— implémenter une nouvelle architecture Xen [9], embarquant une solution d'accélération du traitement de paquets de données, nommée OpenDataPlane (ODP) [10]. Cette architecture représentera l'environnement de virtualisation de MNet et permettra d'héberger les éléments du C-RAN (RRU, BBU, etc.) et d'autres applications virtualisées,
— optimiser l'utilisation des ressources matérielles, notamment celle du processeur (Central Processing Unit (CPU)), au sein de la plateforme MNet, sans impacter le fonctionnement de ODP,
— concevoir une architecture C-RAN partiellement centralisée, dans laquelle la Couche Physique (PHY) est déplacée vers la station de base qui sera alors renommée Remote Radio System (RRS). Par ailleurs, la BBU est désignée comme un Radio Cloud Center (RCC),
— mettre en place un FH switché et ouvert, basé sur le concept Next Generation Fronthaul Interface (NGFI) [11], qui pourrait être partagé entre différents opérateurs, fournisseurs de services et tierces parties, pour permettre un déploiement simplifié de diverses applications (géolocalisation, stockage vidéo, protection, etc.) aux extrémités du réseau mobile,
— intégrer et évaluer les solutions réseau : Software Defined Network (SDN) et Transparent Interconnection of Lots of Links (TRILL) [12], au sein du FH. En outre, une comparaison des performances sera réalisée entre une approche du plan de contrôle centralisée (SDN) et une approche distribuée (TRILL),
— fournir les performances réseau requises pour le C-RAN et la nouvelle génération des réseaux cellulaires (5G),
— assurer la réussite du projet Elastic Network, à travers les objectifs susmentionnés.

1.2 Description du projet Elastic Network

Elastic Network est un projet de recherche, dédié à la conception et à l'implémentation d'une plateforme de test, regroupant les différentes fonctions d'un réseau mobile. Pour ce faire, la virtualisation réseau sera exploitée, dans une architecture LTE, pour simplifier le déploiement des stations de base, appelées également Home eNodeB (HeNB) à l'extrémité du RAN (voire directement chez l'utilisateur), faciliter la mise en place de la partie cœur (EPC) dans l'infrastructure réseau, optimiser l'utilisation des ressources physiques du réseau mobile, pouvoir instancier à la demande les éléments de l'architecture LTE et enfin définir différents cas d'utilisation pour tester et valider une architecture mobile. Par ailleurs, une architecture C-RAN, composée d'éléments virtualisés (vRRU, vRCC, etc.) sera définie et intégrée dans cette plateforme de test.

Le consortium Elastic Network réunit des partenaires industriels, tels que ERCOM et VirtuOR pour les parties virtualisation/interconnexion des fonctions du réseau mobile (VirtuOR), tests et mesure de l'architecture (ERCOM) et des partenaires académiques, tels que EURECOM et le Laboratoire d'Informatique de Paris 6 (LIP6) pour les parties implémentation des composants radio/mobiles (EURECOM), optimisation, gestion et orchestration des fonctions virtualisées (LIP6). Les différents acteurs du projet s'appuieront sur l'utilisation des moyens et ressources fournies par la plateforme d'expérimentation *Com4Innov* [13] dans un environnement LTE.

1.2.1 Objectifs du projet

Le projet Elastic Network devrait atteindre les objectifs suivants :

— développer une architecture de réseau mobile virtualisée de bout en bout, pouvant s'adapter aux différents tests pratiqués par les opérateurs de télécommunication, équipementiers réseau et par les laboratoires de recherche,
— concevoir une architecture C-RAN de référence, basée sur la plateforme logicielle open source Open Air Interface (OAI) développée par EURECOM [14], pour des performances rehaussées, une consommation d'énergie réduite et des CAPEX/OPEX limitées,
— établir une interconnexion réseau fiable et performante, entre les différentes parties du réseau mobile (fronthaul et backhaul), qui devrait s'adresser aux besoins du C-RAN et de la 5G,
— instancier dynamiquement et à la demande, les éléments virtuels du réseau mobile, tels que vHeNB, vRRU et vRCC pour le RAN et Mobility Management Entity (MME), Home Subscriber Server (HSS) et Serving/PDN Gateway (SPGW) pour le réseau cœur,
— implémenter des algorithmes d'orchestration des fonctions C-RAN et LTE, d'allocation dynamique des ressources de traitement (RCC) dans l'architecture C-RAN, pouvant s'adapter à la charge du trafic de données à traiter et enfin de gestion de la migration de machines virtuelles (Virtual Machine (VM)) entre différents serveurs physiques,
— développer un outil logiciel de test, adapté pour les technologies LTE et C-RAN, offrant la possibilité de définir différents cas d'usage pour la validation des architectures mobiles,
— intégrer au sein d'un cloud les différentes parties de la plateforme de test Elastic Network comme illustrée dans la figure 1.3.

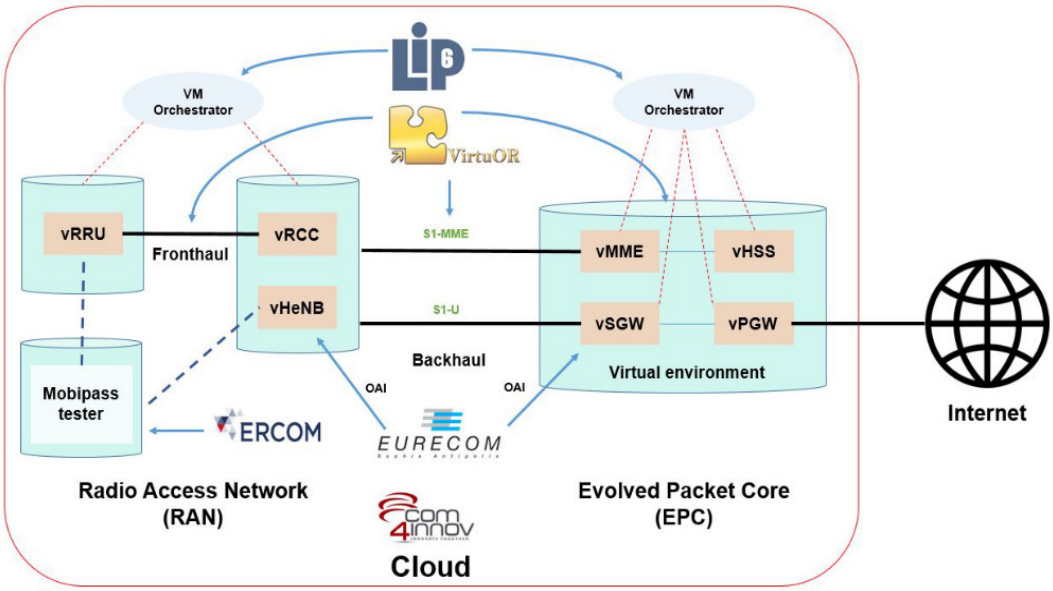

FIGURE 1.3 – Schéma global de l'architecture Elastic Network.

1.2.2 Répartition des tâches

Dans le but d'atteindre les objectifs cités, du projet Elastic Network, les différentes tâches ont été réparties entre les acteurs du projet. Les apports de chaque acteur du projet sont résumés ci-dessous :

- **EURECOM :** fournira l'environnement logiciel OAI, sur lequel l'architecture mobile du projet est basée. OAI implémente l'ensemble des éléments d'un réseau cellulaire 3rd Generation Partnership Project (3GPP), tels que le HeNB et les fonctions MME, HSS, SGW et PGW. De plus, l'équipe d'EURECOM est chargée du développement de la division fonctionnelle, entre les parties radio (RRH/RRU) et traitement (BBU/RCC), au sein de l'architecture C-RAN. Ces deux parties seront ensuite implémentées dans OAI.
- **ERCOM :** fournira la solution de test en charge MOBIPASS qui permet d'émuler plusieurs UE en simultané afin de mesurer la capacité du réseau cellulaire, d'optimiser les performances globales de l'infrastructure et de réduire les OPEX/CAPEX en ajustant le dimensionnement du réseau. Par ailleurs, ERCOM sera chargé de la virtualisation et de la réadaptation de MOBIPASS pour effectuer des mesures sur l'architecture C-RAN proposée dans le projet et enfin sa validation.
- **VirtuOR :** sera chargée de la virtualisation des éléments du réseau cellulaire (HeNB, RRU, RCC, EPC) au sein de la plateforme MNet. VirtuOR s'occupera également de l'interconnexion de ces différents éléments, à travers les réseaux fronthaul (entre la RRU et le RCC) et backhaul (entre le C-RAN/RAN et l'EPC).
- **LIP6 :** développera des algorithmes d'orchestration permettant d'instancier, de manière dynamique et optimale, les multiples fonctions virtualisées du réseau. Ces algorithmes seront par ailleurs adaptés aux architectures LTE et C-RAN.

- ***Com4Innov* :** constituera la plateforme d'expérimentation Cloud, utilisée dans le projet. *Com4Innov* fournit un ensemble d'outils de mesure et de ressources (IP Multimedia Subsystem (IMS), Antennes d'accès LTE, environnements de testbed, etc.) permettant de mettre en place des tests de grande envergure sur l'architecture mobile du projet, pour sa validation.

1.3 Contributions de la thèse

Dans cette thèse, deux principales contributions seront apportées.

1.3.1 Conception d'une nouvelle architecture Xen pour la plateforme MNet

Quatre travaux seront menés dans cette première contribution : **1)** nous présenterons un état de l'art sur les solutions logicielles (libres et propriétaires) d'accélération du traitement de paquets données ; **2)** nous intégrerons le framework ODP au sein de l'environnement Xen de MNet. Nous présenterons les avantages de ODP dans la plateforme, lesquels seront comparés aux autres solutions d'accélération du traitement de paquets pour justifier ce choix en particulier ; *3)* nous réadapterons l'architecture Xen de la plateforme, en implémentant un domaine privilégié, appelé Driver Domain (DD), qui hébergera les librairies et les fonctions de ODP. Cela permettrait d'éviter la surutilisation des cœurs physiques du CPU, en les substituant par des cœurs virtuels (vCPU) au sein du DD ; **4)** nous évaluerons notre architecture Xen à travers des tests de performance et une comparaison sera réalisée avec la précédente architecture.

1.3.2 Conception d'une architecture C-RAN partiellement centralisée

Dans cette seconde contribution, nous accomplirons les tâches suivantes :

— réaliser une présentation détaillée du C-RAN, en décrivant chacun de ses composants, en énumérant ses avantages par rapport l'architecture d'accès radio existante actuellement et en présentant les deux principaux modèles de division de l'interface radio qui sont la centralisation totale et la centralisation partielle,
— détailler l'interface NGFI en décrivant ses éléments, ses différentes couches d'abstraction et ses modèles de division de l'interface radio,
— exposer les solutions TRILL et SDN proposées pour intégrer le fronthaul,
— mesurer les performances de notre architecture C-RAN, en fonction des indicateurs de latence, gigue et débit de données. Une comparaison sera réalisée entre TRILL et SDN.

1.4 Plan de la thèse

Le manuscrit de la thèse sera organisé comme suit : le chapitre 2 présentera les différentes solutions d'accélération du packet processing, dont fait partie ODP. Une présentation de la plateforme

MNet sera également ajoutée. Ce chapitre détaillera la conception de l'architecture Xen ainsi que le processus d'intégration de ODP au sein de MNet. Enfin, les résultats des performances seront exposés. Le chapitre 3 introduira le concept du C-RAN, en donnant ses objectifs, ses composants et les modèles de division de l'interface radio qui lui sont définis. Le chapitre 3 abordera le cœur de notre seconde contribution. Une description de l'interface NGFI ainsi que des solutions TRILL et SDN seront données. Ce chapitre présentera également l'architecture C-RAN proposée et détaillera les performances obtenues par cette dernière. Enfin, une conclusion générale résumera le travail accompli durant cette thèse. Des perspectives seront proposées pour apporter des améliorations à ce travail.

Chapitre 2

Accélération optimisée du traitement des paquets de données au sein de la plateforme MNET

Sommaire

2.1 Introduction

Augmenter la capacité d'un réseau, afin de contenir la charge croissante du trafic de données, représente un défi continuel pour les opérateurs et Fournisseurs d'Accès Internet (FAI). Pour ce faire, ces derniers ont privilégié le surdimensionnement des infrastructures physiques, à travers l'intégration d'équipements puissants (routeur, serveur, etc.), embarquant des processeurs multicœurs, un gros volume de mémoire RAM ainsi que des cartes réseau à forte capacité, pouvant supporter des débits de 10, 40 ou encore de 100 Gbit/s. De plus, un important déploiement de fibre optique (G.650-G.659) et de liaisons Ethernet (IEEE802.3ba, bg, bj et bm) a été réalisé pour le transport des données sur de longues distances et à très haut débit. Néanmoins, ces ressources matérielles sont rarement exploitées à pleine puissance, en raison de la surcharge (overhead) occasionnée par les différentes couches logicielles d'un noyau système, ou encore en raison d'une mauvaise utilisation de la mémoire ou des cœurs CPU présents dans un équipement.

Afin de pallier à ces inconvénients, les équipementiers proposent de nouveaux outils logiciels, appelés également kits de développement (Software Development Kit (SDK)), qui offrent la possibilité de développer des applications de plan de données (Data Plane (DP)) et de traitement de paquets (packet processing) performantes, grâce à une utilisation optimale de l'architecture System-On-Chip (SoC) sous-jacente. Outre ces solutions propriétaires, des développeurs tierces fournissent des environnements et des Application Programming Interface (API) ouverts aux utilisateurs, souhaitant concevoir leur propre plateforme de DP. Deux solutions Open source ont par ailleurs émergées, appelées Intel Data Plane Development Kit (DPDK) [15] et *Netmap* [16], qui fournissent un ensemble de librairies et de drivers dédiés au développement de fonctions d'accélération du traitement de paquets. Récemment, un nouveau projet nommé OpenDataPlane (ODP), a vu le jour. ODP est un ensemble d'API, de librairies et de fonctions, tirant parti des architectures CPU multi-cœur ainsi que des accélérateurs logiciels/matériels sous-jacents pour un traitement accéléré des paquets de données.

Dans la première contribution, ODP sera intégré au sein de la plateforme virtuelle MNet, pour une exploitation optimisée de la capacité d'une interface réseau (Network Interface Card (NIC)) à traiter les flux de données entrants/sortants, grâce à une utilisation améliorée des ressources physiques (CPU et mémoire) et à une exécution indépendante des couches logicielles du noyau Linux. ODP offre de nombreux avantages comparé à DPDK ou Netmap, tels que l'intégration d'un classificateur de flux de données, ou encore la compatibilité de ses librairies avec toutes les cartes réseau existantes sur le marché. Cependant, à l'instar des autres accélérateurs de traitement, ODP présente l'inconvénient de la surutilisation des cœurs CPU embarqués dans l'équipement, ce qui impacte négativement le fonctionnement des applications et des machines virtuelles hébergées, notamment lorsque le nombre alloué de cœurs est réduit. Afin de lever cette limitation, nous proposons une nouvelle architecture Xen pour la plateforme MNet, qui permettrait d'atténuer l'utilisation des ressources CPU, sans altérer les performances obtenues par ODP. Pour ce faire, nous implémenterons ODP au sein d'un domaine privilégié de Xen, appelé *Driver Domain (DD)*, dans lequel des cœurs CPU virtuels (vCPU) seront affectés pour le traitement accéléré des paquets. Ainsi, cela permettrait de réduire la surutilisation des cœurs CPU physiques, provoquée par ODP.

Ce chapitre sera consacré à la description de notre première contribution. Nous présenterons dans

la Section 2.2 les différentes solutions de packet processing existantes dans la littérature. La Section 2.3 décrira la plateforme MNet développée par VirtuOR. La Section 2.4 détaillera l'architecture Xen proposée. La Section 2.5 contiendra les résultats des performances obtenues par notre solution. Enfin, nous terminerons ce chapitre par une conclusion résumant notre contribution.

2.2 Solutions d'accélération du traitement des paquets de données

Un état de l'art des solutions de packet processing existant actuellement dans la littérature fera l'objet de cette section. Nous allons en outre présenter leur architecture, leurs composants ainsi que leur fonctionnement. Enfin, une comparaison sera établie entre les principaux outils d'accélération de traitement de paquets.

2.2.1 OpenDataPlane (ODP)

2.2.1.1 Présentation

OpenDataPlane (ODP) est un projet de développement logiciel qui fournit un environnement de programmation, en langage *C++*, pour des applications de DP. Mis en place en octobre 2013 par Linaro, en partenariat avec plusieurs grands acteurs du secteur informatique (ARM, Broadcom, Cisco, etc.), ODP avait pour objectif initial d'implémenter un outil open source d'accélération de traitement de paquets, pouvant être utilisé sur toutes les architectures de processeur et les NIC disponibles sur le marché. Son lancement s'est effectué finalement en janvier 2015, sous forme d'environnement logiciel, pouvant être déployé sur des distributions Linux ou bien implémenté sur des *bare-metal*.

ODP est constitué d'un ensemble d'API, de fichiers de configuration et de fonctions, s'exécutant dans l'espace utilisateur (*userspace*) de Linux, faisant abstraction des couches matérielles sous-jacentes. ODP peut être utilisé sur différentes architectures SoC et plateformes logicielles, et peut être également couplé aussi bien à des accélérateurs matériels que logiciels, grâce à l'utilisation d'API adaptées. Enfin, ODP fournit des outils de développement d'applications de DP portables et interopérables, pouvant être adaptés à tous les environnements virtuels existants, ou encore être intégrés sous forme de fonctions réseau virtualisées.

2.2.1.2 Architecture

ODP peut être déployé directement sur un environnement bare-metal, ou bien au sein de l'espace utilisateur de Linux (figure 2.1). Un tel déploiement permet de minimiser (voire d'éliminer) le risque d'interférences entre le noyau Linux et les applications ODP en exécution. Des interférences qui se produisent généralement lors d'appels système ou lors de l'utilisation des ressources matérielles de la machine (mémoire, processeurs, etc.).

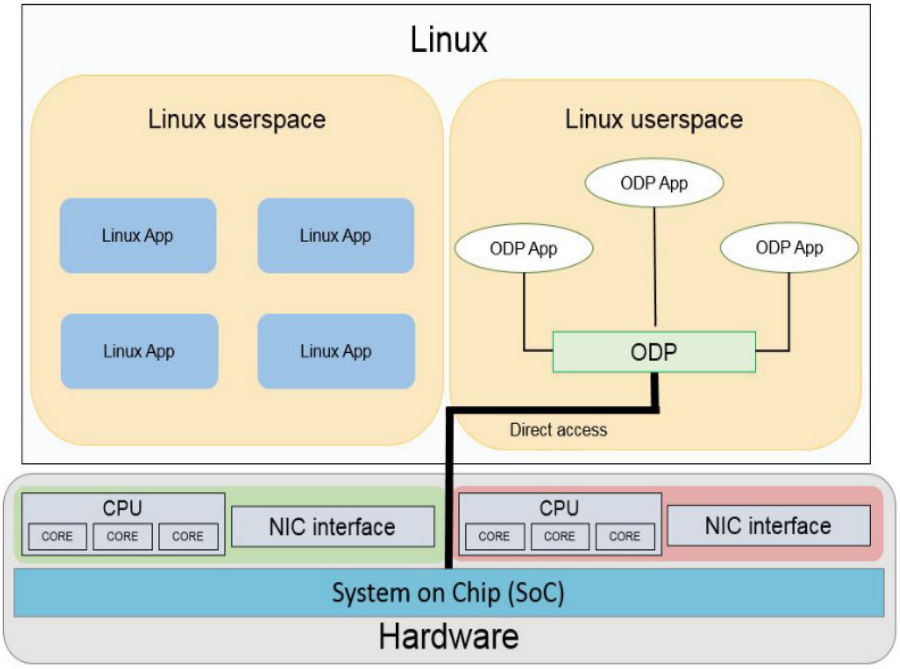

FIGURE 2.1 – Implémentation de OpenDataPlane (ODP) dans Linux.

Afin d'atteindre des performances optimales (haut débit, traitement de paquet accéléré, etc.) sans impacter le fonctionnement global d'une plateforme ou d'un système, les applications ODP accèdent directement aux fonctions (interfaces, registres, etc.) d'un accélérateur matériel, se trouvant sur un SoC. Cet accès direct permet d'éviter une surcharge (overhead) supplémentaire occasionnée par les couches logicielles du noyau. ODP offre également la possibilité d'exploiter efficacement les cœurs du CPU afin d'accélérer le traitement des paquets et ainsi augmenter le débit de données. Nous allons par ailleurs détailler ce point dans cette section.

2.2.1.3 Composants

ODP intègre différents composants et fonctions (figure 2.2), utilisables par les API pour la gestion des ressources matérielles des SoC, pour la gestion et le traitement du trafic de données entrant et pour l'exploitation de l'accélération matérielle/logicielle du traitement des paquets. Nous allons décrire chacun de ces composants :

Buffer

Un buffer est un espace mémoire de taille fixe, ne dépassant pas les 1856 octets, dédié au stockage des métadonnées (identifiant du pool, taille, etc.) et aux paquets de données entrants ou sortants. En

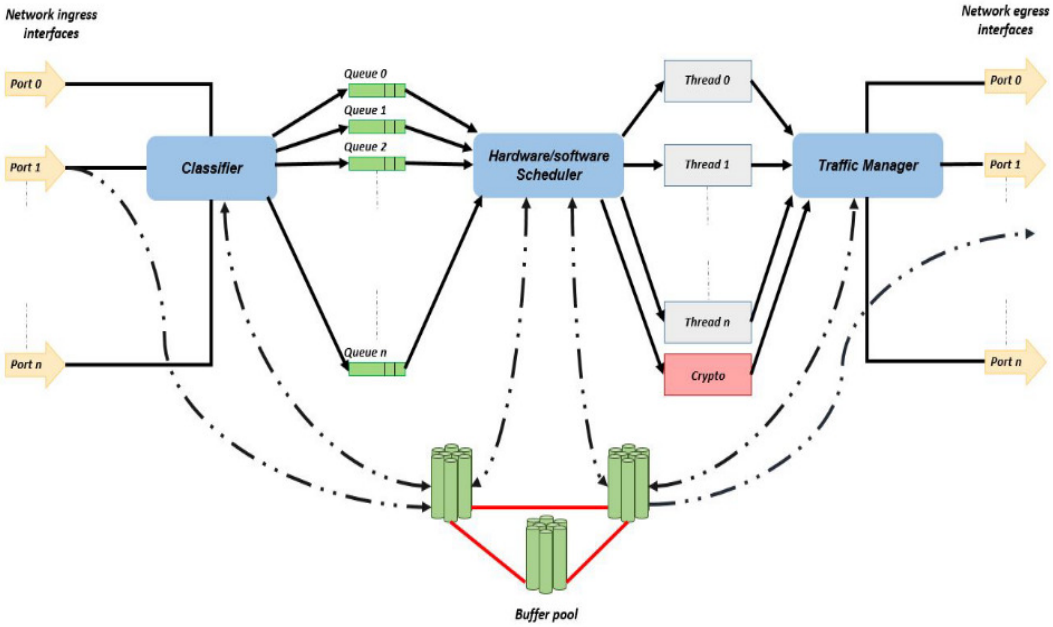

FIGURE 2.2 – Schéma de base du traitement de paquets avec ODP.

outre, les buffers sont regroupés au sein d'un pool de taille limitée (8192 octets) qui est créé à l'initialisation d'une application ODP, pour être ensuite partagé entre les composants de cette application. Une fois l'exécution de cette dernière terminée, l'espace mémoire occupé par le pool est complètement libéré.

Evènement (*Event*)

Un évènement décrit la tâche que va effectuer un thread dans les applications ODP. Un évènement possède un type qui définit à quoi il correspond. Il peut par exemple représenter l'arrivée d'un paquet, lequel doit être traité par l'application, ou bien la réception d'une requête asynchrone. Il peut également représenter les changements opérés au sein des composants de cette application.

Classificateur (*Classifier*)

Relié aux pools de buffers d'une application ODP, le classificateur analyse tous les flux de paquets entrants, stockés dans ces buffers. Chaque paquet est alors classé selon des règles définies par l'utilisateur grâce à des API dédiées. Un flux de paquets peut par exemple être classifié selon le protocole de transport utilisé (Transmission Control Protocol (TCP), User Datagram Protocol (UDP), etc.), ou selon le numéro du Virtual Local Area Network (VLAN) auquel il appartient. Par ailleurs, cette opération génère des évènements au sein de l'application qui sont transférés vers les files d'attentes correspondant à chaque classe. Ces évènements sont par la suite stockés et traités. Bien que le classificateur soit

un composant optionnel dans ODP, son utilisation reste essentielle pour la mise en place d'une QoS, adaptée à chaque application.

File d'attente (*Queue*)

La file d'attente est une structure logicielle ou matérielle, utilisant la politique du premier arrivé, premier servi (First In First Out (FIFO)). Elle est reliée à deux éléments dans ODP qui sont le classificateur d'un côté et le scheduler/thread (selon le modèle utilisé) de l'autre. Chaque file d'attente est assignée à une classe de paquets et stocke de ce fait les évènements (envoyés par le classificateur) correspondant à cette classe en particulier. Un utilisateur pourra également affecter des priorités à une file d'attente selon la classe de paquets à laquelle elle correspond et selon la politique de préemption de trafic mise en place par cet utilisateur.

Toute application ODP pourra accéder aux files d'attente pour y enfiler/désenfiler des évènements grâce à l'utilisation d'un scheduler ou de threads. Néanmoins, il est possible de dédier un ensemble de files à certaines applications critiques qui seront alors les seules à pouvoir y modifier le contenu. Il existe trois types de files d'attente :

- **File parallèle :** ne nécessite pas de synchronisation particulière avec le scheduler ou/et les threads. Les évènements stockés au sein de cette file peuvent être traités en parallèle (grâce à plusieurs threads) et leur ordre est géré par l'application ODP.
- **File atomique :** un thread unique lui est dédié, chacun de ses éléments est traité dans l'ordre et de manière séquentielle par ce thread. Dans ce type de file, un élément ne peut pas être modifié par deux entités à la fois. Il n'est donc pas nécessaire de mettre en place des mécanismes d'exclusion mutuelle dans les applications utilisant ce type de files.
- **File ordonnancée :** où ses éléments peuvent être traités en parallèle par plusieurs threads. Néanmoins, un ordonnanceur doit être utilisé pour réordonner les évènements de la file.

Les files d'attente ODP intègrent également un mécanisme de *batching* qui permet à un thread/ordonnanceur d'enfiler/désenfiler plusieurs éléments à la fois.

Ordonnanceur (*Scheduler*)

Elément spécifique au modèle *Pull*, l'ordonnanceur sert d'intermédiaire entre l'ensemble des files d'attente et les threads (cœurs du processeur). Un scheduler se charge de désempiler les évènements à partir d'une file en prenant en compte sa priorité, afin de les envoyer par la suite vers un thread pour être traités. L'ordonnanceur est de ce fait un élément important dans l'architecture ODP, pouvant prendre jusqu'à cent millions de décisions à la seconde.

Thread

Le thread représente l'unité de programmation fondamentale dans ODP et peut être exécuté en tant que processus Linux ou en tant que thread POSIX (*pthread*). Le nombre de threads utilisé par une

application ODP dépend du nombre de cœurs CPU alloué à cette dernière. Ainsi, pour chaque cœur CPU, un thread est créé. De plus, ODP permet de créer deux types de threads, un thread d'exécution (*worker*) et un thread de contrôle, lequel est chargé de la supervision du premier type.

La fonction principale d'un thread est le traitement du paquet de données, correspondant à la description de l'évènement transmit par l'ordonnanceur. Une fois le traitement terminé, le paquet est envoyé vers le gestionnaire de trafic.

Gestionnaire de trafic (*Traffic manager*)

Le gestionnaire de trafic procède à différentes tâches de contrôle du trafic de données, de gestion de la QoS et de la mise en forme des paquets. Relié à l'ensemble des threads, le gestionnaire représente le dernier maillon de la chaîne de traitement de paquets utilisé par ODP.

Timer

Elément fréquemment utilisé par les applications ODP, le timer permet d'effectuer plusieurs fonctions, telles que la mise en place d'un intervalle de retransmission de paquets, la détection de l'inactivité d'un utilisateur ou encore la limitation du temps d'exécution d'un processus. Ces applications embarquent également des fonctions de configuration de timers, de requêtes et d'annulation de temporisation. Elles peuvent ainsi exécuter des millions de timers en parallèle pour le besoin de leur fonctionnement. Il existe deux types de timers dont l'utilisation varie selon le besoin d'une application :

- **Timer permanent :** présente la particularité de ne jamais expirer. Néanmoins, sa valeur peut être annulée ou réinitialisée par l'application.
- **Timer temporaire :** possède une durée de vie limitée, allant de quelques microsecondes à plusieurs heures.

Service de cryptographie (*crypto*)

ODP fournit des API de cryptographie utilisées comme services par certains protocoles réseau (p. ex. IPsec) pour établir leurs communications. Ces services peuvent être le chiffrage, la vérification de l'intégrité des authentifications et des données via l'utilisation du Keyed-Hash Message Authentication Code (HMAC) [17] ou encore la génération aléatoire de nombres.

Pour accéder à ces services, une session doit être créée et dont les paramètres seront partagés par tous les paquets traités par cette dernière. ODP supporte par ailleurs les sessions synchrones et asynchrones. Une fois la session créée, ces services cryptographiques peuvent être appliqués sous forme d'opérations sur les paquets traités par l'application. Les arguments d'une opération spécifient, pour chaque paquet, les champs qui doivent être encryptés, décryptés et authentifiés.

2.2.1.4 Modèles d'exécution

ODP définit deux modèles d'exécution pouvant être utilisés par les applications selon les tâches auxquelles elles procèdent. Ces deux modèles sont le « *Pull Model* » et « *Push Model* » qui sont illustrés respectivement sur les figures 2.3 et 2.4.

Modèle Pull (*SCHED*)

Le modèle *Pull* intègre un scheduler ODP reliant les files d'attente aux threads. Cet élément sélectionne les paquets (se trouvant dans le pool de buffers) correspondant aux évènements stockés dans les files d'attente et les envoie vers les threads afin d'être traités. L'utilisation d'un scheduler offre les avantages suivants :

— possibilité de grouper un certain nombre de files d'attente afin de les associer à un ou plusieurs threads pour le traitement,
— mise en place de politiques de préemption (p. ex. selon le protocole de transport utilisé) pour le traitement des paquets de données dans certaines applications,
— synchronisation des traitements d'évènements entre différentes files d'attentes, entre l'ensemble des threads et des files ou au sein d'une même file d'attente. Cette synchronisation est notamment nécessaire lors de l'utilisation de files d'attente parallèles par l'application ODP,
— gestion des différents types de files d'attentes (parallèles, atomiques et ordonnancées) au sein d'une même application,
— préservation de l'ordre des paquets de données traités, en particulier dans les files d'attente ordonnancées, grâce à l'utilisation du mécanisme « *ordered lock* » qui permet de synchroniser les évènements devant être traités séquentiellement sur différents threads.

Modèle Push (*POLL*)

Moins complexe dans sa conception, le modèle *Push* n'utilise aucun scheduler. Les files d'attente sont alors directement reliées aux threads et leur gestion se fait directement par l'application. Ce modèle est utilisé sur des applications de traitement « direct » des paquets entrants, ne nécessitant pas la mise en place d'une politique de priorisation ou bien d'une QoS sur son trafic. En outre, le modèle *Push* peut être associé à des SoC, utilisant leur propre scheduler matériel ou logiciel. L'avantage de ce modèle est sa flexibilité dans le traitement des paquets, en comparaison avec le modèle *Pull*. Néanmoins, son utilisation reste limitée et offre peu de fonctionnalités.

2.2.1.5 Interfaces de programmation applicatives (API)

ODP fournit un ensemble hétérogène d'API permettant à un utilisateur de développer des applications de packet processing, adaptées à tous les SoC et les plateformes virtuelles existantes actuellement sur le marché. Nous pouvons répartir ces API en deux catégories décrites comme suit :

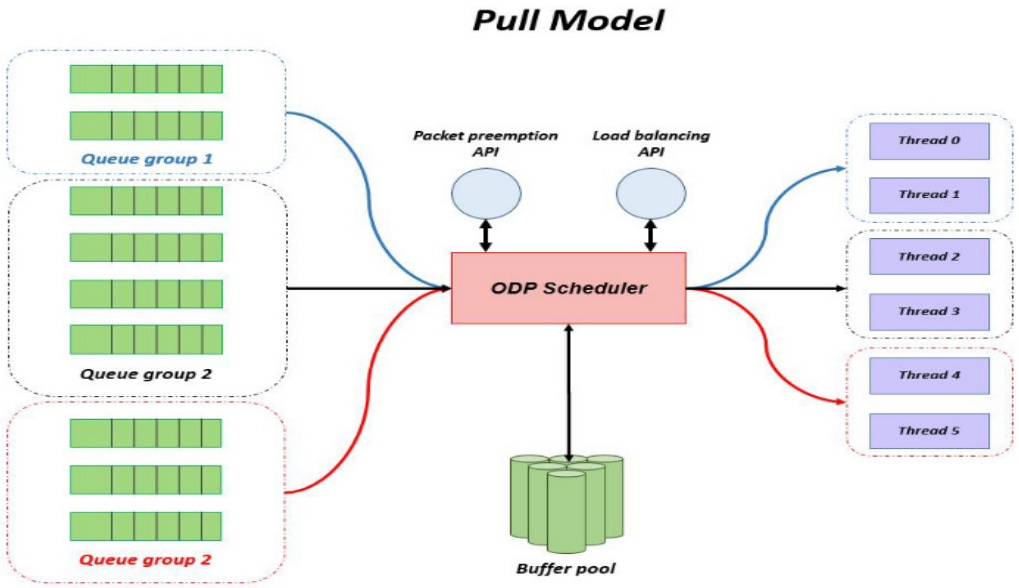

FIGURE 2.3 – Modèle d'exécution *Pull*.

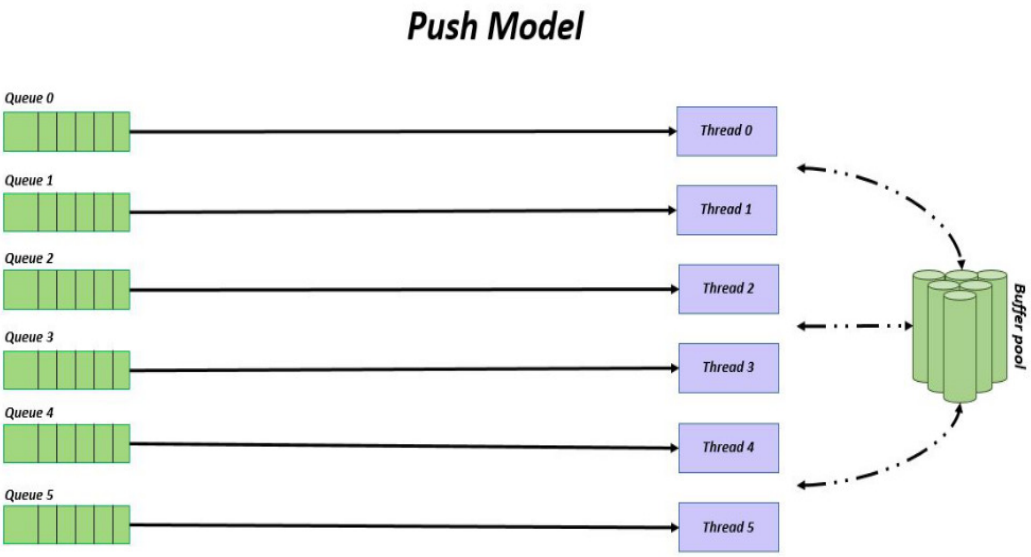

FIGURE 2.4 – Modèle d'exécution *Push*.

- **API synchrone :** s'exécute sur une période courte et finie. Le résultat retourné par ce type d'interfaces indique le succès ou l'échec de son exécution. Les API synchrones sont généralement utilisées pour des opérations nécessitant un temps de réponse court et en temps réel. Ex. lire le compteur de cycles CPU, enfiler/désenfiler des éléments à partir d'une file, etc.

- **API asynchrone :** à l'inverse des API synchrones, les API asynchrones sont utilisées pour des opérations nécessitant un temps d'exécution long ou indéterminé. Les applications ODP génèrent ces interfaces à partir d'appels de fonction et les réponses sont retournées sous forme de messages. En l'absence de traitement, les API asynchrones se mettent en attente active et libèrent de ce fait une partie des ressources mémoire et CPU qu'elles ont consommées. Les fonctions de gestion d'interfaces e/s et d'accélération utilisent notamment ce type d'API.

ODP définit également un ensemble d'API de gestion que nous présentons ci-dessous :

- **API de gestion des ressources :** offre la possibilité d'allouer et de configurer les ressources matérielles et virtuelles (cœurs CPU, interfaces réseau, etc.) disponibles pour être utilisées par les applications ODP.
- **API de gestion de la mémoire :** permet d'allouer des zones mémoire pour chaque élément de l'application ODP. Ces API permettent également de gérer la mémoire partagée entre les éléments ODP, notamment les pools de buffers.
- **API de gestion des threads :** crée et gère des threads logiques qui seront utilisés par l'application pour le packet processing. Le nombre de threads pouvant être créé dépend notamment du nombre de cœurs alloué à cette application. Par ailleurs, les API de gestion des threads ne spécifient pas le modèle d'exécution qui sera utilisé.
- **API de gestion des évènements :** génère et définit des évènements correspondant à des actions produites (arrivée d'un paquet, appel systèmes, etc.) qui seront ensuite stockées dans les files d'attentes.
- **API de gestion des paquets (*Packet I/O*) :** permet aux applications de recevoir ou de transmettre des paquets de données à travers des interfaces logiques de e/s connectées aux ports réseau physiques.
- **API de gestion des classes de trafic :** permet aux applications, à travers le classificateur, de définir et implémenter des politiques de classification du trafic de données, selon différents critères (protocole utilisé, identifiant de VLAN, interface réseau d'entrée, etc.). Une fois classés, les paquets sont ensuite groupés en flux.

2.2.2 Data Plane Development Kit (DPDK)

2.2.2.1 Présentation

Data Plane Development Kit (DPDK) est une solution logicielle open source, développée par *Intel* et *6Wind*, qui comprend un ensemble de librairies et de drivers, s'exécutant dans l'espace utilisateur (*user space*) de Linux. L'objectif principal de DPDK est d'offrir un environnement adapté à la programmation d'applications de DP de haute performance sur des plateformes *Intel x86*. En outre, DPDK fournit une accélération du traitement des paquets et une diminution de la surcharge en terme de trafic dans un équipement réseau. Conçu pour s'exécuter sur des cartes réseau *Intel* (82540, X722, FM10420, etc.), DPDK peut maintenant être supporté par une multitude d'interfaces réseau, allant des cartes *Cisco UCS Virtual Interface Card* aux cartes *Amazon Elastic Network Adapter*, ou même des solutions de paravirtualisation telles que Xen ou QEMU [18]. Enfin, DPDK peut également être associé

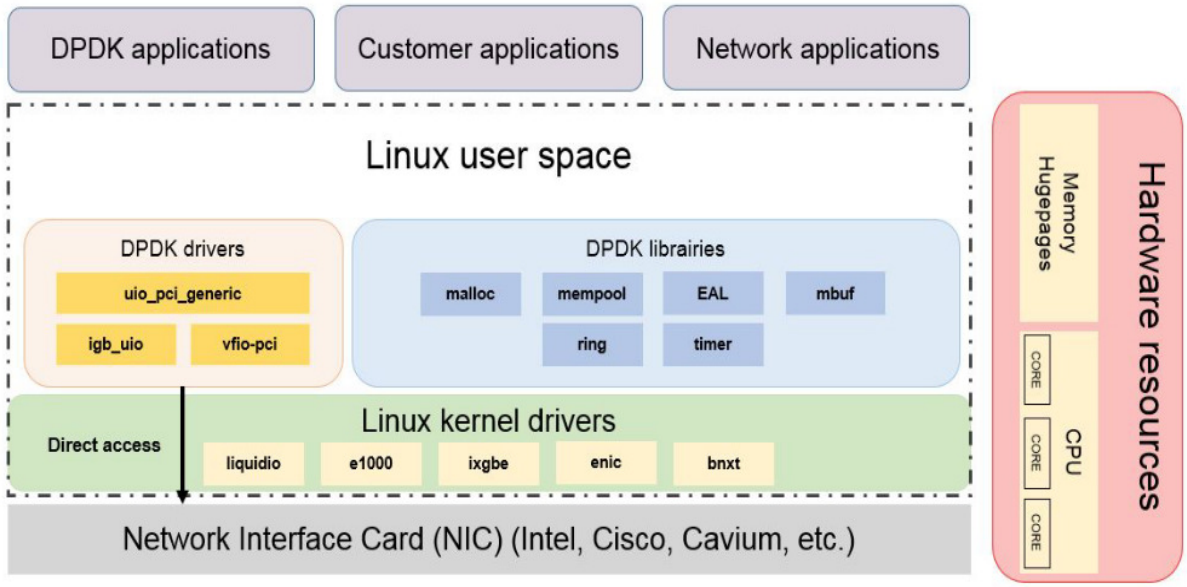

FIGURE 2.5 – Composants de Data Plane Development Kit (DPDK).

à des frameworks et des applications réseau, tels que OpenVSwitch (OVS) et ODP.

2.2.2.2 Composants

Nous allons décrire dans cette partie l'ensemble des fonctions composant DPDK, développées en *C/Python* et pouvant être installées sur des architectures *i686, x86_64* ou bien *ppc_64*.

2.2.2.2.1 Couche d'abstraction de l'environnement (Environment Abstraction Layer (EAL))

DPDK intègre une couche d'abstraction de l'environnement (EAL) qui fournit une interface de gestion et d'accès aux ressources matérielles d'un équipement réseau. Les tâches réalisées par cette couche sont données ci-dessous :

— chargement et exécution des applications DPDK,
— assignation des unités d'exécution à des cœurs CPU,
— allocation des zones mémoire à travers l'utilisation d'une fonction *mmap()*,
— abstraction des adresses Peripheral Component Interconnect (PCI), en fournissant une interface d'accès à l'espace d'adressage PCI,
— utilisation de fonctions de traçage d'erreurs et de débogage,
— utilisation de fonctions de verrous et de synchronisation tels que les *spinlocks*,
— identification des caractéristiques du processeur et de sa compatibilité avec les fonctions DPDK compilées,

— gestion des interruptions système,
— utilisation de fonctions d'alarme.

EAL utilise un ensemble de fonctionnalités nécessaires pour l'exécution des tâches présentées précédemment. Ces fonctionnalités sont également décrites dans cette partie.

Multi-processus

EAL fournit un modèle de développement supportant le multi-processing, basé sur un mécanisme qui permet à un certain nombre de processus de s'exécuter de manière collective, dans le but d'effectuer un traitement efficace des paquets. En outre, EAL apporte de nouvelles fonctionnalités, telles que la génération de deux types de processus, possédant leurs propres permissions et partageant la même ressource mémoire (mais utilisant différentes *hugepages*). Ces deux types de processus sont présentés ci-dessous :

- **Processus primaire :** possède toutes les permissions sur la mémoire partagée entre les processus et peut notamment initialiser cette mémoire lors de son exécution.
- **Processus secondaire :** n'offre pas la possibilité d'initialiser la mémoire partagée mais peut être associé à une mémoire pré-initialisée, dans laquelle il est possible d'y allouer des objets.

Au lancement d'une application DPDK, un processus primaire est nécessairement exécuté afin de configurer la mémoire partagée. Les processus secondaires seront alors en mesure de s'exécuter après cette étape d'initialisation. EAL permet également d'utiliser les différents modèles de déploiement de processus présentés ci-dessous :

- **Modèle symétrique :** génère un ensemble de processus (un seul processus primaire et le reste des processus seront secondaires) possédant une charge de travail équivalente.
- **Modèle asymétrique :** dans ce modèle, le processus primaire agit comme load-balancer qui répartit la charge de traitement entre les différents processus secondaires s'exécutant dans l'application.
- **Modèle indépendant :** dans ce modèle, les processus s'exécutent indépendamment l'un de l'autre.
- **Modèle de groupes indépendants :** il est possible de rassembler un certain nombre de processus en un seul groupe. Chaque groupe fonctionnera alors de manière indépendante par rapport aux autres groupes.

Module Xen (*rte_dom0_mm*)

DPDK est compatible avec une multitude de plateformes virtuelles et paravirtuelles, allant de *VMware ESXI* jusqu'à Xen. Néanmoins, le partage de mémoire en *hugepages* n'est pas supporté par l'environnement Xen. Pour pallier à cela, EAL fournit un module de noyau Linux appelé *rte_dom0_mm*, qui apporte un moyen de gérer l'espace mémoire au niveau du *Domain 0 (Dom0)* de Xen. EAL fournit également une interface Input-Output Control (IOCTL), permettant d'allouer de la mémoire et

de cartographier les segments de cette dernière, en récupérant leurs informations à partir du module *rte_dom0_mm*.

Evènements d'interruptions dans le userspace

EAL crée un thread permettant de détecter l'arrivée ou bien l'envoi de paquets au niveau des interfaces d'e/s réseau. A la réception d'un paquet, EAL déclenche un évènement appelé *RX* qui permet de « réveiller » le fonctionnement de l'application, dans le cas de l'utilisation du mode d'exécution « *idle* ».

Zones et segments mémoire (*memzone*)

Dans DPDK, les portions contiguës de la mémoire sont décrites par EAL comme zones mémoire (*memzone*), identifiées par un nom unique. Les *memzone* peuvent être réservées selon un alignement spécifique des adresses mémoire ou selon la taille définie pour les *hugepages*.

Multi Thread POSIX (*pthread*)

EAL offre la possibilité d'associer plusieurs *pthreads* à un seul et même cœur CPU. Pour cela, EAL exploite les cycles « *idle* », où un cœur CPU n'effectue aucun traitement. La fonction va alors affecter ce temps libre à l'exécution d'un *pthread* en particulier. Ainsi, il est possible d'alterner l'exécution entre plusieurs *pthreads* sur un même cœur. Ce mécanisme complexe nécessite néanmoins la gestion du changement de contexte lors d'un switch entre les *pthreads*. Par ailleurs, afin d'optimiser l'utilisation des ressources CPU, un *pthread* sera associé non pas à un seul mais à plusieurs cœurs à la fois.

2.2.2.2.2 Librairies

DPDK intègre un ensemble de librairies utilisables par les applications, pour un traitement performant des paquets de données. Le schéma de ces librairies est donné dans la figure 2.6.

Gestionnaire d'anneau (*librte_ring*)

Un anneau est une structure de données FIFO de taille fixe, possédant un identifiant unique et permettant de gérer l'ensemble des files d'attente dans une application DPDK, sans l'utilisation de mécanismes de verrou. Un anneau fournit deux fonctions de manipulation de files d'attente. Une fonction « *bulk* » qui permet d'enfiler/désenfiler un nombre défini d'objets (pouvant échouer lorsque le nombre défini est trop élevé) et une fonction « *burst* » qui se déclenche en cas d'échec de la fonction *bulk* et qui permet d'enfiler/désenfiler le plus grand nombre d'objets possible.

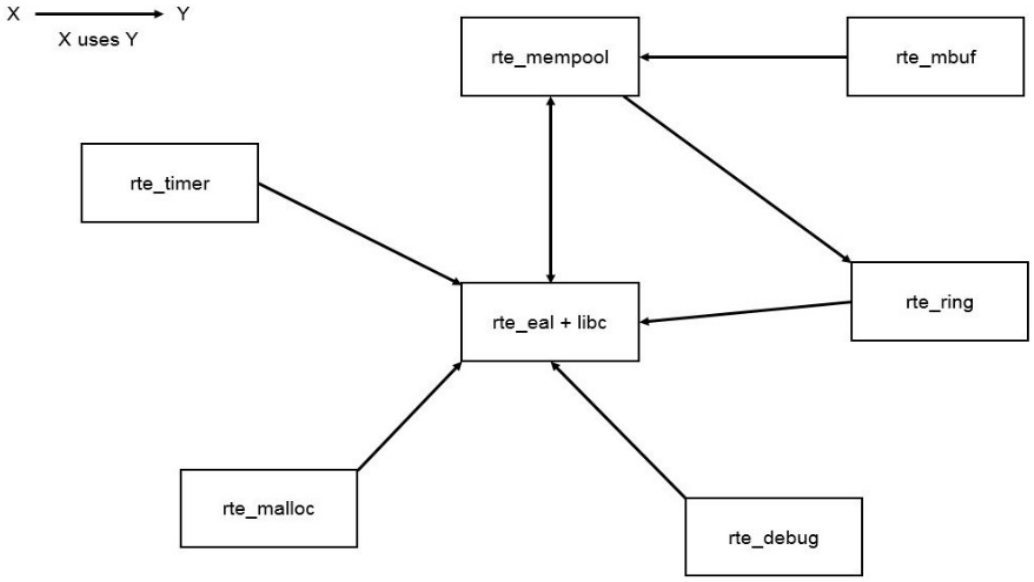

FIGURE 2.6 – Schéma des librairies DPDK.

Gestionnaire de pool de mémoire (*librte_mempool*)

Un pool gère l'allocation d'objets de taille fixe dans la mémoire. Il est identifié par un nom et utilise un assistant « *mempool* » pour stocker les objets inutilisés dans un anneau. Un autre assistant est utilisé par le gestionnaire de pool, permettant d'aligner la mémoire en ajoutant un « rembourrage » (*padding*) entre chaque objet stocké. Cet alignement apporte notamment une grande amélioration des performances d'une application DPDK en partageant le traitement des objets entre les différents canaux de la RAM.

Le gestionnaire de pool affecte également un cache pour chaque cœur de processeur. Ces caches stockent les nombreuses requêtes d'accès aux pools envoyées par les différents cœurs CPU. Ainsi, chaque cœur CPU possèdera un accès total à son espace de stockage dans le pool, à travers l'utilisation de verrous.

Gestionnaire de buffer de paquets de données (*librte_mbuf*)

La librairie *mbuf* offre la possibilité de créer, d'allouer ou de libérer des buffers, servant à stocker les paquets de données reçus par l'application DPDK ainsi que leurs métadonnées (type du paquet, longueur, offset, etc.). Pour ce faire, la librairie *mempool* est utilisée pour la gestion de l'espace mémoire. Il existe deux méthodes de stockage pour les paquets dans DPDK :

- **Méthode 1 :** consiste à stocker un paquet et ses métadonnées dans un seul buffer. Dans ce cas précis, une zone mémoire de taille fixe est affectée pour le stockage de ces données. L'avantage

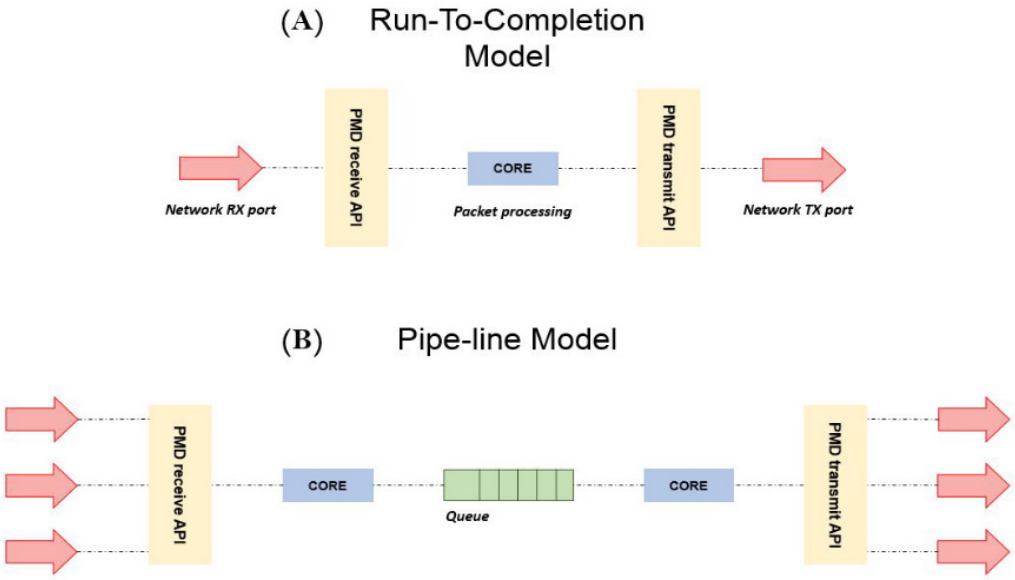

FIGURE 2.7 – Modèles de traitement de paquets de DPDK.

de cette méthode est l'utilisation d'une seule opération pour allouer/libérer les données d'un paquet dans la mémoire.

- **Méthode 2 :** consiste à séparer les paquets de leurs métadonnées pour être stockés dans deux buffers distincts. Cette méthode présente l'avantage d'être plus flexible et permet d'accéder ou de modifier les métadonnées sans manipuler le paquet correspondant.

Les applications DPDK utilisent généralement la première méthode pour constituer une chaîne de buffers grâce à l'utilisation de pointeurs. Cette chaîne permet notamment de lier les paquets appartenant au même flux de données, en exploitant les informations de contrôle présentes dans les métadonnées des paquets. En outre, la librairie *librte_mbuf* fournit différentes API et fonctions, servant à créer/allouer/libérer des buffers de données, d'accéder à certaines informations du paquet et aux pointeurs de la chaîne, d'ajouter ou de supprimer des données, etc.

Gestionnaire de timer (*librte_timer*)

La librairie *librte_timer* fournit aux applications DPDK une interface de création/relance/suppression de timer, permettant une exécution asynchrone de certaines fonctions. Ces timers peuvent être utilisés une seule fois ou à plusieurs reprises. Ils peuvent également être chargés par un cœur CPU et exécutés par un autre cœur. Enfin, pour optimiser ses performances, une application aura la possibilité de désactiver un timer inactif.

Un timer est défini par un statut représenté par un couple (état, propriétaire). L'attribut propriétaire (*owner*) correspond à l'identifiant du cœur CPU (*core id*) qui génère le timer. En outre, un état peut

prendre 4 valeurs décrites comme-suit :

- **STOPPED :** le timer n'appartient à aucun cœur CPU.
- **CONFIG :** le timer appartient à un cœur CPU et ne peut pas être modifié par un autre cœur.
- **PENDING :** le timer peut être modifié par tous les cœurs CPU appartenant à la même liste que celle du cœur créateur.
- **RUNNING :** le timer ne peut pas être modifié par les cœurs appartenant à la liste.

2.2.2.2.3 Drivers

DPDK utilise des drivers « *Poll Mode* » (Poll Mode Driver (PMD)) compatibles avec des cartes réseau de 1, 10 ou 40 Gbit/s. Ces drivers consistent en un ensemble d'API et de modules Linux, s'exécutant dans le userspace et permettant de configurer les interfaces réseau des équipements, ainsi que leurs files d'attente. En outre, PMD possède un accès direct aux ports e/s de cette interface, ce qui permet d'outrepasser les drivers du noyau Linux et ainsi d'éviter l'ajout d'une charge supplémentaire apportée par ces derniers au traitement du trafic de données.

Modèles d'exécution

DPDK fournit deux modèles d'exécution pour ses applications de traitement de paquets :

- **Modèle *run-to-completion* :** chaque cœur CPU logique est relié à un port RX (réception) et à un port TX (transmission) à travers une API de réception du PMD. Ce cœur va traiter chaque paquet de données transitant par l'équipement à travers l'exécution d'une boucle de packet processing. Ce traitement se fait en trois étapes : le cœur CPU récupère d'abord les paquets entrants à travers l'API de réception, ensuite chaque paquet est traité de manière séquentielle et une fois le traitement terminé, il envoie ces paquets vers l'API de transmission du PMD pour qu'ils soient acheminés vers leur destination. Le modèle *run-to-completion* (figure 2.7(A)) est également appelé modèle synchrone.
- **Modèle *pipe-line* :** dans ce modèle, un premier cœur CPU est assigné à un ou plusieurs ports RX afin de gérer la réception (à travers une API PMD de réception) et le stockage des paquets entrants dans des files d'attente, en attendant leur traitement. Un deuxième cœur, connecté à ces files d'attente à travers un anneau, est alors assigné au traitement des paquets. Une fois le traitement terminé, le cœur CPU numéro 2 envoie les paquets à travers des anneaux vers l'API de transmission du PMD. Contrairement au modèle *run-to-completion*, le modèle *pipe-line* (figure 2.7(B)) s'exécute de manière asynchrone.

Enfin, dans le but d'optimiser les performances de packet processing, DPDK intègre un système d'accès à la mémoire non uniforme (Non Unifom Memory Access (NUMA)) pour une meilleure utilisation de la mémoire par les cœurs du processeur.

Identification et configuration des équipements

Lors de son initialisation, DPDK exécute une fonction de détection/énumération des interfaces réseaux connectées. Deux identifiants sont alors attribués à l'interface : un ***index***, utilisé pour désigner les ports (RX/TX) aux fonctions des API du PMD, ainsi qu'un ***nom***, pour désigner les ports à la console de messages, d'administration et de débogage. La configuration des ports de l'interface réseau se fait à travers les opérations suivantes :

— allocation des ressources PCI,
— réinitialisation de l'équipement à un état par défaut connu par l'application,
— initialisation de la couche physique et de liaison,
— initialisation des compteurs statistiques.

API du Poll Mod Driver

Le PMD apporte un certain nombre d'API et de fonctions, n'utilisant pas de verrous logiciels mais qui cependant ne peuvent pas être appelées à s'exécuter sur le même objet (file d'attente, port, etc.) par deux cœurs CPU différents. Cet ensemble d'API est utilisé pour différents objectifs, allant de la gestion des ressources du processeur, de la mémoire et des interfaces réseau jusqu'à la configuration de la qualité de service, des couches réseau et des fonctions de hachage. Le codage et le fonctionnement de ces API sont détaillés dans [19].

2.2.3 Netmap

2.2.3.1 Présentation

A l'instar de DPDK, *Netmap* est un framework dédié à l'accélération du packet processing dans des équipements réseau. Il peut être déployé sur des distributions Linux/FreeBSD ou plus récemment sur Windows et ses librairies sont compatibles avec toutes les cartes réseau présentes actuellement sur les équipements réseau. *Netmap* fournit un ensemble de mécanismes permettant d'optimiser l'acheminement et le traitement du trafic de données entre les ports réseau et les applications ou processus s'exécutant dans l'espace utilisateur. Ces mécanismes ont pour objectif de :

— définir une représentation « allégée » et compacte des métadonnées d'une interface réseau pour notamment simplifier leur utilisation. Pour cela, les caractéristiques spécifiques à un certain type d'équipement sont masquées aux applications,
— exploiter des structures de données (anneaux, buffers, files d'attente) pré-allouées à l'exécution de *Netmap* qui vont permettre d'éviter l'allocation dynamique de la mémoire, basée sur le volume des paquets entrants. Ce mécanisme optimise ainsi l'utilisation des zones mémoire et apporte un gain de temps grâce à la pré-allocation,
— supprimer la réplication (ou la copie) des données, en accordant aux applications ainsi qu'aux ports réseau, un accès direct aux différentes structures de données et à la mémoire partagée.

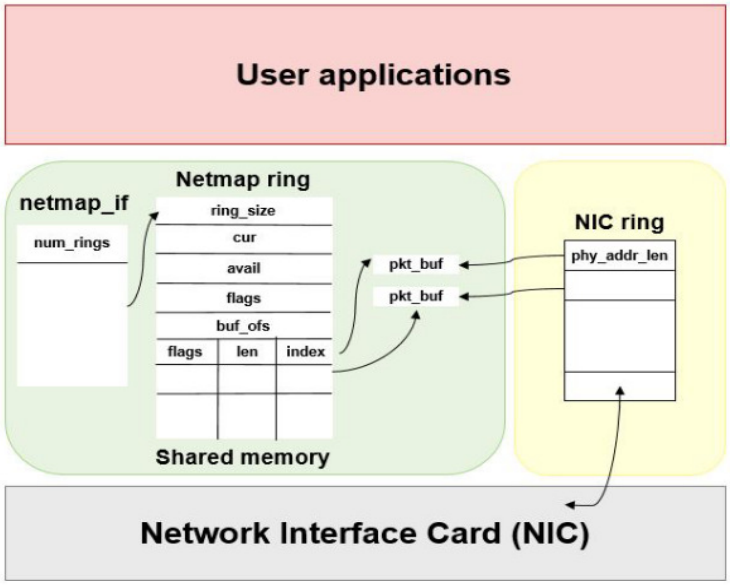

FIGURE 2.8 – Architecture logicielle de *Netmap*.

Cet accès direct sera néanmoins protégé de tout changement susceptible de corrompre le bon fonctionnement de l'application de packet processing, de l'équipement ou encore du système d'exploitation,

— exploiter des fonctions matérielles, telles que les files d'attente matérielles pour de meilleures performances.

Afin d'éviter les interruptions provenant du système d'exploitation, *Netmap* sépare partiellement, lors de son exécution, les interfaces réseau auxquelles il est lié, des couches protocolaires du noyau. Les paquets transitant par les ports d'entrée/sortie sont alors acheminés vers les applications (et vice versa) à travers des anneaux logiciels alloués au niveau de la mémoire partagée.

2.2.3.2 Eléments logiciels

Afin de garantir un bon acheminement du trafic de données entre les ports réseau et les applications, *Netmap* fournit trois types d'objets qui sont associés à chaque interface réseau utilisée (figure 2.8). Ces objets sont alloués dans la même zone mémoire et sont partagés entre les différents processus exécutés. Néanmoins, chaque zone mémoire est dédiée à une seule interface pour une meilleure isolation. Une description de chacun des objets utilisés par *Netmap* est donnée ci-dessous :

- **Buffer de paquets (*pkt_buf*) :** est une structure de données de taille fixe, pré-allouée à l'exécution de *Netmap*, partagée entre les ports réseau d'entrée/sortie et les applications utilisateur. Chaque buffer est identifié par un index unique, pouvant être translaté en une adresse virtuelle par les applications, ou bien en une adresse physique par l'interface réseau. De plus, un buffer

est également décrit par une longueur de données (taille) et par des *flags* de signalisation. Cet ensemble de métadonnées est stocké au niveau d'un « *slot* », se trouvant dans l'anneau de *Netmap*. Enfin, le buffer sert d'intermédiaire entre les anneaux de données de l'interface réseau et celui de *Netmap* pour le stockage des paquets entrants/sortants.

- **Anneau :** est une structure de données allouée au niveau de la mémoire partagée, permettant de transporter les paquets stockés dans les buffers vers les applications utilisateur. Un anneau est défini par les attributs suivants : ***Ring_size*** qui représente le nombre de *slots* appartenant à l'anneau. ***cur*** qui désigne la position courante de lecture/écriture dans l'anneau. ***avail*** qui détermine le nombre de buffers disponibles pour un traitement, c.-à-d. les buffers contenant des paquets entrants ou bien les buffers vides pour la transmission des paquets sortants. ***buf_ofs*** qui indique le décalage entre l'anneau et le début des buffers de paquets dans la mémoire partagée. Enfin, l'attribut ***slots*** qui représente un tableau de taille *Ring_size*, contenant les attributs de chaque buffer de paquet.
- **Interface *Netmap* (*netmap_if*) :** contient les informations de l'interface réseau, accessibles en lecture seule. Cet objet est décrit par deux attributs : ***num_rings*** qui désigne le nombre d'anneaux utilisés pas l'application et ***ring_ofs[]*** qui indique le décalage en mémoire entre chaque anneau *Netmap*.

2.2.3.3 Implémentation

Ainsi que nous l'avons mentionné précédemment, *Netmap* peut être implémenté dans une distribution FreeBSD ou Linux et s'exécute dans l'espace utilisateur (userspace). L'avantage d'une telle exécution est la simplification de la gestion et du contrôle des ressources. Cela permet également d'éviter les interruptions provoquées, par le système d'exploitation, qui peuvent éventuellement affecter les performances des applications. En outre, un certain nombre d'outils sont intégrés afin d'offrir un environnement de développement flexible et performant aux utilisateurs.

2.2.3.3.1 Interface de programmation d'applications (API)

Netmap fournit un ensemble d'API permettant aux utilisateurs d'allouer, d'accéder et de gérer la mémoire et les structures de données partagées entre les applications de packet processing. Ces API sont décrites ci-dessous :

- ***Ioctl()* :** représente un fichier de description et peut utiliser trois différents arguments (NIOREG, NIOCTXSYNC, NIOCRXSYNC). L'argument NIOREG contient le nom de(s) l'interface(s) utilisée(s) par l'application et retourne (à travers *ioctl*) la taille de la mémoire partagée, contenant les structures de données ainsi que le décalage de l'objet *netmap_if* (*ring_ofs*). L'argument NIOCTXSYNC gère la transmission des paquets de données ainsi que la mise à jour des valeurs de l'objet *avail* et des anneaux utilisés par *Netmap*. L'argument NIOCRXSYNC gère la réception des paquets et la mise à jour de l'objet *slot*, en précisant les longueurs et les données des paquets pouvant être lus. L'utilisation de *ioctl* est « non bloquante » et ne nécessite

donc pas de réplication des données. Cette fonction peut également traiter plusieurs paquets en simultané.

- ***Blocking I/O* :** ou primitive de blocage, est utilisée par les appels système *select()* et *poll()*. Elle permet également de mettre à jour les anneaux *Netmap*.
- **Interface multi-queue :** utilisée sur des cartes possédant plusieurs anneaux, cette interface permet de configurer les fichiers de description selon deux modes. Le mode par défaut qui permet au descripteur de files (*ioctl*) de gérer l'ensemble des anneaux *Netmap*. Ainsi que le mode alternatif qui crée et affecte un fichier de description à chaque anneau pour une exécution indépendante de ces derniers, sans la nécessité d'une synchronisation entre les anneaux.

2.2.3.3.2 Communication avec la couche réseau du noyau

A l'inverse de DPDK, Netmap permet à la couche réseau du noyau Linux de garder l'accès aux interfaces et de générer du trafic de données à travers ses fonctions (*ifconfig, ping, etc.*). Pour cela, les API décrites précédemment sont utilisées par *Netmap* pour la transmission ou la réception des paquets du noyau à travers des anneaux et des fonctions *ioctl* NIOCRXSYNC (pour la transmission), NIOCTXSYNC (pour la réception).

2.2.3.3.3 Résilience des données

Bien qu'une mauvaise exécution des applications n'est pas à exclure, *Netmap* permet néanmoins d'éviter les crashes du noyau Linux, en séparant les zones de mémoire utilisées par ce dernier de celles utilisées par *Netmap*. En outre, afin d'éviter une éventuelle corruption des données contenues dans les anneaux ou les buffers *Netmap*, il est possible d'allouer une zone mémoire pour chaque anneau utilisé.

2.2.3.3.4 Non-réplication des données

Netmap permet d'éviter la réplication des paquets de données lors d'une transmission entre les interfaces réseau. Pour cela, les indexes des buffers partagés sont échangés entre les *slots* de réception, se trouvant au niveau de l'interface d'entrée, et de transmission, se trouvant au niveau de l'interface de sortie.

2.2.4 Autres solutions de packet processing

2.2.4.1 PF_RING

Egalement nommé PF_RING ZC (pour Zero Copy), PF_RING est un framework de packet processing propriétaire, développé par *Ntop* et implémenté sur des distributions Linux [20]. PF_RING permet de capturer, d'analyser et de traiter les paquets de données, transitant par les machines physiques ou

virtuelles l'exécutant. Dans sa version de base, PF_RING possède une architecture modulaire qui lui offre une flexibilité accrue, par rapport aux SoC ou aux cartes réseau utilisés (Endace, Exablaze, etc.). En outre, PF_RING intègre la fonctionnalité « *Zero Copy* » qui permet d'éviter la réplication des données dans la mémoire (à l'instar de *Netmap*).

PF_RING ZC est considéré comme le remplaçant de Direct NIC Access (DNA)/Libzero, dont il reprend certaines fonctionnalités, telles que l'accès direct aux registres et à la mémoire des cartes réseau. PF_RING ZC apporte néanmoins de nouvelles fonctionnalités, telles que, l'utilisation des *hugepages*, ou encore, l'ajout d'un nouveau mode appelé « *one-copy* » qui permet de copier les données d'un paquet pour être traitées au niveau du noyau Linux à travers un module. Souvent comparé à *Netmap* pour ses performances équivalentes, PF_RING ZC reste cependant payant, contrairement à *Netmap* ou à *Intel* DPDK.

2.2.4.2 PacketShader

PacketShader est un framework logiciel [21], destiné aux routeurs exploitant l'accélération d'un processeur graphique (Graphic Processing Unit (GPU)) pour un traitement des paquets de données plus performant, avec une réduction de la charge de calcul pour les cœurs CPU du routeur. Pour cela, *PacketShader* utilise une carte graphique Nvidia GTX480 qui intègre 480 cœurs, répartis sur 32 processeurs, pouvant ainsi exécuter jusqu'à 480 threads en simultané pour plus de performance. Par ailleurs, *PacketShader* utilise un nouveau schéma de buffers de paquets, appelé « *huge packet buffer* », qui permet d'allouer deux larges buffers de taille fixe pour le stockage des métadonnées, dans le premier buffer et les données d'un paquet dans le second buffer. *PacketShader* intègre également le mécanisme de traitement par lot (*batch processing*), pour réduire la surcharge des processus, due au traitement des paquets.

Les expérimentations réalisées sur un driver *ixgbe* ont montré que *PacketShader* permet d'atteindre des débits de 39 Gbit/s avec des paquets IPv4 de 64 octets et 38 Gbit/s avec des paquets IPv6.

2.2.4.3 OpenOnload

OpenOnload est une pile réseau open source, développée par *Solarflare* sur des distributions Linux et FreeBSD, pour un traitement des paquets plus performant, une faible latence et un débit plus élevé. A l'instar de DPDK ou ODP, *OpenOnload* s'exécute dans l'espace utilisateur, ce qui lui permet d'outrepasser les couches du noyau du système et ainsi réduire l'overhead, lors du traitement du trafic entrant. L'utilisation de *OpenOnload* reste néanmoins limitée à certaines cartes réseau spécifiées dans [22]. Une version payante appelée *Enterprise Onload* est proposée par *Solarflare* et qui offre des performances supérieures à la version libre. Les principaux apports de *Enterprise Onload* sont donnés ci-dessous :

— accélération du traitement des paquets UDP/TCP à travers l'utilisation de la librairie *TCPDirect*,
— faible latence des communications inter-applications,

— 3 millions de paquets traités par seconde avec un seul cœur CPU,
— pas de modification de noyau ou d'application requise.

2.2.4.4 PFQ

PFQ est un ensemble open source de librairies *C++* et de modules Linux, pouvant s'exécuter dans l'espace utilisateur et qui fournit une capture ainsi qu'une transmission parallélisée des paquets de données au sein d'un équipement réseau [23]. Pour ce faire, PFQ exploite les cœurs CPU et les files d'attente des interfaces réseau, à travers les éléments suivants :

- **Packet fetcher :** permet de récupérer les paquets entrants à partir des drivers d'interfaces réseau, pour ensuite les enfiler dans des files d'attente appelées « *batching queues* », pouvant stocker jusqu'à cent paquets (capacité maximale).
- **Bloc de pilotage (ou de démultiplexage) :** consiste en une matrice de routage qui distribue les paquets capturés et stockés aux différents sockets, utilisés par PFQ.
- **File d'attente du socket :** représente l'interface entre le noyau et l'espace utilisateur de Linux. Cet élément contient deux buffers (files d'attente) qui stockent les paquets transmis par le bloc de pilotage (dans le premier buffer) et ceux transmis par les applications (dans le second buffer). Le stockage des données et des métadonnées de ces paquets est alors géré par des threads qui sont alloués à chacune des portions de ces deux buffers.

PFQ traite jusqu'à 14.8 millions de paquets par seconde et peut supporter des applications développées en *C, C++* ou bien *Haskell*.

2.2.4.5 SnabbSwitch

SnabbSwitch est un switch virtuel open source [24], conçu en langage *Lua* pour s'exécuter sur l'espace utilisateur de l'hyperviseur Kernel-based Virtual Machine (KVM) et qui a pour objectif d'améliorer les performances réseau dans un environnement SDN/NFV à l'instar de OVS-DPDK. *SnabbSwitch* introduit un nouveau compilateur, appelé *Lua Just In Time (LuaJIT)* qui permet d'optimiser l'exécution de l'environnement virtuel, en traitant les opérations représentatives du comportement dynamique des applications. Pour cela, *LuaJIT* s'appuie sur deux modules appelés *bit* et *ffi* qui fournissent des fonctions pour une compilation bit à bit des opérations (grâce au module *bit*) ou encore pour des appels de fonctions *C*, à partir d'un code *Lua* (grâce au module *ffi*). Le principal avantage de ce compilateur est son traitement dynamique, tout au long de l'exécution des applications (à l'inverse de OVS qui possède une compilation préalable et statique à son exécution). Ce qui améliore le traitement du trafic de données passant par le switch. *SnabbSwitch* est constitué de trois principaux éléments présentés ci-dessous :

- ***App* :** représente un module logiciel qui gère et contrôle les liaisons d'entrée/sortie de *SnabbSwitch* à travers l'utilisation de deux méthodes d'exécution, la méthode *pull* pour le traitement des paquets entrants et la méthode *push* pour l'envoi de ces paquets vers les ports de sortie.

	ODP	DPDK	Netmap	PF_Ring ZC	PacketShader	OpenOnload / EnterpriseOnload	PFQ	SnabbSwitch
Langage de programmation	C (C99)	C/Python	C/C++ / Python	C++11	C/Python	C/Python	C/ C++ / Haskell	Lua
Plateforme	Linux/FreeBSD	Linux/FreeBSD	Linux/FreeBSD / Windows	Linux	Linux	Linux/FreeBSD	Linux	QEMU/KVM
Environnement d'exécution	Userspace	Userspace	Userspace	Kernel	Userspace	Userspace	Userspace	Userspace
Réplication des paquets	Selon l'application	Non (zero copy)	Non (zero copy)	Non (zero copy)	Oui	Oui	Selon le mode d'exécution	Oui
Utilisation des Hugepages	Selon l'application	Oui	Non	Oui	Non	Non	Oui	Non
Interfaces réseau supportées	Compatible avec toutes les interfaces	Limité à certaines interfaces	Compatible avec toutes les interfaces	Compatible avec toutes les interfaces	Intel 82598/82599	Limité à certaines interfaces	Driver Intel IXGBE	Compatible avec toutes les interfaces
Accès au code	Open source	Open source	Open source	Payant (version Zero copy)	Accès à certaines fonctionnalités seulement	Open source (OpenOnload)	PFQ	Open source

TABLE 2.1 – Tableau comparatif entre les principales solutions de packet processing

Deux *App* sont nécessaires à l'exécution de *SnabbSwitch*. Le premier module représente le driver de l'interface réseau qui permet d'interagir avec cette dernière, en accédant à son espace d'adressage directement à partir de l'espace utilisateur, grâce à la libraire *mmap*, sans passer par les couches du noyau. Le second module représente l'implémentation de l'interface réseau virtuelle qui permet de faire communiquer les machines virtuelles « guest » avec le *SnabbSwitch*. En outre, un certain nombre de modules sont disponibles pour diverses utilisations, telles que la limitation du trafic de données pour une VM, le filtrage de paquets, etc.

- ***Link* :** représente un anneau qui sert à stocker les paquets échangés entre les *app*.
- ***App engine* :** permet d'initialiser et de configurer les deux précédents éléments. *App engine* peut être reconfiguré dynamiquement et gère également l'exécution des méthodes *pull* et *push*, pour le traitement des paquets de données. Ainsi, il représente l'élément principal de *SnabbSwitch*.

2.2.5 Comparaison

Bien que les solutions de packet processing citées précédemment présentent certaines similarités entre elles, telles que l'utilisation de structures de données (files d'attentes, threads, anneaux, etc.) ou encore l'exécution des fonctions dans l'espace utilisateur Linux, elles possèdent toutefois des différences notables dans la conception de l'architecture de chaque framework, la gestion des ressources matérielles, le traitement des paquets de données, etc. Par conséquent, nous avons résumé les particularités de chacune des solutions dans la table 2.1. En outre, des comparaisons des performances, entre plusieurs accélérateurs, ont été menées dans [25] et [26].

2.3 Plateforme Metamorphic Network (MNet)

2.3.1 Présentation

Metamorphic Network (MNet) constitue une plateforme virtuelle qui permet à des utilisateurs d'instancier, de manière simple et sans configuration, des VM au sein d'un environnement Xen. Ces

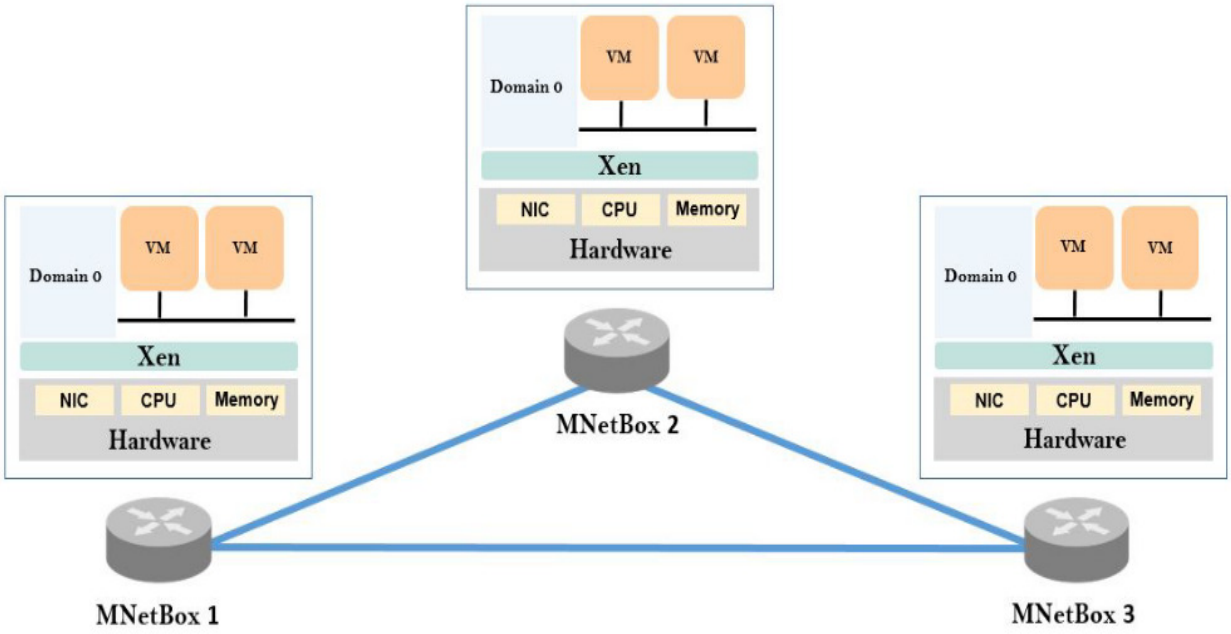

FIGURE 2.9 – Architecture de la plateforme Metmorphic Network (MNet).

VM peuvent représenter des applications, des outils logiciels ou bien des fonctions réseau (routeur, point d'accès Wi-Fi, contrôleur SDN, etc.), s'exécutant de manière isolée entre elles et par rapport au système hôte sous-jacent. En outre, des ressources CPU et mémoire peuvent être allouées dynamiquement à chaque VM, à partir du système hôte, afin d'optimiser les performances de la plateforme.

La plateforme MNet (figure 2.9) est implémentée au sein d'un ensemble de nœuds physiques, appelés « MNetBox », interconnectés à travers un réseau filaire local, ou bien à travers un tunnel overlay (Virtual Extensible LAN (VXLAN) [27], Generic Network Virtualization Encapsulation (GENEVE) [28], etc.), lorsque deux nœuds sont localisés dans différents réseaux.

2.3.2 Hyperviseur Xen

Xen est un moniteur de machines virtuelles (Virtual Machine Monitor (VMM)), désigné également comme un hyperviseur de type-1, pouvant être implémenté directement sur l'équipement physique (*bare-metal*) ou installé sur le système d'exploitation (Operating System (OS)) hôte (à partir de la version 4.0). Xen s'appuie sur un environnement de paravirtualisation (Paravirtualization (PV)) qui constitue une interface logicielle similaire (mais non identique) aux interfaces matérielles et logicielles (système hôte) sous-jacentes. La PV fournit des performances élevées et quasi équivalentes à la machine physique hôte. En contrepartie, les noyaux des VM invitées doivent être portés et recompilés (compilation des modules Xen, installation de drivers, etc.). Par conséquent, seules des distributions open source (p. ex., Linux) pourront être paravirtualisées dans Xen.

Xen supporte également la virtualisation matérielle (Hardware-assisted Virtualized Machine (HVM) ou *full virtualization*) qui permet d'héberger des OS propriétaires (p. ex., Windows). Cependant, les performances obtenues par la HVM, lors d'opérations e/s, demeurent significativement inférieures à celles obtenues par la paravirtualisation, en raison du surcoût causé par l'émulation complète des couches matérielles dans la VM. Par ailleurs, une virtualisation matérielle nécessite la présence des technologies *Intel VT* ou *AMD-v* dans le processeur de la machine hôte. Xen intègre différents éléments (figure 2.10), décrits comme suit :

- ***Domain 0 (Dom0)* :** constitue l'interface virtuelle intermédiaire entre les machines invitées et les couches matérielles sous-jacentes. *Dom0* est créé automatiquement par Xen, au démarrage de l'équipement physique et possède un accès privilégié aux ressources CPU/mémoire de ce dernier, sans toutefois impacter l'exécution des applications présentes dans le système hôte. En outre, *Dom0* permet de créer, supprimer et contrôler dynamiquement la totalité des VM invitées, à travers une interface de contrôle.
- ***Control interface* :** représente la partie gestion de l'hyperviseur Xen et qui peut être manipulée par l'utilisateur à travers le *Dom0*. L'interface de contrôle intègre des fonctions de gestion des ressources mémoire (Memory Management Unit (MMU)), CPU (grâce à un ordonnanceur) et réseau des VM invitées. Elle permet par conséquent de déterminer, à travers un fichier de configuration, la taille de la mémoire, le nombre de cœurs CPU et d'interfaces réseau virtuels d'une VM, lors de sa création. De plus, l'interface de contrôle peut accorder, à une machine virtuelle, un accès direct aux ports physiques e/s sous-jacents (interface réseau, ports USB, etc.) grâce au mode *Passthrough*. Un mécanisme de timers est également intégré dans Xen afin de maintenir la précision d'exécution de la machine physique (*real time*) et des VM invitées (*virtual time*). De plus, une troisième notion de timers est définie dans [9], nommée *wall-clock time*, qui se charge de l'ajustement entre les horloges des OS hôtes et invités. Enfin, l'interface de contrôle gère les deux modes, *Live* et *Offline*, de migration des VM entre les machines physiques.
- ***Domain User (DomU)* :** appelé également *Domain Guest*, *DomU* intègre le système d'exploitation de la machine virtuelle invitée. Ses ressources virtuelles (CPU, réseau, mémoire, etc.) sont définies, lors de sa création, dans un fichier de configuration qui est exécuté à partir du *Dom0*. Toutefois, certains paramètres, tels que la taille de la mémoire, peuvent être modifiés dynamiquement lorsqu'une VM invitée est en cours de fonctionnement. En outre, un utilisateur peut créer/supprimer plusieurs *DomU*, paravirtualisés ou bien matériellement virtualisés, au sein d'un même équipement physique. Les VM contenus dans ces *DomU* vont alors s'exécuter de manière isolée.

Xen présente de nombreux avantages, en comparaison à des hyperviseurs concurrents de type 1, tels que, KVM, Microsoft HyperV ou encore VMware ESXI :

— Xen est open source et gratuit, à l'instar de KVM. Bien qu'il existe des versions propriétaires développées par CITRIX (*XenServer*) et ORACLE (*ORACLE VM*),
— en supportant les techniques HVM et PV, Xen peut être implémenté sur toutes les architectures de processeur existantes actuellement (X86, ARM, etc.), embarquant les technologies *Intel-VT/AMD-V* ou pas. En outre, Xen peut embarquer différents OS invités, telles que les nombreuses distributions Linux ou encore le système Windows,

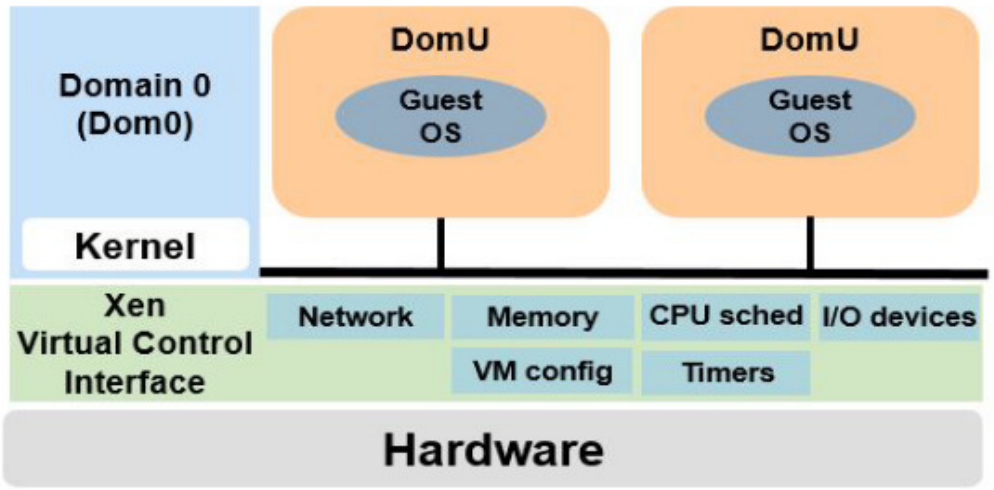

FIGURE 2.10 – Architecture de l'hyperviseur Xen.

— Xen constitue le seul hyperviseur open source, supportant la paravirtualisation, bien que cette dernière puisse être utilisée par KVM sur les drivers e/s de la machine physique.

2.3.3 Interconnexion des MNetBox

L'interconnexion des MNetBox s'opère à travers des liaisons filaires de type Ethernet. En outre, deux solutions réseau peuvent être déployées au sein de la plateforme MNet. La première solution, consiste en la mise en place du protocole de couche 2 TRILL dans le réseau local interconnectant les MNetBox. TRILL intègre des solutions de couche 3, telles que le calcul des plus courts chemins entre les différents nœuds (utilisation du protocole de routage Intermediate System to Intermediate System (IS-IS) [29]), la construction d'arborescences multicast et l'utilisation d'un compteur de sauts, empêchant l'apparition d'éventuelles boucles dans le réseau. Toutefois, TRILL garde les avantages d'un protocole de couche 2, tels que l'auto configuration, la vitesse de transmission et la réduction du temps de convergence de la topologie du réseau. La deuxième solution consiste en la mise en place d'un réseau SDN, reliant les MNetBox à travers un ensemble de switches. Dans ce type d'architecture, un ou plusieurs contrôleurs (selon le nombre de MNetBox déployées) sont virtualisés au sein de la plateforme MNet. Ces contrôleurs fournissent une vue globale du réseau et permettent de gérer l'acheminement des flux données transitant par les switches SDN. Les deux solutions réseau introduites dans cette partie seront détaillées dans le chapitre 4.

Toutefois, un tunnel overlay peut être établi entre deux MNetBox, lorsque ces dernières sont déployées dans deux réseaux distincts. Ainsi, la plateforme MNet supporte deux solutions de tunneling :

- **Virtual eXtensible Local Area Network (VXLAN) :** est un protocole de tunneling standardisé par Internet Engineering Task Force (IETF). VXLAN possède un schéma d'entête de couche

2 (Ethernet) sur un entête de couche 3 (IP). Il permet ainsi de transporter des trames Ethernet au-dessus d'un réseau IP. Pour cela, la trame est encapsulée dans un datagramme UDP, utilisant par défaut le port 4789. De plus, chaque segment VXLAN est identifié par un champ d'entête de 24 bits, nommé VXLAN Network Identifier (VNI). Par conséquent, il est possible de créer plus de 16 millions de segments au sein d'un même domaine administratif. Un nombre significativement supérieur aux 4096 réseaux VLAN, pouvant être déployés au maximum. Les communications entre les VM connectées au même segment (mais se trouvant dans des réseaux différents) se font à travers les extrémités du tunnel VXLAN, nommées VXLAN Tunnel End Point (VTEP). Les VTEP sont généralement établis au niveau de l'hyperviseur et sont associés aux adresses MAC/IP (appelées également adresses externes) de la machine hôte. A la réception d'une trame Ethernet, envoyée par une VM invitée, le VTEP « local » analyse l'adresse MAC de destination, à partir de l'entête de la trame (appelé entête interne), pour déterminer le VTEP distant associé à cette destination. Les entêtes VXLAN (contenant le VNI du segment), UDP et les entêtes externes (contenant l'adresse MAC/IP du VTEP distant) sont alors ajoutées à la trame avant qu'elle ne soit envoyée à la destination. Une fois arrivé, le VTEP distant analyse le VNI du segment associé à la trame avant de retirer les entêtes externes et de transmettre vers la VM destination. Dans le cas de non-connaissance de la destination, un mécanisme de diffusion multicast est mis en place dans VXLAN, lequel permet de diffuser l'adresse MAC non connue à tous les VTEP du segment.

- **Generic Network Virtualization Encapsulation (GENEVE) :** est un tunnel overlay, récemment standardisé par IETF. A l'instar de VXLAN, GENEVE encapsule des trames Ethernet dans de l'UDP, utilisant le port 6081, assigné par Internet Assigned Numbers Authority (IANA), pour être transportées par la suite au dessus d'un réseau IP. En outre, chaque segment est identifié par un VNI unique de 24 bits. Néanmoins, GENEVE se distingue de VXLAN par son indépendance du plan de contrôle (Control Plane (CP)). Pour ce faire, des options peuvent être ajoutées par un constructeur/opérateur directement dans l'entête du protocole de tunneling, afin de l'adapter aux différents CP utilisés.

2.4 Intégration de ODP dans la plateforme MNet

L'accélération du traitement des paquets de données nécessite une importante utilisation des ressources physiques d'un équipement réseau. En effet, les outils logiciels de type ODP, *Intel* DPDK ou *Netmap* tirent parti de la capacité maximale des cœurs CPU et de la mémoire qui leurs sont alloués. Cela peut négativement affecter le fonctionnement de l'équipement et des VM hébergées dans ce dernier, notamment lorsque le nombre de cœurs CPU et la taille de la mémoire sont limités. Il est ainsi nécessaire d'adapter l'architecture de la plateforme MNet pour pouvoir atténuer cette surutilisation des ressources, sans impacter les performances pouvant être apportées par le framework de packet processing utilisé.

Nous allons décrire, dans cette section, l'architecture Xen que nous avons implémentée pour pouvoir intégrer le framework ODP. Notre objectif, à travers cette contribution, est d'optimiser le traitement du trafic de données, transitant par MNet, tout en minimisant l'utilisation de ses ressources matérielles

qui pourraient, selon le contexte (nombre élevé de VM actives, équipement physique à faible capacité, etc.), être limitées. Par ailleurs, nous avons choisi ODP, comme solution de packet processing, pour les raisons suivantes :

— logiciel open source, ODP est soutenu par *Linaro* ainsi que par treize équipementiers réseau, dont Cisco, Nokia, Cavium, etc.
— supporte différentes architectures de processeurs, telles que *Intel x86, ARM, MIPS*, etc.
— supporte également la majorité des cartes réseau disponibles actuellement sur le marché, grâce à la couche d'abstraction apporté par ses API,
— intègre une fonction de classification des paquets de données, selon le protocole de transport utilisé (TCP, UDP, etc.) ou selon le numéro du VLAN.

Il existe toutefois d'autres travaux menés dans le domaine de l'optimisation des ressources ainsi que de l'accélération du packet processing dans un environnement virtuel. Nous allons ainsi résumer, dans cette section, certaines des propositions existantes dans la littérature.

2.4.1 Travaux connexes

Considéré comme un concept novateur lors de sa présentation par European Telecommunications Standards Institute (ETSI) en 2012, la technologie NFV est actuellement largement déployée dans les infrastructures Cloud des opérateurs et des fournisseurs de services. En outre, une multitude de plateformes NFV ont émergé, apportant de nouvelles fonctions pour les DP, de contrôle et de gestion, redéfinissant ainsi les architectures réseau existantes. Nous citons Open Platform for Networks Functions Virtualization (OPNFV) [30], une plateforme open source qui fournit un ensemble de fonctions, d'outils et de couches logicielles, destinés au développement et au déploiement de composants NFV. OPNFV repose sur trois principes qui sont : **1) l'intégration** d'une variété d'outils open source pour adresser les besoins de chaque plan réseau (données, contrôle et gestion) ; **2) le test** de ces différents outils selon des paramètres spécifiques à chaque fonction réseau ; **3) l'apport de nouvelles fonctionnalités** destinées à améliorer le fonctionnement global de la plateforme ainsi que faciliter le déploiement de cette dernière dans les réseaux d'opérateurs. OPNFV associe par ailleurs *Intel* DPDK à ODP pour l'optimisation et l'accélération de la couche du DP. Ces deux technologies sont gérées à travers la version *Ocata* de *OpenStack*.

La problématique d'optimisation des ressources dans un environnement Xen a été initialement traitée dans [31] et dans [32], où une nouvelle architecture Xen a été présentée, intégrant un deuxième domaine virtuel privilégié (en plus du Dom0), nommé *Driver Domain (DD)*. Ce dernier possède un accès direct aux ressources réseau de la machine physique et permet de traiter le trafic de données transitant entre le réseau sous-jacent et les VM invitées. Cette architecture Xen offre des gains de performance pour les VM invitées de 340% (débit de transmission) et de 18% (débit de réception). Nous nous sommes par ailleurs basés sur le travail réalisé dans [32] pour définir notre architecture. Une autre contribution a été apportée dans [33], avec *ClickOS*, une plateforme virtuelle basée sur Xen, optimisée pour une utilisation dans les middleboxes. De plus, *ClickOS* fournit des performances optimales avec un délai avoisinant les 45 μs et un débit de 10 Gbit/s.

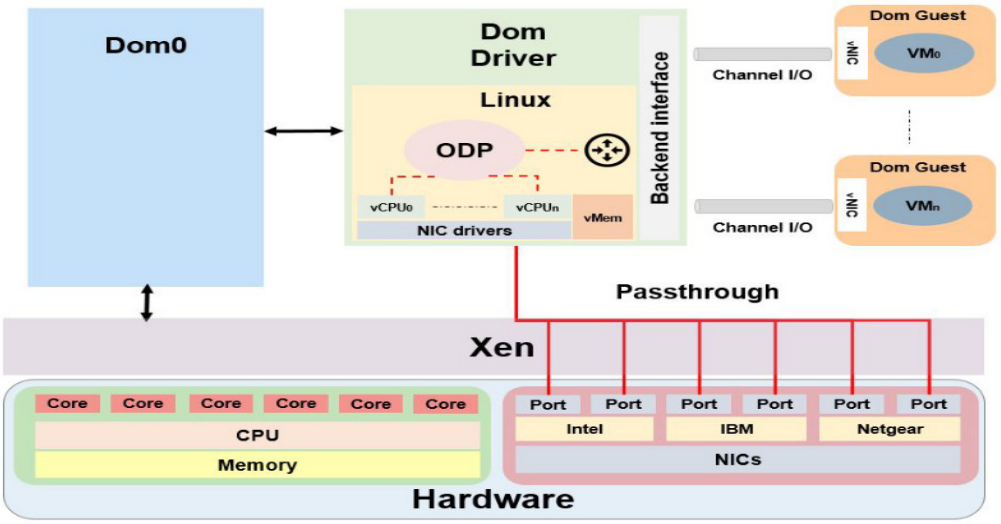

FIGURE 2.11 – Nouvelle architecture Xen pour la plateforme MNet.

Une multitude de solutions, utilisant *Intel* DPDK, ont été proposées, à l'exemple de [34], une architecture qui peut simultanément gérer le traitement du trafic de plusieurs fonctions réseau virtualisées (Virtual Network Function (VNF)) grâce à DPDK. Bous citons également NetVM [35], une nouvelle plateforme de virtualisation, implémentée au sein d'un environnement KVM et utilisant les librairies DPDK. NetVM permet d'accélérer les communications inter-VM et d'augmenter le débit de données de 250%, en comparaison à une architecture Single-Root Input/Output Virtualization (SR-IOV) [36]. Cependant, *Kourtis et al* proposent d'unir les deux frameworks, DPDK et SR-IOV, pour améliorer les performances des VNF [37]. Enfin, une nouvelle architecture hybride (logicielle/matérielle) a été proposée, appelée SoftNIC [38]. Elle fournit une plateforme programmable, pouvant étendre les fonctionnalités des interface réseau pour une meilleure flexibilité et plus de performance, en comparaison avec des NIC traditionnelles. SoftNIC peut atteindre 40 Gbit/s en allouant 1 à plusieurs cœurs CPU à DPDK.

Des travaux ont été également menés sur les librairies *Netmap* et sur son précurseur VALE [39], telles que *PTnetmap* [40], un équipement réseau virtuel permettant de relier des VM directement à l'environnement *Netmap* sous-jacent, sans passer par les couches de l'hyperviseur. Ainsi, les performances obtenues par *PTnetmap* sont quasi-équivalentes à celles fournies nativement par *Netmap*.

2.4.2 Architecture Xen-ODP

Afin de réadapter l'architecture Xen de la plateforme MNet, nous nous sommes basés sur la solution proposée dans [32], en intégrant un DD au niveau de chaque nœud physique de MNet (MNetBox). A l'instar des autres VM invitées dans Xen, la configuration des paramètres de démarrage ainsi que la création du DD sont effectuées à partir du *Dom0*. Ces paramètres sont le nombre de cœurs CPU virtuels embarqués, taille de la mémoire vive, système utilisé, numéros des bus/périphériques des interfaces ré-

seau allouées, etc. Néanmoins, DD possède un accès direct et exclusif aux NIC qui lui sont allouées grâce au mode *Passthrough*, fourni par Xen (figure 2.11). De plus, DD peut être connecté, à travers une interface *Backend* et des canaux Input/Output (I/O), à un ou plusieurs domaines secondaires *Dom Guest*, qui hébergent les différentes VM invitées de la MNetBox. Ainsi, tous les paquets de données envoyés/reçus par les *Dom Guest* transitent obligatoirement par DD. La configuration et la création des *Dom Guest* se font également au niveau du *Dom0*. Enfin Les VM invitées communiquent entre elles (ou bien avec l'extérieur de la MNetBox) grâce à un bridge logiciel qui est établi dans DD.

Le framework ODP est déployé au sein de l'espace utilisateur Linux de DD. En outre, un certain nombre de cœurs CPU virtuels ($vCPU_i$) lui sont alloués pour l'accélération du traitement de trafic transitant par DD. ODP permet également de gérer les différentes NIC exploitées ainsi que le bridge, interconnectant les VM invitées. L'architecture proposée dans la figure 2.11 présente deux avantages : **1)** la possibilité d'allouer un nombre de vCPU supérieur au nombre de cœurs CPU physiques présents sur l'équipement sous-jacent (*CPU overcommitment*), bien que cela peut dégrader les performances réseau des VM sur certaines architectures de processeur ; **2)** l'exécution isolée du DD impacte peu les performances des ressources physiques sous-jacentes et des *Dom Guest*.

De ce fait, le principe de notre solution est de substituer les cœurs CPU physiques par des cœurs virtuels présents dans DD. ODP crée alors un nombre de threads correspondant au nombre de vCPU utilisés (un thread pour chaque vCPU). Chaque thread permet par la suite de traiter les paquets de données transitant par DD, selon le mode défini à l'exécution de ODP (voir Section 2.2.1).

2.5 Evaluation des performances

Dans cette section, nous avons comparé les performances fournies par notre nouvelle architecture à celles fournies par une architecture Xen traditionnelle. Pour ce faire, nous avons mesuré les indicateurs suivants : le débit de données et le nombre de paquets traité par seconde. Les implémentations évaluées dans cette partie sont présentées dans la figure 2.12. Chaque MNetBox embarque, dans le *Dom0*, un noyau Linux propriétaire, développé par VirtuOR et est équipée par un processeur $IntelCore^{TM}2Duo$ de 2.5 GHz, ainsi que par quatre cartes réseau de type *Intel 82571EB Gigabit Ethernet*. En outre, des VM de type Linux Debian sont déployées, au sein du *DomU* et du *Dom Guest* des architectures « Native » et « New », respectivement. Ces VM intègrent des générateurs de trafic de données qui vont permettre d'évaluer les performances des deux implémentations. Enfin, une autre VM Linux Debian, contenant la version 1.3 de ODP, est instanciée dans le domaine *Driver* et elle est directement reliée à l'interface réseau sous-jacente via le mode *Passthrough*. Nous avons utilisé en particulier cette version car elle fournit un outil d'accélération de paquets. Cet outil sera par la suite remplacé, à partir de la version 1.6, par DPDK et *Netmap*.

Les indicateurs de performance cités précédemment sont mesurés entre deux VM invitées, localisées dans deux MNetBox distinctes et connectées à travers une liaison Ethernet. Pour cela, des paquets UDP, d'une taille fixe de 1512 octets, sont générés par la VM source vers la VM destination, à travers l'outil *Iperf* dans l'architecture native, et à travers un générateur ODP dans la nouvelle architecture. Dans ce second cas, le trafic de données passera par le DD/ODP et transitera directement vers la NIC,

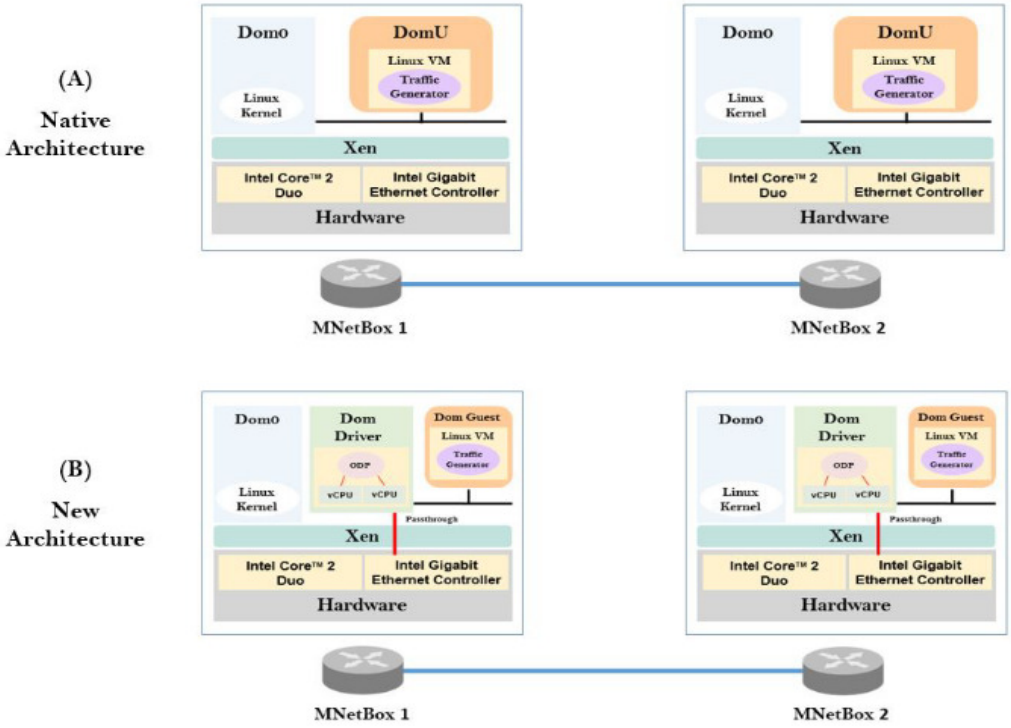

FIGURE 2.12 – Plateformes d'expérimentation des architectures Xen « Native » et « New ».

contrairement à l'architecture native, où les données sont d'abord traitées par Xen avant d'être acheminées vers la destination. Les résultats obtenus sont illustrés dans les figures 2.13, 2.14 et 2.15.

Les résultats du débit ainsi que du nombre de paquets traités (figure 2.13) sont obtenus à partir d'une moyenne, calculée sur un intervalle de 100 sec. Par ailleurs, nous varions le nombre de cœurs CPU virtuels, embarqués respectivement dans DD, pour l'architecture « New » et dans le *DomU*, pour l'architecture « Native ». Nous observons que l'utilisation de ODP (architecture « New ») apporte un gain moyen de 15% pour le débit de données (figure 2.13(A)), avec 958 Mbit/s (contre 816 Mbit/s pour l'architecture native) sur une Gigabit NIC, lorsque le nombre de vCPU est supérieur à 1. Même constat pour le nombre moyen de paquets UDP traité à la seconde (figure 2.13(B)), avec une augmentation de 17% pour la nouvelle architecture, en comparaison avec l'architecture Xen traditionnelle. Enfin, le pourcentage d'utilisation de la bande passante digitale (figure 2.14) augmente également grâce à ODP, atteignant les 95%, avec 2 et 3 vCPU. Notons que les performances de la nouvelle architecture stagnent avec l'utilisation de 3 ou 4 vCPU, dû à la saturation de la carte réseau. Ces performances tendent même à baisser sur la plateforme MNet, lorsque le nombre de vCPU est supérieur à 4 (nombre maximum de cœurs physiques dans la MNetBox), en raison du CPU *overcommitment*.

Nous avons mesuré également, pour chacune des architectures Xen, l'augmentation de l'utilisation des cœurs CPU physiques et virtuels au sein de la MNetBox. Nous avons noté que ODP a ajouté un overhead de 89% des ressources du processeur virtuel du DD. Cependant, cette surexploitation

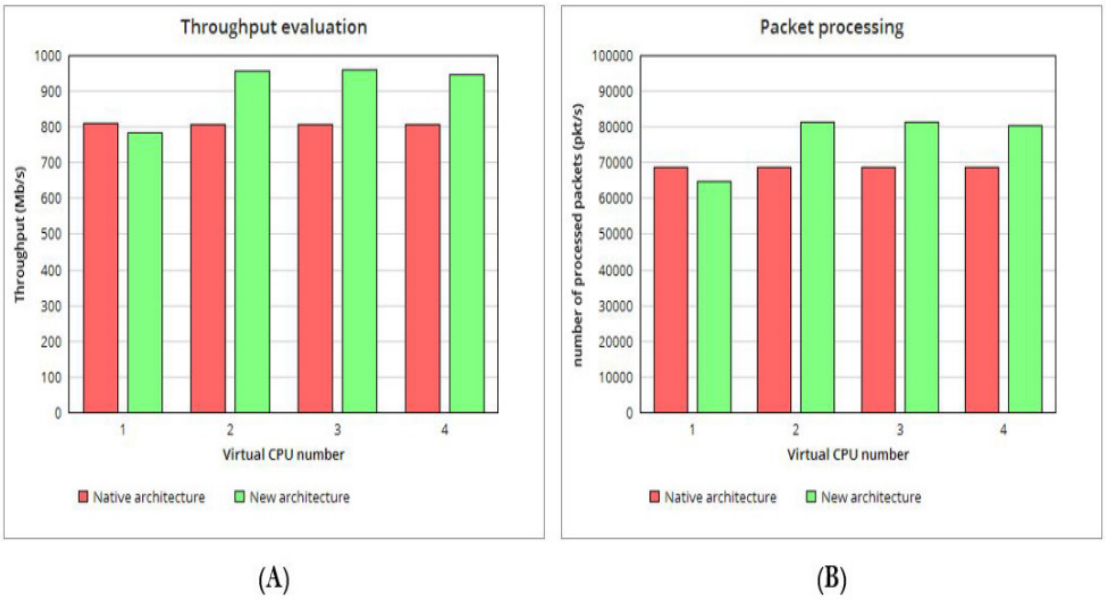

FIGURE 2.13 – Moyenne du débit de données et du nombre des paquets traités à la seconde en fonction du nombre de cœurs vCPU.

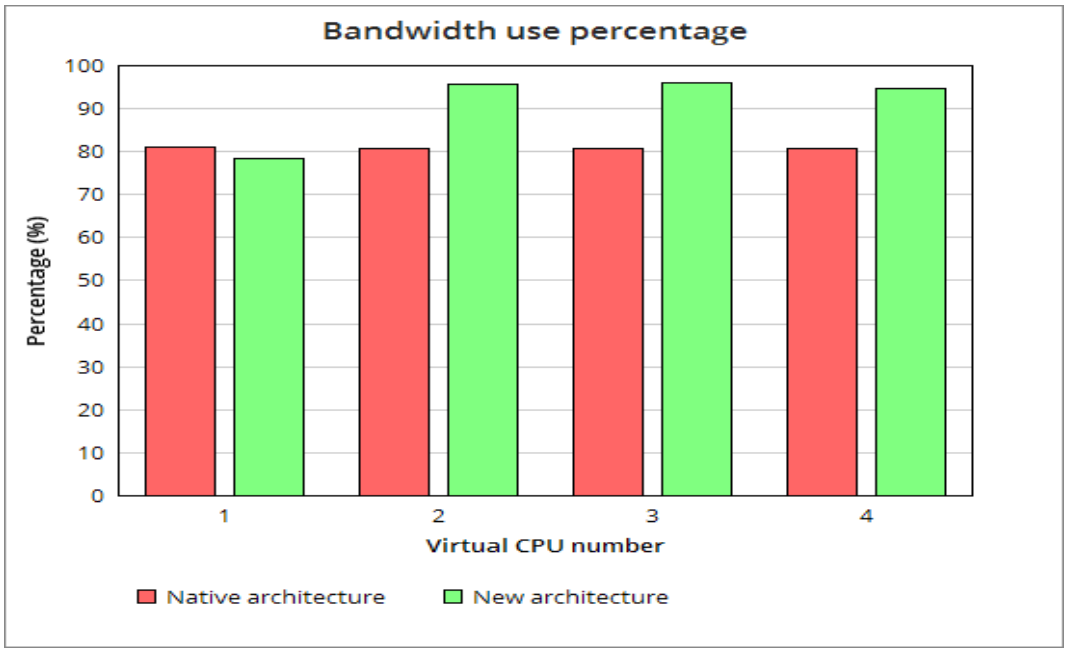

FIGURE 2.14 – Comparaison du pourcentage d'utilisation de la bande passante digitale entre les architectures *New* et *Native* .

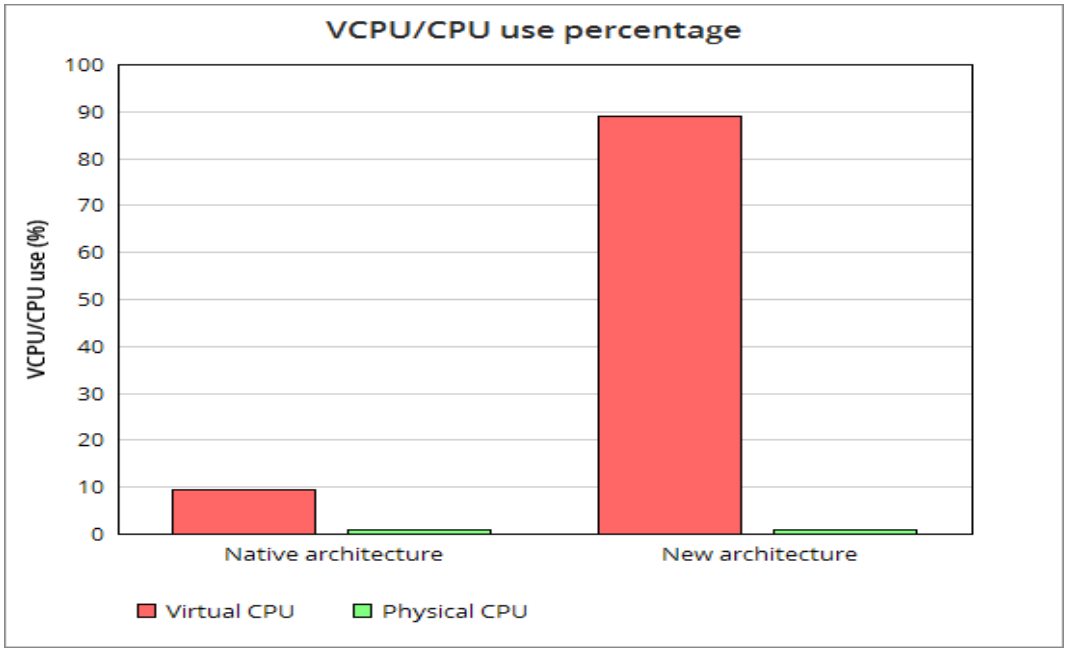

FIGURE 2.15 – Comparaison de l'augmentation (en pourcentage) de l'utilisation des CPU physiques (Dom0) et des CPU virtuelles.

n'impacte pas les cœurs CPU physiques utilisés par le *Dom0*, avec seulement 1% d'overhead. Par ailleurs, le *DomU* (architecture native) n'utilise que 9.4% des ressources vCPU et, à l'instar du DD, n'affecte pas les performances du *Dom0*.

2.6 Conclusion

Atteindre les capacités maximales des interfaces réseau est devenu une réalité ces dernières années, grâce à l'émergence de frameworks logiciels, permettant un traitement accéléré des paquets de données. Néanmoins, cela se fait au détriment d'une surexploitation des ressources matérielles (CPU, mémoire, etc.) de l'équipement. Cet inconvénient est d'autant plus influent dans une plateforme virtuelle, où ces ressources (limitées) sont partagées entre plusieurs VM. De ce fait, nous avons proposé, durant ce travail, une nouvelle architecture Xen pour la plateforme MNet, intégrant OpenDataPlane (ODP) au sein d'un domaine virtuel privilégié, appelé *Driver Domain* (DD). DD permet de relier les VM invitées de MNet aux NIC sous-jacentes et traiter tout le trafic de données transitant par lui. De plus, un certain nombre de processeurs virtuels (vCPU) sont alloués à ODP pour un packet processing accéléré.

Cette nouvelle architecture Xen améliore les performances réseau de la plateforme MNet (débit, nombre de paquets traités à la seconde, etc), de 15% en moyenne grâce à ODP. En outre, l'utilisation des vCPU dans le DD permet d'éviter une surexploitation des ressources CPU physique sous-jacentes. Par ailleurs, Cette nouvelle architecture Xen sera déployée dans la partie cœur (EPC) du réseau mo-

bile. Cependant, ces expérimentations ont été effectuées sur des équipements à moyenne puissance, possédant des ressources limitées en termes de processeur (4 cœurs CPU) et de carte réseau (1 Gbit). Par conséquent, nous souhaitons étendre les tests de notre architecture Xen sur des machines plus performantes, embarquant un grand nombre de cœurs CPU et intégrant des NIC à grande capacité (10 Gbit/s, 40 Gbit/s, etc.). Par ailleurs, nous projetons d'associer *Intel* DPDK à ODP, pour de meilleures performances.

Dans le chapitre 3, nous allons introduire l'architecture du Cloud Radio Access Network (C-RAN), en décrivant le principe de séparation de l'unité radio (RRU) et de l'unité traitement (BBU), en présentant ses composants ainsi que ses modèles de centralisation et enfin en énumérant ces avantages.

Chapitre 3

Présentation de l'architecture Cloud Radio Access Network (C-RAN)

Sommaire

3.1 Introduction

L'importante augmentation du nombre d'utilisateurs mobiles ainsi que l'accroissement de la quantité de données échangées, à travers les infrastructures de télécommunications, ont poussé les opérateurs et les manufacturiers réseau à concevoir une nouvelle génération de réseaux mobiles, plus flexible, plus efficiente en terme de consommation d'énergie et qui pourrait répondre aux contraintes de performance et de QoS, imposées par les nouvelles solutions IT (IoT, réalité augmentée, etc.). Ainsi, Cloud

Radio Access Network (C-RAN) a été définie pour être la nouvelle architecture d'accès radio des réseaux mobiles 5G. Introduit par IBM sous le nom de Wireless Network Cloud (WNC) [41], le C-RAN a été par la suite décrit dans [7] comme une architecture « green », conçue pour pallier à certaines limitations présentes dans l'architecture RAN (LTE-A) actuelle. Pour ce faire, le C-RAN devrait relever les défis suivants :

— réduire les dépenses d'investissement (CAPEX), liées à l'acquisition de sites d'installation, à l'achat d'équipements composant une cellule, à l'installation du réseau d'accès, ainsi que celles d'exploitation (OPEX), liées aux opérations de maintenance, d'entretien des équipements et de la mise à jour des systèmes logiciels,
— réduire la forte consommation d'énergie dans les réseaux d'accès, causée par le déploiement d'un grand nombre de stations de base, nécessaire pour offrir une meilleure couverture. Pour cela, le C-RAN doit intégrer des solutions logicielles et matérielles, permettant d'optimiser la consommation d'énergie tout en améliorant les performances réseau. De plus, de nouvelles stratégies de déploiement d'antennes doivent être pensées pour améliorer la couverture du réseau,
— augmenter la capacité du réseau d'accès afin de prendre en charge le nombre croissant d'utilisateurs ainsi que les données générées par leurs terminaux mobiles,
— améliorer la gestion de la mobilité des utilisateurs et l'impact de leur consommation de données sur les stations de base, se trouvant dans leur zone géographique. Pour ce faire, il est nécessaire de prendre en considération le taux d'utilisation de chaque station de base, en fonction de la période de temps, pour déterminer les zones à forte densité d'utilisateurs. Cette solution aurait également pour objectif d'optimiser la consommation d'énergie en mettant en veille active les stations localisées dans des zones à très faible densité humaine.
— ouvrir le réseau d'accès radio aux fournisseurs de services pour le déploiement de solutions logicielles qui permettraient, d'une part, d'offrir aux utilisateurs de nouveaux services, contenus et applications avec de meilleures QoE et de performances possibles, et d'autre part, de pouvoir traiter et acheminer le trafic de données au plus près des terminaux, sans forcément passer par le réseau cœur de l'opérateur (EPC), ce qui permettrait de réduire le coût de transport des données vers l'EPC.

Nous présenterons, dans ce chapitre, le concept du C-RAN. nous décrirons notamment les éléments qui composent son architecture. Ensuite, nous énumérerons les avantages apportés par le C-RAN. Enfin, nous détaillerons les deux modèles définis pour la centralisation du système radio dans le C-RAN.

3.2 Composants de l'architecture C-RAN

Dans l'architecture actuelle des réseaux LTE, l'eNodeB représente l'élément principal de la partie d'accès radio. Il se charge de la réception/transmission des signaux radio à partir de/vers les terminaux utilisateurs (UE), de la gestion des ressources radio (Radio Resource Management (RRM)), de la compression des entêtes IP, de la sécurité, etc. Distribués aux extrémités du réseau, les eNodeB servent de stations de base aux UE ainsi que de relais vers l'EPC et internet. Dans l'architecture C-RAN (figure

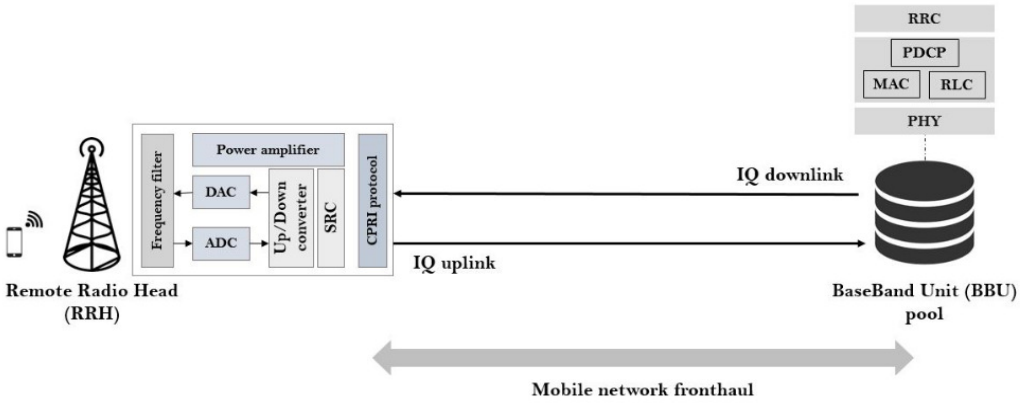

FIGURE 3.1 – Composition de l'architecture C-RAN.

3.1), les eNodeB sont remplacés par de simples antennes radio qui permettent de réceptionner et/ou de transmettre les signaux radio. Nommées Remote Radio Head/Unit (RRH/RRU), ces antennes sont déployées au plus près de l'utilisateur et de ses terminaux. Les fonctions de traitement de l'interface radio, appelées BaseBand Unit (BBU), sont par ailleurs séparées des antennes pour être rassemblées et centralisées au sein d'un nouvel élément appelé BBU pool ou Cloud BBU (C-BBU). A la manière d'un Cloud, le pool BBU permet de traiter tous les signaux reçus à partir des différentes RRH reliées à lui. Cet élément va alors exécuter les fonctions BBU en offrant de meilleures performances et en consommant moins d'énergie que les eNodeB qu'il remplace dans le réseau.

3.2.1 Remote Radio Head (RRH)

La RRH est composée des éléments suivants :

— un filtre permettant d'enlever certaines bandes ou fréquences du signal transitant par l'antenne,
— un amplificateur qui permet d'accentuer la puissance du signal,
— un convertisseur de signal analogique-numérique (ADC) et numérique-analogique (DAC),
— un convertisseur up/down et un module de conversion d'échantillons du signal (Sample Rate Conversion (SRC)),
— un module d'encapsulation des données In-phase/Quadrature (IQ) dans des paquets, utilisant le protocole d'interface, Common Public Radio Interface (CPRI) [42] ou d'autres standards tels que, Open Base Station Architecture Initiative (OBSAI) [43] et Open Radio Interface (ORI) [44].

Les RRH sont généralement déployées aux extrémités du réseau mobile et sont localisées à une distance, pouvant atteindre les 40 km de la BBU [45]. Elles permettent d'amplifier, filtrer et récupérer/transmettre les signaux radio à partir/vers des UE ou du pool BBU. Les RRH n'exécutent pas de calcul en local et un certain nombre d'entre elles peuvent être connectées à une passerelle appelée RRH

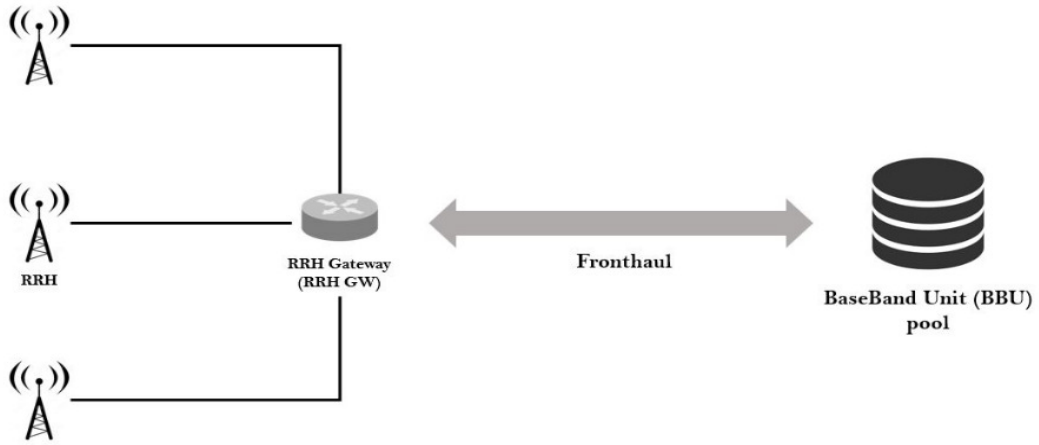

FIGURE 3.2 – Architecture C-RAN utilisant la RRH gateway.

Gateway (RRH GW) (figure 3.2), qui se charge de grouper les signaux reçus par les antennes pour les transmettre au pool BBU.

3.2.2 Fronthaul

L'ensemble des RRH sont reliées au pool BBU à travers un réseau FH, lequel permet de transporter les données IQ généralement encapsulées par le protocole CPRI à travers un lien point-à-point. Alors que dans une architecture LTE-A, l'échange des signaux entre l'antenne et la BBU se fait au niveau d'un eNodeB (en interne), cet échange peut être étendu sur plusieurs kilomètres dans une architecture C-RAN. Par conséquent, l'infrastructure réseau du fronthaul doit prendre en considération les contraintes strictes, citées ci-dessous :

— une large bande passante, avec un débit supérieur à 10 Gbit/s, en raison de la charge supplémentaire apportée par le transport des données IQ à travers le fronthaul [7],
— une faible latence de transmission nécessaire pour le traitement des signaux par la couche PHY du réseau d'accès (se trouvant dans le pool BBU). De ce fait, selon [7], le Round Trip Time (RTT) ne doit pas excéder les 5 μs. Néanmoins, selon [46], cette valeur variera entre 100 μs et 400 μs et dépendra des équipements utilisés ou/et de l'architecture adoptée par les opérateurs,
— une synchronisation des horloges de l'équipement radio (Radio Equipment (RE)), se trouvant dans la RRH et celle du contrôle de l'équipement radio (Radio Equipment Control (REC)) qui se trouve dans le pool BBU,
— une faible fraction de fréquence du signal Fractional Frequency Offset (FFO) qui ne doit pas dépasser les 2 parties par million (Part Per Million (ppm)) [47], ainsi qu'une faible gigue (variation de latence au niveau du FH).

Afin de répondre à ces contraintes techniques, un certain nombre de solutions réseau ont été proposées pour intégrer le fronthaul et transporter le trafic radio entre les RRH et le pool BBU. Les avantages et les inconvénients de chaque solution sont décrits ci-dessous.

3.2.2.1 Fibre optique dédiée (fibre noire)

La fibre optique dédiée peut fournir un débit puissant (jusqu'à 100 Gbit/s) sur de longues distances (jusqu'à 50 km) et elle représente la solution la plus efficace, en terme de performance. De ce fait, elle permet de relier les antennes radio au pool BBU et de satisfaire toutes les contraintes liées au transport des données IQ à travers le fronthaul. En outre, elle peut être déployée de manière rapide et ne nécessite pas d'équipements particuliers pour son fonctionnement. Cependant, la fibre optique dédiée reste coûteuse à étendre et à maintenir, en raison de la forte utilisation de ressources de fibre et pourrait par conséquent ne pas atteindre un des principaux objectifs, définis pour le C-RAN qui est celui de la réduction des CAPEX/OPEX sur l'infrastructure réseau.

3.2.2.2 Multiplexage en longueur d'onde (Wavelength Division Multiplexing (WDM))

WDM est une technique qui permet de transporter plusieurs longueurs d'ondes optiques (40-80) sur une seule et même fibre [45], en utilisant différents canaux (chaque canal correspond à une longueur d'onde) grâce à l'intégration de modules de transpondeur au niveau des deux parties du C-RAN. L'avantage de WDM est son coût d'exploitation réduit comparé à celui de la fibre optique dédiée et son optimisation des ressources optiques. En outre, cette technique garantit des débits élevés et une faible latence, paramètres indispensables pour le transport du trafic de données à travers le FH. Il existe deux différents types de multiplexage en longueur d'onde :

- **Dense Wavelength Division Multiplexing (DWDM) :** pouvant obtenir entre 80 et 160 canaux optiques, méthode utilisée lorsque les espacements entre les longueurs d'onde doivent être très courts. L'avantage de DWDM, est son utilisation d'ondes amplifiables, sans passer par un amplificateur électrique, ce qui permet d'éviter la reconversion du signal optique en signal électrique et vice versa. Néanmoins, cette technique reste coûteuse car elle nécessite une régulation de la température du laser d'émission de longueurs d'onde, en raison de l'intervalle court, présent entre chaque impulsion du signal procédée par le laser.
- **Coarse Wavelength Division Multiplexing (CWDM) :** technique moins coûteuse, car ne nécessitant pas de régulation de la température des lasers (les intervalles d'impulsions sont plus longs que pour DWDM) et n'utilisant pas d'amplificateur optique pour une raison d'incompatibilité. Ainsi, cette méthode peut effectuer du multiplexage avec des composants moins chers mais sur de courtes distances.

WDM est considéré actuellement comme la solution la plus appropriée pour l'architecture C-RAN en raison de ces performances, sa scalabilité, ainsi que son coût amoindri (notamment la méthode CWDM). Par conséquent, son déploiement dans le fronthaul à travers un réseau de transport optique (Optical Transport Network (OTN)) est envisagé par un grand nombre d'opérateurs [48].

3.2.2.3 Micro-ondes

Pouvant être utilisée sur de larges bandes de fréquence (70-80 GHz [49]), la transmission par micro-ondes représente une alternative intéressante pour le transport des données IQ à travers le FH. Cette

solution peut être mise en place dans des stades, campus universitaires ou encore dans des aéroports, où il est généralement difficile de déployer de la fibre optique. Les micro-ondes offrent une faible latence et permettent d'atteindre des débits de 10-40 Gbit/s sur de courtes distances (inférieures à 1.5 km [50]). Néanmoins, Ces performances restent en deçà de celle définies pour le fronthaul C-RAN. Les micro-ondes restent tout de même une solution envisageable pour une utilisation dans le backhaul [51], notamment dans certaines régions où il peut être compliqué d'établir des connexions filaires.

3.2.2.4 Ethernet

Ethernet représente un standard réseau mature, largement déployé dans les infrastructures existantes. Alors qu'il constitue le médium de transport principal dans le backhaul, l'adoption de ce protocole dans le FH est sérieusement envisagée par un grand nombre d'opérateurs et d'équipementiers qui voient en Ethernet un moyen peu coûteux et efficace de relier les RRH au pool BBU. En outre, Ethernet présente de nombreux avantages pour le transport des données, tels que :

— utilisation de fonctions d'opération, d'administration et de maintenance (Operations, Administration and Management (OAM)) pour la détection et la localisation d'erreurs, le diagnostic des données ainsi que l'apport d'informations liées aux performances du réseau,
— faible coût d'exploitation des équipements Ethernet actuellement déployés dans les réseaux d'accès des opérateurs, ainsi que la possibilité de les partager avec d'autres infrastructures pour le transport des données,
— possibilité de contrôler, gérer et orchestrer le trafic de données transitant dans le réseau grâce à l'utilisation de technologies de type SDN,
— transmission des données à très haut débit, pouvant atteindre les 100 Gbit/s entre deux extrémités du réseau [52].

Bien qu'Ethernet soit une alternative intéressante et moins coûteuse que WDM, cette solution possède néanmoins des limitations, pouvant restreindre son utilisation dans le FH. La première limitation est l'absence de synchronisation des horloges qui est nécessaire lors de la transmission des données IQ entre l'émetteur et le récepteur qui peut conduire à l'apparition d'erreurs de transmission et/ou à la corruption du signal source. La seconde limitation concerne les performances de latence et de gigue apportées par Ethernet qui restent en deçà du seuil requis pour le transport de données dans le FH. Ces performances peuvent par ailleurs se dégrader si des nœuds ou switches intermédiaires se trouvent entre les RRH et la BBU.

Afin d'adapter Ethernet à la transmission radio, un certain nombre de travaux a été mené pour répondre aux contraintes imposées par le transport des données IQ à travers le protocole CPRI. Nous citons, comme exemple, le standard Synchronous Ethernet (SyncE), protocole défini par l'Union Internationale des Télécommunications (International Telecommunication Union (ITU)) sous les recommandations G-8261, G-8262 et G-8264. Le principe de SyncE est de fournir une synchronisation stricte des horloges avec une fraction de fréquence de $\pm$ 4.6 ppm, contre $\pm$ 100 ppm pour la norme Ethernet IEEE 802.3, dans la couche physique du modèle Open Systems Interconnection (OSI), nécessaire pour le transport des fréquences radio à travers un réseau Ethernet [53]. D'autres standards ont par ailleurs été définis prenant en considération cette synchronisation des transmissions à travers Ethernet,

tels que, IEEE 1588 Precision Time Protocol (PTP) (synchronisation dans la couche liaison) [54] et IETF Network Time Protocol (NTP) (synchronisation dans la couche réseau) [55]. Une nouvelle spécification du protocole CPRI, appelée eCPRI [56], est à l'étude pour intégrer la 5ème génération de réseaux mobiles. Proposé par de grands noms de l'industrie des télécommunications, tels que, Huawei, Ericsson, Nokia et NEC, eCPRI aura pour objectif de réduire l'utilisation de la bande passante (de 10 fois), d'exploiter les infrastructures existantes à travers Ethernet, d'adopter Ethernet comme la principale solution de transport dans le fronthaul, etc. Des projets de recherche ont été également lancés pour faire converger les architectures 4G et 5G afin de réduire les coûts CAPEX/OPEX liés à la mise en place d'une nouvelle architecture de réseau d'accès. Nous citons notamment, le projet Elastic Network qui prévoit d'adopter Ethernet comme protocole de transport des données radio à travers le FH.

3.2.3 Baseband Unit (BBU)

Appelée également Radio Cloud Center (RCC) dans certaines architectures, la Baseband Unit (BBU) est composée d'un ensemble de couches et de fonctions, servant à traiter les signaux radio, transitant par le réseau d'accès. Alors qu'elle est associée à l'unité radio (RU/RH) dans un eNodeB dans l'architecture RAN de la 4G, la BBU est séparée de cette même RU (appelée alors RRH/RRU) dans une architecture C-RAN pour être, par la suite, regroupée avec d'autres unités de traitement au sein d'un même élément, appelé pool BBU. Ce dernier est généralement virtualisé et centralisé dans un serveur (ou un mini Datacenter) se trouvant entre les RRH et le réseau cœur (EPC). Le pool BBU fournit ainsi les ressources physiques nécessaires pour le traitement des signaux.

La centralisation des unités BBU en un seul pool offre un grand nombre d'avantages, tels que, le partage des ressources virtuelles et matérielles entre les différentes BBU, l'efficience énergétique grâce à la possibilité de désactiver le fonctionnement de certaines unités BBU selon leur charge de traitement, la simplification des opérations de configuration des cellules, l'augmentation de la capacité de calcul pour le traitement des signaux radio, la gestion améliorée de la mobilité des UE ainsi que des opérations de handover, etc. Néanmoins, afin de fournir ces performances, il est nécessaire d'intégrer des algorithmes d'ordonnancement des ressources partagées en sein du pool BBU et de déterminer le bon placement de ce dernier par rapport aux RRH déployées dans le réseau.

Une unité BBU intègre les trois couches protocolaires composant l'eNodeB : la couche physique (PHY/L1), la couche (Media Access Control (MAC)/L2) ainsi que la couche contrôle (Radio Resource Control (RRC)/L3). Les fonctions de chacune de ces couches sont présentées ci-dessous.

3.2.3.1 Couche physique

La couche physique représente la couche basse (L1) de la BBU. Elle se charge de la transmission/réception des données IQ à travers le fronthaul grâce à l'exécution des fonctions suivantes :

- **Codage de canal :** permet de détecter les changements (erreurs) produits sur les bits de données reçues grâce à l'utilisation du code Cyclic Redundancy Check (CRC). Ces erreurs sont par la suite corrigées grâce à l'utilisation des fonctions Forward Error Correction (FEC), lequel

permet d'ajouter des bits redondants à la donnée ainsi que de la fonction Automatic Repeat Request (ARQ) qui, en cas d'existence d'erreur, va solliciter une retransmission du paquet de données erroné.
- **Adaptation de lien :** nommée également Adaptive Modulation and Coding (AMC). Cette fonction permet de moduler les bits à transmettre avec un *code rate* correspondant à la qualité du canal de transmission.
- **Multiple-Input Multiple-Output (MIMO) :** technique de multiplexage permettant de transmettre les données vers plusieurs RRH en simultané.
- **Modulation multiporteuse :** permet de transmettre les données modulées sur de multiples porteuses en même temps.
- **Mesures radio :** estime la qualité du signal et du canal de transmission, ainsi que la puissance des signaux émis par les différentes RRH (cellules).
- **Synchronisation :** notamment celle des horloges entre la RRH et la BBU.
- **Signalisation des informations de contrôle :** entre l'UE et les réseaux d'accès.

3.2.3.2 Couche MAC

La Couche MAC, appelée également couche 2 (L2), est composée de trois sous couches qui interviennent pour la transmission des paquets de données (le DP) et pour le CP. La description de chacune de ces sous couches est donnée ci-dessous :

- **Packet Data Compression Protocol (PDCP) :** intégrant plusieurs fonctions. La sous couche PDCP permet de compresser les entêtes des paquets, en utilisant des mécanismes tels que Robust Header Compression (RoHC) pour améliorer l'efficacité spectrale sur des services de type voix sur IP (VoIP). Elle permet également de chiffrer et de protéger l'intégrité des données de signalisation RRC. Enfin, cette sous couche se charge de la détection et de la suppression des doublons (paquets de données) qui apparaissent généralement lorsqu'un handover se produit entre deux cellules.
- **Radio Link Control (RLC) :** assure le contrôle des liaisons de données grâce à des fonctions de détection d'erreurs, de retransmission de paquets (en cas d'erreur), d'ordonnancement de ces derniers et d'optimisation des transmissions grâce à l'utilisation de fenêtres d'émission/réception.
- **Medium Access Control (MAC) :** permet d'accéder et d'adapter la transmission des données au canal de transport correspondant. Pour cela, la sous couche MAC utilise des fonctions d'allocation dynamique des ressources radio, de maintien de synchronisation (en lien montant) et de priorisation des flux de données. La sous couche MAC utilise également le mécanisme Hybrid Automatic Repeat Request (HARQ) pour une transmission fiable des paquets.

3.2.3.3 Couche Radio Resource Control (RRC)

Troisième couche de la pile protocolaire, RRC assure la configuration et le contrôle des couches sous-jacentes. Elle est ainsi responsable des fonctions suivantes :

— diffusion et décodage des informations système liées aux couches Access Stratum (AS) et Non-Access Stratum (NAS),
— gestion des envois/réceptions de radio messagerie (paging),
— gestion des connexions entre les couches RRC de l'UE et de l'unité de traitement,
— etablissement, configuration et maintenance des connexions point à point des Radio Access Bearer (RAB),
— gestion des fonctions et des clefs de sécurité,
— gestion et contrôle de la mobilité des UE en mode connecté et en mode veille,
— gestion de la QoS et des mesures de l'UE.

3.3 Avantages du C-RAN

Le C-RAN présente de nombreux avantages, comparé aux architectures d'accès radio actuelles. Nous allons énumérer dans cette section les principaux avantages apportés par le C-RAN aux utilisateurs ainsi que pour les opérateurs mobiles :

- **Efficience énergétique :** grâce à la centralisation de la partie traitement de l'interface radio. L'architecture C-RAN permet de réduire considérablement (de dix fois) le nombre de stations de base déployées, par rapport aux architectures actuelles [7]. Cela implique une baisse significative de la consommation d'énergie qui est habituellement élevée au niveau de ces stations de base. En outre, le partage des ressources entre les différentes unités BBU présentes dans le pool offre la possibilité de mettre en veille certaines de ces unités, lorsque leur charge de travail est faible. Les traitements restant peuvent alors être migrés vers d'autres unités au sein du pool pour plus d'efficience énergétique. Le C-RAN réduit également la quantité d'énergie consommée, lors de la transmission du signal, grâce à une gestion plus efficace des interférences et un déploiement plus dense des RRH qui vont alors former de petites cellules dans le réseau.
- **Réduction des CAPEX-OPEX :** le regroupement des BBU en un petit nombre de pools permet de réduire leur coût de déploiement, de gestion et de maintenance. De plus, la distribution des RRH aux extrémités du réseau est plus simple à effectuer en comparaison avec les eNodeB actuels, ce qui diminuera les dépenses réalisées par les opérateurs dans les infrastructures d'accès. Le C-RAN pourra également exploiter les installations déjà présentes dans les réseaux mobiles, notamment les infrastructures Ethernet déployées dans les réseaux LTE. Enfin, tel que ne l'avons mentionné précédemment, l'efficience énergétique de l'architecture C-RAN permettra de considérablement baisser les coûts liés à la consommation d'électricité et d'autres énergies nécessaires au fonctionnement du réseau.
- **Amélioration de la capacité spectrale :** l'architecture C-RAN facilitera la mise en place de nouveaux mécanismes de détection et d'atténuation d'interférences intercellulaires, grâce notamment à la simplification du partage d'informations (au sein du pool BBU), liées aux signaux envoyés par les UE actifs et les données liées à l'état des canaux (Canal State Information (CSI)). Cela garantie une plus grande efficacité spectrale, en comparaison avec l'architecture LTE.
- **Adaptabilité au trafic de données non-uniforme :** la virtualisation des unités de BBU au sein

d'un pool centralisé permet de gérer l'allocation et le partage des ressources entre les BBU. Par conséquent, les périodes de forte/faible charge, à un endroit particulier du réseau, sont mieux traitées par le C-RAN. En outre, l'utilisation des ressources est optimisée grâce à la possibilité d'activer/désactiver des unités BBU ou encore de répartir le traitement dans tout le pool. Enfin, le load balancing intégré au FH permet de prendre en charge le trafic de données non-uniformes, généré par des UE en mouvement.

- **Amélioration des performances réseau :** en raison de la séparation des parties RRH et BBU dans le C-RAN, il est devenu nécessaire de mettre en place une infrastructure réseau performante dans au sein du FH. En particulier, en termes de bande passante et de latence. Cela profitera donc aux utilisateurs qui auront accès à des débits élevés (dépassant les 10 Gbit/s) et à de nouveaux services, lesquels pourront exploiter ces performances et offrir une meilleure QoE aux les utilisateurs.

3.4 Modèles de centralisation du C-RAN

La division effectuée entre l'unité radio (RRH) et l'unité de traitement (BBU) dans l'interface radio est la principale innovation apportée par le C-RAN. Bien que cette séparation apporte de nombreux avantages (voir la Section 3.3), elle impose cependant des contraintes techniques strictes, notamment en termes de latence et de gigue. Dans [7], deux modèles de division ont été définis afin d'adapter l'architecture C-RAN aux infrastructures réseau existantes et aux technologies utilisées pas ces dernières. Ces deux modèles diffèrent principalement dans le nombre de couches protocolaires centralisées dans le pool BBU. De ce fait, il existe deux différentes « centralisations », correspondant à chacun des deux modèles.

3.4.1 Centralisation complète (*Full centralization*)

Dans ce modèle (figure 3.3 (A)), les couches PHY, MAC et RRC, séparées de la RRH, sont centralisées dans le pool BBU. La centralisation complète représente le modèle de division optimal dans une architecture C-RAN, telle qu'elle a été définie dans [7]. Néanmoins, son implémentation nécessite la prise en compte des conditions suivantes :

— une large bande passante pour le transport des données IQ entre les RRH et le pool BBU distant,
— une faible latence dans le canal de transmission (au niveau du FH), afin de garantir notamment le bon fonctionnement du mécanisme HARQ (couche L2), pour une transmission fiable des données ainsi qu'un traitement efficace des signaux radio en temps réel,
— une synchronisation stricte des horloges, entre la RRH et la couche physique qui est centralisée dans la BBU, pour le transport des paquets CPRI. Cela implique la présence d'une faible gigue dans le FH et l'utilisation de protocoles pouvant supporter cette synchronisation,
— un débit élevé de transmission de données dans le FH, généré par le protocole CPRI. P.ex., une transmission sur une bande passante de 20 MHz génère au minimum 14.7 Gbit/s de données [7].

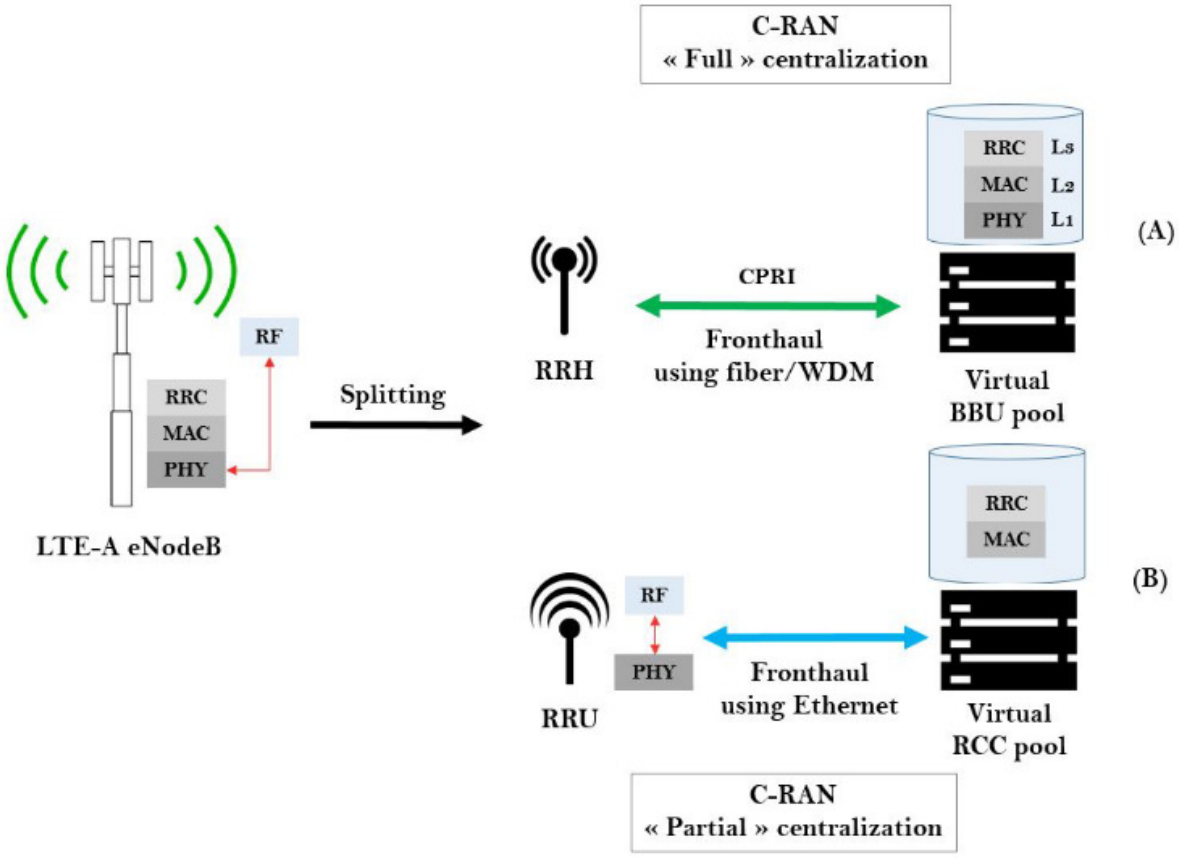

FIGURE 3.3 – Modèles de « centralisation » des couches protocolaires dans une architecture C-RAN.

Les conditions décrites ci-dessus exigent le déploiement d'une infrastructure réseau robuste dans le fronthaul qui permettrai de fournir les performances requises pour la transmission des signaux radio, à partir des RRH vers le pool BBU. Une telle infrastructure peut s'avérer coûteuse pour les opérateurs de télécommunications qui seront ainsi contraints de déployer de nouveaux équipements et de nouvelles liaisons pour adapter leur réseau à l'architecture C-RAN. En outre, seule une infrastructure utilisant de la fibre optique (dédiée ou pas) avec la technique de multiplexage WDM pourra supporter une centralisation complète [48]. Cependant, ce modèle de division présente un grand nombre d'avantages qui sont donnés ci-dessous :

— un déploiement simplifié et moins coûteux des antennes radio (stations de base) qui ne nécessiteront plus l'utilisation de ressources de calcul pour le traitement des signaux radio aux extrémités du réseau d'accès,
— un partage optimisé des ressources (physiques et virtuelles) dans le pool BBU, pour un meilleur traitement des signaux/données reçues, impliquant les traitements collaboratifs intercellulaires ainsi qu'une scalabilité accrue (ajout de la puissance de calcul),
— des performances élevées, en termes de débit, latence, gigue, etc. Cela offrira aux fournisseurs

d'accès/services une infrastructure réseau pouvant héberger et supporter un grand nombre d'applications, nécessitant une bonne qualité de service (jeux vidéo, streaming, etc.),
— possibilité de mettre en place des algorithmes de répartition de charge optimisés entre les différentes BBU, au sein d'un pool,
— implémentation de techniques avancées de communications radio, telle que Coordinated Multipoint (CoMP) [57].

3.4.2 Centralisation partielle (partial centralization)

Bien que la centralisation totale représente le modèle de division optimal pour une architecture C-RAN, son implémentation demeure complexe, en raison des contraintes techniques qu'elle impose et du coût que son déploiement implique au niveau du FH. Les opérateurs ont donc réfléchi à un modèle de centralisation moins exigeant en termes de performances et qui pourra être mis en place sur les infrastructures réseau existantes. Appelée centralisation « partielle » (figure 3.3 (B)), cette solution alternative consiste à grouper uniquement les couches protocolaires supérieures à la couche PHY (L1) dans le pool BBU. La couche L1 est alors déplacée vers les RRH où ses fonctions permettront de traiter les signaux radio reçus aux extrémités du réseau. Cette division partielle réduit l'utilisation de la bande passante de 20-50% par rapport à une centralisation complète, grâce notamment au transport des signaux démodulés à travers le FH [7]. En outre, la charge des données est également considérablement réduite au niveau du médium de transport. Enfin, les contraintes de performances (latence, gigue, etc.) imposées par la centralisation totale sont atténuées dans ce modèle de division.

Le principal avantage de la centralisation partielle demeure le coût de son déploiement/maintenance (OPEX/CAPEX) dans le FH qui est inférieur au modèle de centralisation complète. Il n'est en effet plus nécessaire d'établir un réseau de fibre optique pour connecter les RRH au pool BBU, en raison des contraintes moins fortes requises par la centralisation partielle. Par conséquent, les infrastructures réseau déjà établies peuvent être réadaptées ou même directement réutilisées pour déployer un C-RAN. Des projets sont par ailleurs menés par les opérateurs de télécommunications dans le but de concevoir de nouvelles méthodes et architectures qui permettraient d'exploiter des solutions largement déployées telles qu'Ethernet pour l'établissement d'un médium de transport dans le FH. Nous avons notamment mentionné précédemment la spécification eCPRI qui a pour but d'adapter le standard CPRI à une utilisation à travers Ethernet. Une nouvelle architecture C-RAN a été également proposée, appelée Next Generation Fronthaul Interface (NGFI), cette architecture adopte un division partielle des couches protocolaires de l'interface radio, ainsi qu'une solution de transport basée sur de l'Ethernet au niveau du FH. NGFI sera présentée plus en détail dans le chapitre 4.

La centralisation partielle possède néanmoins un certain nombre de limitations en comparaison à la centralisation complète. Ces limitations sont les suivantes :

— consommation d'énergie élevée aux extrémités du réseau d'accès, en raison de la présence de ressources de calcul, utilisées par la couche L1 pour le traitement des signaux radio au niveau des RRH. Le déplacement de ces ressources vers les stations de base, impliquera un déploiement plus coûteux et plus complexe, en comparaison avec la centralisation complète,

— flexibilité réduite, en termes de partage de ressources, entre les unités BBU au sein du pool. Cela entraîne une baisse d'efficacité dans le traitement collaboratif entre les cellules du réseau,
— performances réseau affaiblies au niveau du FH, pouvant impacter le déploiement de certains services sensibles à la latence réseau,
— utilisation inefficace de certaines fonctions avancées de communication radio telles que CoMP, ainsi qu'une interaction complexifiée entre la couche physique et la couche L2, notamment lors de l'exécution du mécanisme HARQ.

3.5 Conclusion

A travers ce chapitre, nous avons exposé une vue détaillée du Cloud Radio Acceess Network (C-RAN), en présentant tout d'abord le principe de base de ce concept qui consiste à séparer l'unité radio (Remote Radio Head (RRH)) de l'unité de traitement (BaseBand Unit (BBU)), généralement regroupées en un seul élément (eNodeB) dans une architecture RAN-LTE traditionnelle. En outre, nous avons énuméré les principaux défis à relever par le C-RAN, en perspective des besoins exprimés par la nouvelle génération de réseaux mobiles, en termes de performance, flexibilité, coût et efficience énergétique. Nous avons par ailleurs décrit les fonctions de chacun des éléments composant le C-RAN, en détaillant les solutions pouvant être intégrées dans les RRH, BBU et le réseau FH. Nous avons également indiqué les avantages apportés par cette architecture aux opérateurs, d'une part, notamment en termes de flexibilité, coût et partage des ressources, ainsi qu'aux utilisateurs, d'autre part, en termes de QoS, performance et disponibilité. Enfin, nous avons présenté les deux modèles de centralisation définis dans le C-RAN, la centralisation complète et la centralisation partielle, en décrivant leurs architectures respectives, en détaillant le schéma de division de la couche L1 entre la RRH et la BBU ainsi qu'en donnant les avantages et les limitations de chacun de ces modèles.

Dans le chapitre 4, nous allons présenter deux architectures C-RAN partiellement centralisées que nous avons définies dans le cadre du projet Elastic Network. Nous proposons d'intégrer respectivement un réseau TRILL ainsi qu'un réseau SDN dans le FH de chacune des deux architectures. En outre, une comparaison des performances réseau sera réalisée, entre nos deux propositions, afin de déterminer leurs capacités/limitations, par rapport aux exigences de débit, latence, gigue, etc. imposées par la 5G.

Chapitre 4

Conception d'une architecture C-RAN partiellement centralisée au sein de la plateforme MNet

Sommaire

4.1 Introduction

La centralisation de la BBU dans le C-RAN apporte certains avantages tels que **1)** le partage d'un grand nombre de ressources pour un traitement efficace du signal et **2)** l'apport de gains de multiplexage. En outre, deux modèles de centralisation, nommés « centralisation complète (*Full centralization*) » et « centralisation partielle (*partial centralization*) », ont été définis afin d'adapter l'architecture C-RAN aux différentes technologies radio/réseau existantes. Bien que la centralisation complète représente le modèle de division (*splitting*) optimal entre les RRH et les BBU, permettant de grouper les couches L1/L2/L3 dans le pool BBU, son implémentation requiert le déploiement, dans le FH, d'une infrastructure réseau robuste qui fournit des performances élevées, notamment en termes de latence et de débit. De plus, le transport des données IQ à travers le FH nécessite une large bande passante ainsi qu'une synchronisation des horloges/fréquences entre les RRH et les BBU. Par conséquent, l'utilisation de la fibre optique semble indispensable pour mettre en place un tel modèle, ce qui pourrait rebuter certains opérateurs de télécommunications, peu enclins à investir davantage sur de nouvelles infrastructures de fibre optique, bien que le coût engendré par ces dernières tend à diminuer dans les années à venir.

La centralisation partielle est ainsi conçue pour atténuer l'utilisation de la bande passante, réduire les contraintes de latence/synchronisation ainsi que minimiser les CAPEX/OPEX. Un certain nombre d'opérateurs pensent par ailleurs à adopter ce modèle de division afin de réutiliser leurs infrastructures existantes et standardiser le transport du trafic de données à travers Ethernet, dans des réseaux fronthaul/backhaul unifiés. Par conséquent, une nouvelle interface de FH a vu le jour, nommée Next Generation Fronthaul Interface (NGFI). NGFI est basée sur un modèle de centralisation partielle, dans lequel la BBU est renommée en Radio Cloud Center (RCC) et un certain nombre de RRH sont interconnectées grâce à la Radio Aggregation Unit (RAU). Le couple RRU-RAU définit alors un Remote Radio System (RRS) qui représente la station de base dans un C-RAN basé sur l'interface NGFI. En outre, NGFI devrait adopter un réseau switché permettant de transporter des données entre les RRS et le pool RCC, au lieu d'une liaison point-à-point adoptée généralement dans un modèle de centralisation complète.

Dans ce chapitre, nous allons présenter une nouvelle architecture C-RAN, basée sur une interface NGFI, dans laquelle les fonctions RCC et RAU seront virtualisées au sein de la plateforme MNet. De plus, nous allons déployer, au sein d'un FH switché, deux solutions d'interconnexion réseau qui sont TRILL et SDN. Dans cette seconde solution, un contrôleur est virtualisé et centralisé dans le pool RCC, fournissant, via le protocole OpenFlow [58], une vue globale des nœuds/switches établis dans le FH et permettant de gérer le transport des données radio. A travers cette contribution, nous avons pour objectif de : **1)** implémenter une architecture C-RAN réaliste et flexible qui permet d'atténuer les contraintes (large bande passante, faible latence, etc) imposées par une centralisation complète. Pour cela, nous allons adopter Open Air Interface (OAI), un logiciel open source qui fournit des éléments de l'EPC, des fonctions de la couche protocolaire 3GPP ainsi que des modèles de *splitting* fonctionnels de l'interface radio ; **2)** réduire les dépenses CAPEX/OPEX en réutilisant les infrastructures réseau

existantes (équipements réseau, liaisons Ethernet, etc.) pour le transport des données à travers le FH ; **3)** montrer, à travers nos expérimentations, que les données IQ paquetisées peuvent être transportées sur plusieurs sauts (à travers plusieurs nœuds) ; **4)** garantir le transport simultané des données radio ainsi que d'autres types de trafic au sein du même réseau FH, pouvant être partagé par différentes entités (opérateurs, fournisseurs de services, etc.).

Ce chapitre sera organisé de la manière suivante : la Section 4.2 introduira l'interface NGFI. La Section 4.3 présentera l'architecture C-RAN proposée, en décrivant chacun de ses éléments ainsi que les solutions utilisées (SDN, TRILL, etc.). La Section 4.4 sera consacrée à l'évaluation des performances obtenues par le réseau FH de notre architecture C-RAN. Une comparaison entre TRILL et SDN sera par ailleurs effectuée. Enfin, nous terminerons ce chapitre par une conclusion et les perspectives de notre travail.

4.2 Présentation de Next Generation Fronthaul Interface (NGFI)

NGFI est une nouvelle interface de FH (figure 4.1), pensée pour réduire les coûts de déploiement d'une architecture C-RAN ainsi que pour fournir un réseau de transport flexible, entre les unités radio et les unités de traitement. Pour cela, NGFI est basée sur un modèle de division partiellement centralisée, dans lequel certaines fonctions radio, présentes dans le pool BBU, sont décentralisées vers les stations de base, selon le type de *splitting* utilisé [59]. Par conséquent, les éléments du C-RAN sont redéfinis en RCC, pour la partie de traitement (BBU) et en RRS, pour la partie radio (RRH). En outre, les connexions point-à-point, entre les RRH et les BBU, sont abandonnées au sein de l'interface NGFI, au profit d'un réseau composé d'un ensemble de nœuds (switches, routeurs, etc.), interconnectant plusieurs RRS et RCC et transportant les données radio paquetisées grâce à l'utilisation d'un protocole adapté.

4.2.1 Eléments de l'interface NGFI

Les différents éléments de l'architecture C-RAN/NGFI sont décrits ci-dessous :

- **Remote Radio System (RRS) :** représente la partie radio et la station de base du C-RAN. Un RRS est composé d'une ou plusieurs antennes (RRU) qui sont reliées à une RAU. Ces deux composants (RRU et RAU) peuvent par ailleurs être placés au sein d'un même équipement physique ou bien séparés. Une RAU intègre des fonctions de la couche PHY pour un traitement initial du signal radio (p.ex. modulation/démodulation du signal). De plus, elle joue le rôle de passerelle, reliant les RRU au pool RCC et encapsulant les données radio dans des trames Ethernet ou dans des paquets, selon la solution adoptée dans le FH. Enfin, certaines fonctions de la RAU peuvent être virtualisées aux extrémités du réseau.
- **Radio Cloud Center (RCC) :** remplace la BBU dans une architecture C-RAN classique. A l'instar des BBU, plusieurs RCC peuvent être regroupés en un seul pool, implémenté au sein d'un serveur ou un mini Data center. Une unité RCC intègre les fonctions des couches L2/L3 et permet de traiter les données radio, transmises par les RRS connectés à son pool.

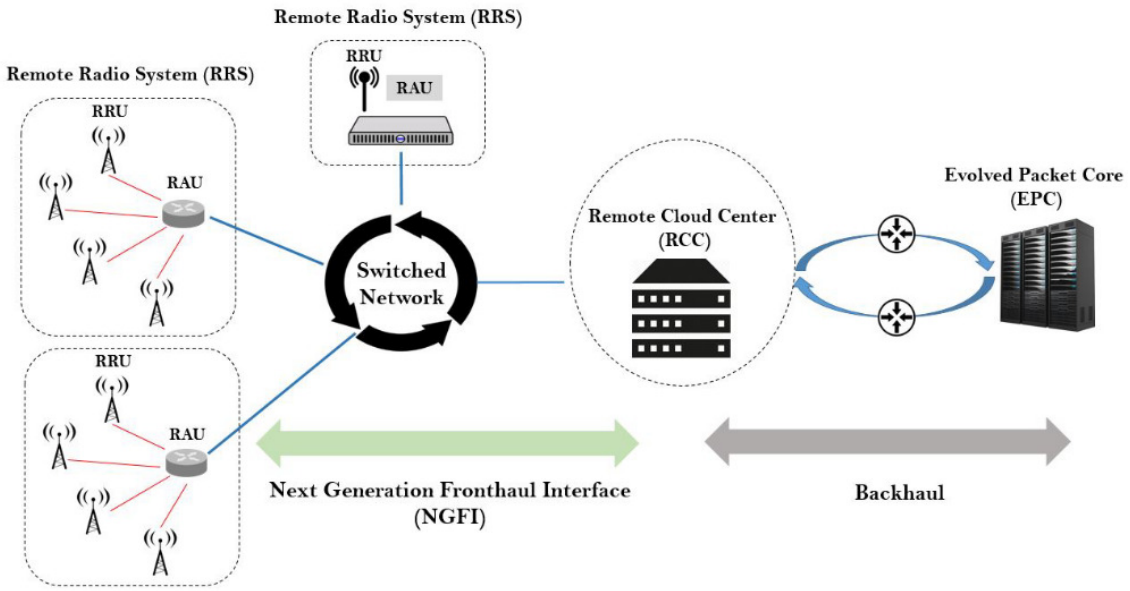

FIGURE 4.1 – Architecture C-RAN basée sur un fronthaul NGFI.

- **Fronthaul :** bien qu'aucun standard n'a encore été défini pour l'interconnexion des éléments du C-RAN, NGFI devrait adopter des protocoles basés sur l'échange de paquets, à travers un réseau switché (Ethernet, MPLS, routage IP, etc.). En outre, NGFI fournit un ensemble de fonctionnalités pour une exploitation optimisée de la bande passante, telles que le multiplexage statistique ou la modulation adaptative à la charge de données. Enfin, NGFI devrait s'adapter aux architectures LTE existantes mais aussi aux technologies prévues pour la 5G. Pour cela, la transmission Ethernet est recommandée pour plus de flexibilité et d'efficacité [11].

4.2.2 Couches logiques de l'interface NGFI

L'interface NGFI peut être décomposée en trois couches logiques distinctes (figure 4.2) nommées : couche de données (*Data Layer*), couche d'adaptation des données (*Data Adaptation Layer*) et couche physique de transport (*Physical Carrier Layer*). La couche de données intègre les fonctions liées au contrôle et à la synchronisation des données radio/utilisateur. Le fonctionnement de cette couche dépend du type de *splitting* implémenté, de la bande passante utilisée et de l'architecture d'accès radio déployée (4G/5G). La couche d'adaptation des données concerne les fonctions de transmission et de conversion des données radio qui sont adaptées au médium de transport sous-jacent. Cette couche permet notamment de moduler/démoduler, coder/décoder ou même encapsuler les données radio dans des trames Ethernet ou des paquets CPRI, pour être transportées par la suite à travers le FH. En outre, des fonctions de type HARQ sont utilisées pour garantir une transmission fiable des données. Enfin, la couche physique de transport représente l'infrastructure réseau sous-jacente, déployée dans le FH. Différentes solutions sont par ailleurs proposées pour cette couche, telles qu'Ethernet, WDM, fibre optique, etc.

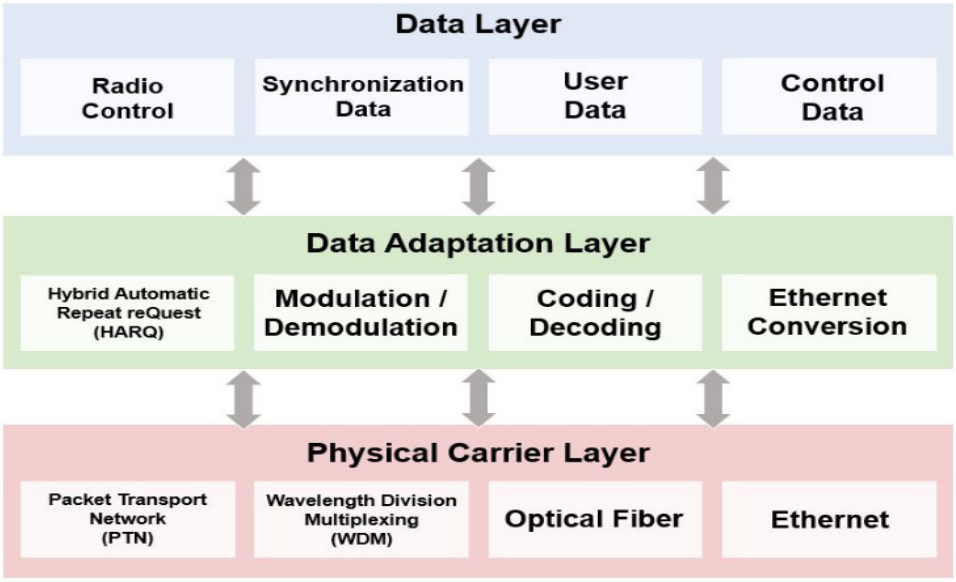

FIGURE 4.2 – Couches logiques de l'interface NGFI.

4.2.3 Avantages de NGFI

Outre les avantages de la centralisation partielle dans l'architecture C-RAN, l'interface NGFI apporte des fonctionnalités supplémentaires, résumées ci-dessous :

- adaptabilité de la transmission des données radio, en fonction de la charge endurée par le réseau FH, à travers l'utilisation du multiplexage statistique, pour plus d'efficacité,
- réduction de la bande passante utilisée entre les RRS et les RCC,
- adoption d'un médium de transport dans le FH, basé sur Ethernet, afin de simplifier le déploiement/maintenance des infrastructures, partager les ressources du réseau FH entre les opérateurs et réduire les OPEX/CAPEX,
- définition d'un réseau unifié pour le fronthaul et le backhaul,
- virtualisation des fonctions réseau, pour une architecture C-RAN flexible qui pourra s'adapter à l'intégration de nouveaux services 5G.

4.2.4 Modèles de division de l'interface RRS-RCC

Cinq modèles de division fonctionnelles sont définis pour l'interface NGFI, afin d'adapter cette dernière aux contraintes imposées par les technologies de radio/transport utilisées dans le FH. La figure 4.3 illustre par ailleurs ces différentes divisions, appliquées sur les couches protocolaires d'accès radio des unités RRU-RCC. Notons que les parties, représentées en vert dans la figure, concernent les couches de traitement « utilisateur » et sont dépendantes du trafic de données, tandis que les parties représentées en jaune concernent les couches de traitement « cellule » et sont indépendantes du trafic.

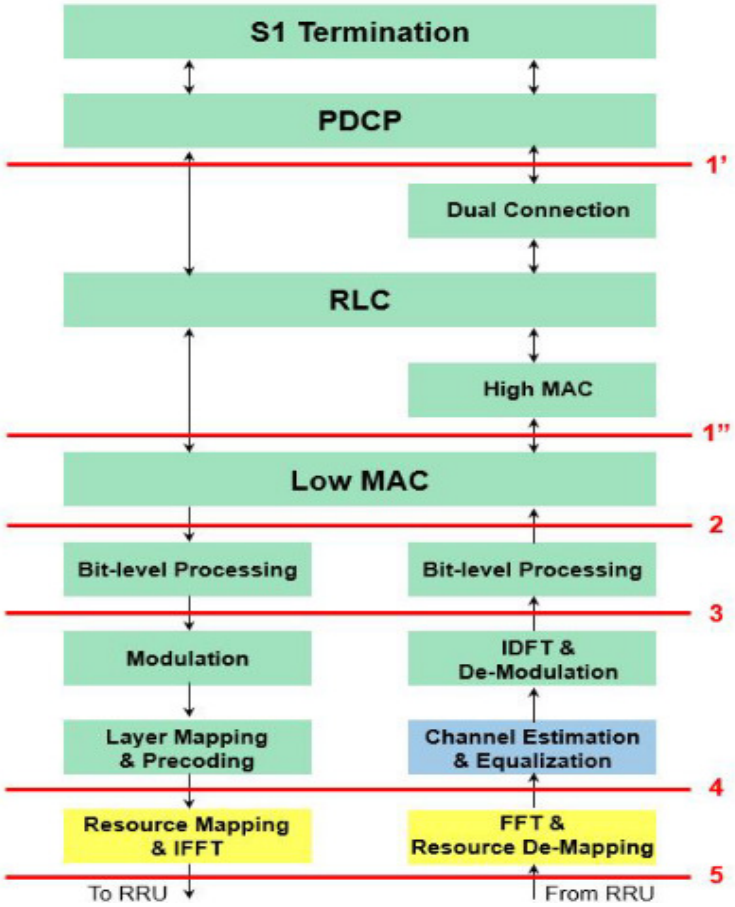

FIGURE 4.3 – Modèles de division fonctionnelle pour les unités RRU-RCC dans NGFI.

Les modèles de division sont décrits ci-dessous :

- **Division interne de la couche L2 (1' et 1") :** est adaptée pour les réseaux LTE actuels ainsi que pour les réseaux 5G. La division de la couche 2 est simple à implémenter et peut supporter différentes technologies radio. En outre, grâce à sa flexibilité, ce modèle s'adapte à la variation de la capacité des stations de base, aux différents niveaux de densité des cellules, aux nombreux supports de transport réseau existants (notamment Ethernet) et peut supporter un intervalle de latence, compris entre 100 μs et 10 ms [11].
- **Division entre les couches L2/L1 (2) :** les apports additionnels de cette division par rapport au modèle 1 sont la centralisation du traitement et de l'ordonnancement dans le pool RCC, grâce à la séparation nette entre la couche PHY et la couche MAC. De plus, ce modèle 2 offre la possibilité de faire varier la largeur de la bande passante, selon le volume du trafic transitant par le FH. Enfin, elle permet l'intégration d'algorithmes de *scheduling* complexes pour le partage et la gestion des ressources.
- **Division interne de la couche PHY (3 et 4) :** dans ces modèles de division, les fonctions de

codage/décodage (modèle 3) et de modulation/démodulation (modèle 4) sont centralisées dans l'unité RCC. Ces divisions permettent l'implémentation de techniques de coopération et de transmission avancées (CoMP, Massive MIMO, etc.). En contrepartie, l'utilisation de la bande passante augmente avec des valeurs *Downlink-Uplink*, avoisinant les 125-464 Mbit/s pour le modèle 3 et 498-2689 Mbit/s pour le modèle 4 [11].

- **Division Baseband/RF (5) :** représente le modèle de division optimal qui est adopté dans des architectures C-RAN complètement centralisées. Cette division permet une séparation totale entre la RRU (qui intègre seulement les fonctions Radio Frequency (RF)) et le RCC/BBU et peut supporter un grand nombre d'antennes radio. Néanmoins, ce modèle nécessite une large bande passante (9830 Mbit/s pour le *downlink* et 9830 Mbit/s pour le *uplink*) et une très faible latence. Bien que cette division n'est actuellement pas envisagée dans l'interface NGFI, son implémentation sera réalisée dans un futur proche.

4.3 Implémentation d'une architecture C-RAN basée NGFI dans MNet

Implémenter une architecture C-RAN flexible et à bas coût, pouvant répondre aux besoins de la nouvelle génération de réseaux mobiles. Tel est l'objectif à atteindre à travers la contribution, présentée dans cette section. Pour cela, nous proposons d'intégrer l'interface NGFI dans notre architecture pour une utilisation réduite de la bande passante, une exploitation avancée des infrastructures réseau existantes, notamment Ethernet, une réduction des CAPEX/OPEX, etc. Mais d'abord nous allons résumer les principaux travaux menés par les secteurs industriels et académiques dans ce domaine de la conception C-RAN.

4.3.1 Travaux connexes

Depuis sa présentation en 2015, l'interface NGFI a fait l'objet de nombreux travaux, définissant les modèles de division fonctionnelle à adopter, les technologies d'accès radio à implémenter ou encore l'architecture C-RAN adaptée à cette interface. Nous citons notamment la première conception d'une architecture C-RAN basée sur NGFI, présentée par *Chih-Lin et al.* [60], dans laquelle les BBU sont déployées à la manière de fonctions réseau virtualisées. De plus, une solution de type Radio over Ethernet (RoE) est proposée pour le transport des données radio à travers un FH basé sur une solution Packet Transport Network (PTN).

Huang et al. ont par ailleurs proposé une architecture C-RAN basée sur un modèle de division hybride [61]. Deux différentes unités radio, nommées RRH et RRH1, sont déployées dans le réseau d'accès (selon la densité de la région). La RRH est associée à un modèle de C-RAN complètement centralisé et est caractérisée par le faible coût de son déploiement (car ne contient que les fonctions RF), mais requiert une large bande passante ainsi qu'une faible latence. Tandis que la RRH1, associée à un modèle partiellement centralisé, impose moins de contraintes, du point de vue des performances mais reste plus coûteuse à déployer (car elle intègre des fonctions de traitement du signal radio). De plus, les auteurs ont défini un algorithme glouton (*greedy*) qui permet de minimiser les coûts de mise en place des deux types de RRH. Bien que les résultats des simulations, présentées dans le papier, soient

intéressants du point de vue coût, l'implémentation d'une telle architecture C-RAN demeure complexe en pratique.

Chang et al. ont présenté FlexCRAN, une architecture C-RAN basée sur la plateforme logicielle OAI et utilisant Ethernet comme solution de transport réseau dans le FH [62]. Afin d'évaluer la robustesse de cette architecture, les auteurs ont défini trois indexes de performance (Key Performance Index (KPI)) qui sont : le débit de données/RTT (performance réseau), la charge de traitement du matériel utilisé et la QoS/latence du plan de données. Les résultats obtenus à partir des expériences réalisées ont montré qu'Ethernet peut être une solution de FH efficace et robuste, selon les KPI évalués dans ce travail, bien que des pertes de paquets et des dégradations des performances sont parfois observées au sein du FH.

L'unification des réseaux backhaul/fronthaul fait partie des objectifs établis pour les réseaux 5G. De ce fait, des propositions d'architectures ont été faites dans [63] et notamment dans [64], où *De La Olivia et al* présentent une architecture C-RAN, dans laquelle les RRH, le pool BBU et l'EPC sont interconnectés à travers un réseau à grande capacité (*carrier grade*) nommé Xhaul. Deux principaux éléments ont été définis dans cette architecture. Le premier élément, nommé Xhaul packet Forwarding Element (XFE), représente la partie centrale du Xhaul et fournit une infrastructure fronthaul/backhaul, intégrant des technologies avancées de transport de données (fibre optique, 10Gbit Ethernet, WDM, etc.), des routeurs/switches à haute capacité, etc. Le second élément, appelé Xhaul Control Infrastructure (XCI), permet de gérer l'infrastructure XFE sous-jacente grâce à des contrôleurs SDN. En outre, XCI garantit la qualité de transmission, la QoS ainsi que de grandes performance dans le réseau de transport, notamment en fournissant des mécanismes de load balancing, des technologies d'ingénierie de trafic, des gestionnaires d'interférence radio, etc.

Les travaux susmentionnés apportent des solutions intéressantes, permettant pour certaines de définir l'architecture réseau d'accès radio pour la 5G. Nous souhaitons, à travers notre contribution, les compléter en implémentant une architecture C-RAN réaliste qui pourra exploiter les technologies et les infrastructures réseau existantes pour le transport de données hétérogènes à travers le FH.

4.3.2 Description des éléments de l'architecture C-RAN

La conception de notre architecture C-RAN (figure 4.4) s'appuie sur un modèle de centralisation partielle, utilisant le *splitting B* de la plateforme OAI, présenté dans [65], qui correspond à la division interne de la couche PHY (modèle 4), décrite dans la Section 4.2. Les éléments et les technologies, implémentés dans notre architecture, sont présentés ci-dessous.

4.3.2.1 Nœud Remote Radio System (RRS node)

Le nœud RRS représente la station de base de notre architecture C-RAN et est implémenté au sein d'une machine physique (MNetBox), embarquant un système Software Defined Radio (SDR), constituant l'unité radio (RRU). De plus, un environnement MNet-Xen est mis en place pour l'hébergement,

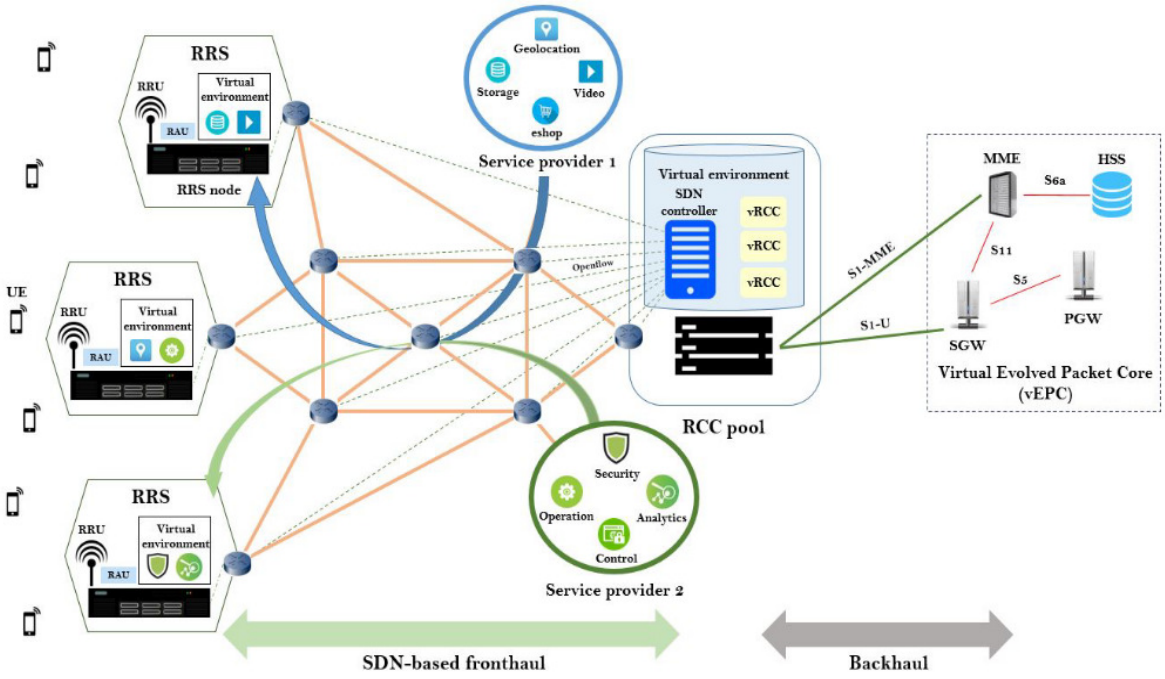

FIGURE 4.4 – Architecture C-RAN, basée sur une interface NGFI.

sous forme de VM ou de VNF, d'applications/services pouvant être déployés par différents opérateurs, fournisseurs de services ou encore par des tierces parties.

L'unité d'agrégation RAU, développée sur OAI, est directement virtualisée au sein du domaine privilégié *Dom0* (à la place du *DomU*), pour des raisons de performances et d'accès rapide aux ressources physiques de la MNetBox. En effet, la liaison RRU-RAU est sensible aux opérations d'e/s pouvant se produire entre les *DomU* et les couches physiques sous-jacentes. Cela peut notamment provoquer des erreurs de transmission/réception des signaux radio ou encore des pertes de connections entre les deux unités. Par conséquent, la RAU est directement connectée à la RRU à travers un port USB3.0 et peut exploiter sans restriction les ressources de mémoire/CPU de la machine physique. L'inconvénient d'une telle implémentation est l'impossibilité de migrer la RAU entre deux nœuds RRS car l'environnement Xen ne supporte pas actuellement la migration du *Dom0*.

L'intégration d'un environnement virtuel, au sein du nœud RRS, apporte de nombreux avantages tels que :

— le déploiement simplifié et rapide de services orientés utilisateurs (serveur vidéo, serveur de jeux, passerelle IoT, etc.) ou opérateurs (contrôleur d'accès, firewall, etc.) aux extrémités du RAN, pour une meilleure QoS, une disponibilité accrue et une meilleure QoE. De ce fait, le nœud RRS peut être une extension d'un serveur Mobile Edge Computing (MEC) [66],
— le partage optimisé des ressources entre les différentes VM instanciées au sein du nœud,
— le coût réduit de déploiement et de maintenance du nœud RRS, grâce à sa configuration simplifiée et minimisée,

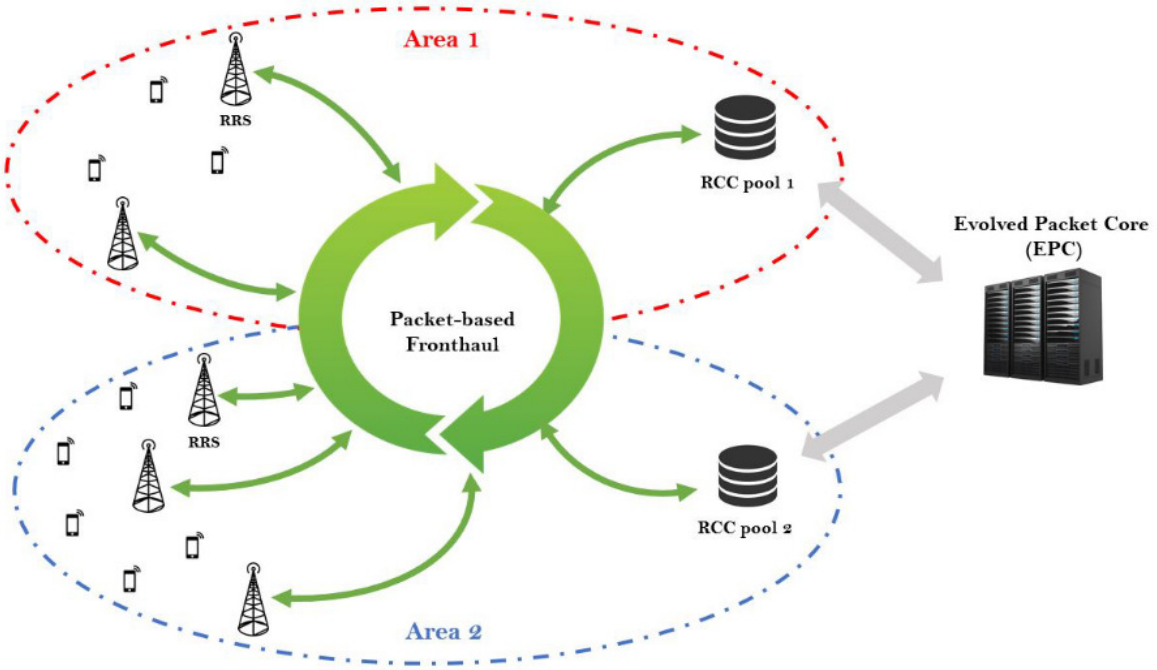

FIGURE 4.5 – Zones du réseau C-RAN gérées par les pools RCC.

— la consommation d'énergie atténuée par la possibilité de mettre en veille les nœuds RRS, localisés dans des zones géographiques à très faible densité d'utilisateurs.

4.3.2.2 Pool Radio Cloud Center (RCC)

Le pool RCC intègre un ensemble d'unités RCC (développées également sur OAI), virtualisées et contenues dans des *DomU* (Xen). A l'instar du RRS, le pool RCC est implémenté au sein d'une MNetBox, dans laquelle il partage les ressources physiques de cette dernière avec d'autres services et VM. En outre, chaque unité RCC est dédiée à un nœud RRS et ne traite que les signaux radio envoyés par ce nœud précis.

Un pool RCC peut être assigné à une zone géographique spécifique (Figure 4.5), dans laquelle il permet de gérer le fonctionnement des nœuds RSS localisés dans cette zone, ainsi que de traiter les opérations de handover à l'intérieur de cette dernière. De plus, les différents pools RCC déployés dans le réseau peuvent communiquer entre eux, à travers le FH, pour coordonner les opérations de handover inter-zones.

4.3.3 Solutions réseau pour le fronthaul

Les éléments de l'architecture C-RAN sont interconnectés à travers un réseau FH, composé d'un ensemble de nœuds (switches, routeurs, etc.) et utilisant des liaisons Ethernet. En outre, nous adoptons deux solutions pour les plans de données et de contrôle qui sont le protocole TRILL et le modèle SDN. Une description de ces deux solutions sont données ci-dessous.

4.3.3.1 Transparent Interconnection of Lots of Links (TRILL)

TRILL est une solution « *carrier grade* » d'interconnexion réseau pour la couche 2 (L2), standardisée par IETF, qui utilise un protocole de routage IS-IS. TRILL intègre des mécanismes de couche 3 (L3) tels que, **1)** le calcul du plus court chemin entre deux extrémités, grâce à l'utilisation de l'algorithme de *Djikstra* (implémenté dans IS-IS) ; **2)** l'implémentation d'un compteur qui évite l'apparition de boucle en se décrémentant à chaque saut effectué par le paquet de données et **3)** la construction d'arborescences logiques, pour la transmission du trafic multicast/broadcast. Cependant, TRILL garde les avantages d'un protocole de niveau 2 tels que, l'auto configuration des nœuds, la vitesse de transmission des données, la faible latence du réseau ainsi que le faible coût de déploiement.

TRILL est implémenté au sein d'un switch particulier, appelé Routing Bridge (RB). Chaque RB est identifié par un « *nickname* » unique de 2 octets (2^{16} possibilités), affecté par le protocole TRILL, prenant alors comme valeur l'abréviation de son *system-ID* dans IS-IS [67], ou bien, assigné directement par l'utilisateur lors de la configuration du RB, possédant ainsi une priorité plus élevée. Toutefois, certaines valeurs de *nicknames* sont réservées et donc ne doivent pas être choisies, notamment les valeurs se trouvant entre 0xFFC0 et 0xFFFF et la valeur 0x0000 qui indique généralement un nickname pas reconnu par le protocole. Par ailleurs, lorsque deux RB se voient affectés le même *nickname*, TRILL sélectionne celui qui possède le *system-ID* le moins prioritaire afin de modifier sa valeur. Enfin, il est possible qu'un RB puisse recevoir plusieurs *nicknames*, lorsque ce dernier est le nœud racine de plusieurs arborescences dans le réseau.

Les RB peuvent être directement déployés dans des réseaux L2/L3, sans pour autant perturber le fonctionnement de ces derniers. En effet, lorsqu'un RB se trouve entre deux routeurs IP interconnectés, ce dernier est considéré comme un simple switch et il est automatiquement ignoré. De même avec des switches L2. De plus, Un RB construit sa table d'acheminement selon le *nickname* de ses voisins (et non plus par rapport aux adresses MAC). Cela permet notamment de réduire le volume des tables et d'optimiser le temps de recherche du nœud de destination.

Pour la détection des stations terminales, le mécanisme End-Station Address Distribution Information (ESADI) a été initialement intégré dans le RB. ESADI annonce dans le réseau, par une opération d'inondation (*flooding*), une partie ou la totalité des adresses MAC des terminaux reliés au RB. Toutefois, pour des raisons d'efficacité (ESADI utilisant un procédé lourd), ce mécanisme est abandonné en faveur d'une technique d'apprentissage des adresses MAC lors des opérations de décapsulation des trames Ethernet [68].

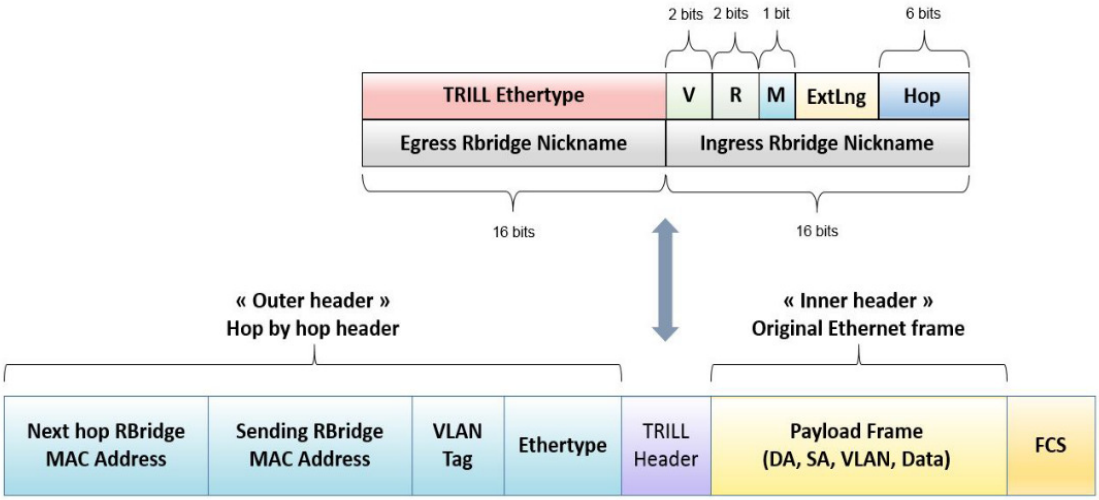

FIGURE 4.6 – Encapsulation d'une trame Ethernet dans un réseau TRILL.

Entête TRILL

L'entête TRILL (figure 4.6) est composé des champs suivants :

- ***nickname* du RB d'entrée (*Ingress*) (16 bits) :** représente le *nickname* du RB se trouvant à l'entrée du réseau.
- ***nickname* du RB de sortie (*Egress*) (16 bits) :** représente le *nickname* du RB se trouvant à la sortie du réseau.
- **Compteur de saut (*Hop count*) (6 bits) :** utilisé pour prévenir contre l'apparition de boucles dans le réseau. Ce compteur est décrémenté, à chaque saut, par le RB recevant la trame. Si le compteur atteint la valeur de zéro, la trame sera alors automatiquement rejetée.
- ***Multi-destination flag M* (1 bit) :** ce bit indique si la trame doit être acheminée à un ensemble de destinations, à travers un arbre de distribution (M = 1). Dans ce cas précis, le *nickname* de la racine (*root*) de l'arbre de distribution est spécifié dans le champ *nickname* du RB de sortie de l'entête TRILL.
- **Version V (2 bits) :** du protocole TRILL.
- **Bits réservés (2 bits) :** peuvent être exploités par un opérateur pour intégrer de nouvelles fonctionnalités au protocole.
- **Longueur de l'extension de l'entête TRILL (*ExtLng*) (5 bits) :** aussi appelé longueur des options (*Opt-length*), ce champ permet de spécifier la longueur des options utilisées dans l'entête.

Lors de l'encapsulation d'une trame Ethernet dans le réseau TRILL, un second entête est ajouté, nommé entête de sortie ou entête saut-par-saut (*Hop-by-Hop*). Ce dernier contient l'adresse MAC du RB transmettant la trame, l'adresse Mac du prochain RB, le VLAN *tag* ainsi que l'*Ethertype* qui est de 0x22FF. Les valeurs de cet entête de sortie (notamment les champs des adresses MAC des RB) sont modifiées à chaque saut par les RB.

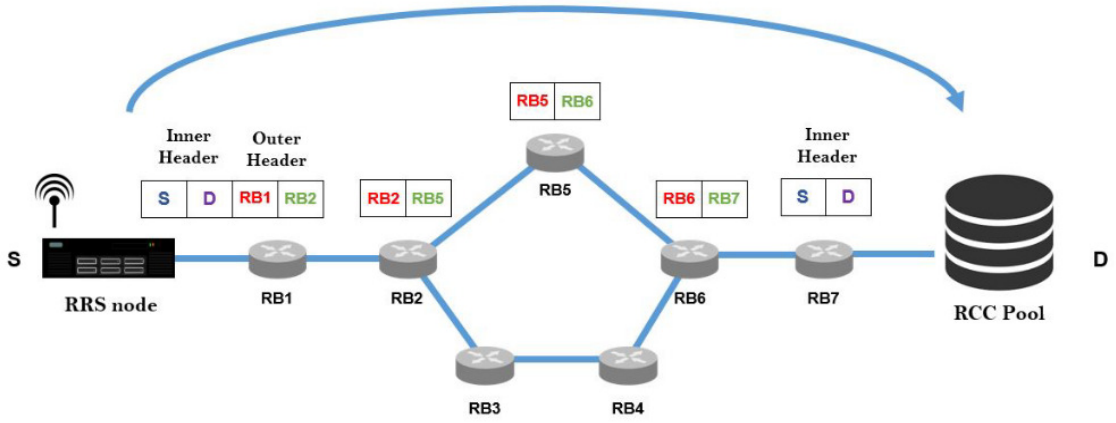

FIGURE 4.7 – Processus d'acheminement des paquets dans un réseau TRILLL.

Acheminement des données

Les étapes de constitution d'un réseau TRILL et d'acheminement des trames Ethernet à travers ce réseau sont décrites ci-dessous :

— chaque RB du réseau exécute le protocole à états de liens IS-IS pour déterminer, à travers l'algorithme *Djikstra*, les plus courts chemins vers d'autres RB et ainsi construire l'arborescence multicast,
— un RB est ensuite désigné (Designated Routing Bridge (DRB)) pour la localisation des stations terminales, à travers le mécanisme d'apprentissage des adresses MAC. Ces adresses sont alors diffusées à travers l'arbre pour détecter les RB connectés à chacune des stations terminales,
— à l'entrée d'une trame dans le réseau, le RB d'entrée encapsule cette dernière, en ajoutant les entêtes TRILL et externes. La trame est alors acheminée, à travers le réseau, jusqu'au RB de sortie (connecté à la destination) qui décapsule les deux entêtes ajoutés dans le réseau TRILL. Ce processus d'acheminement est par ailleurs illustré dans la figure 4.7.

Multi-destination et multipath

Il est possible d'implémenter une solution de multipath dans TRILL, grâce à l'algorithme Equal Cost MultiPath (ECMP) [69] qui permet d'envoyer un flux de données, possédant la même destination, simultanément sur plusieurs chemins de même longueur.

TRILL permet également de mettre en place une transmission multi-destination en exploitant les arbres logiques construits durant la phase de signalisation. Ainsi, le trafic de données est acheminé vers plusieurs destinations en simultané, à travers les différentes branches (liaisons) de l'arbre. En outre, une charge de données peut être répartie sur plusieurs arbres (load balancing), possédant le

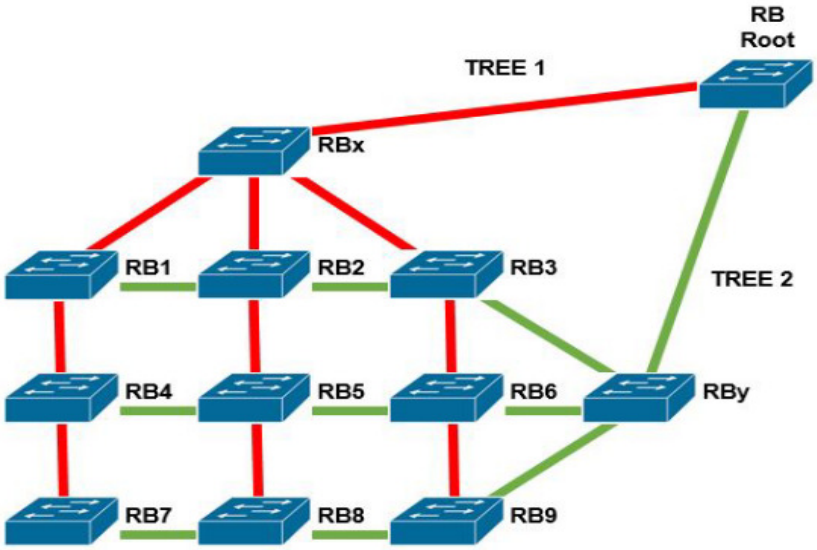

FIGURE 4.8 – Construction d'arborescences multicast possédant un RB racine commun.

même RB racine et partageant un certain nombre de feuilles (RB) (figure 4.8). Toutefois, dans ce cas précis, aucune branche (liaison) ne doit être commune à plus d'un arbre.

4.3.3.2 Software Defined Network (SDN)

SDN est une nouvelle architecture réseau, dont le principe est de séparer la couche logique du CP, de l'infrastructure physique sous-jacente (DP). De ce fait, les routeurs, switches et middlebox, présents dans un réseau, sont dépossédés de leur « intelligence » et sont transformés en de simples nœuds d'acheminement du trafic de données. Par ailleurs, le CP est logiquement centralisé au sein d'un contrôleur logiciel qui peut être implémenté sur une machine distante au réseau, réparti sur plusieurs machines (nous parlons alors d'un contrôleur SDN distribué) ou même déployé dans la même machine physique qu'un nœud du réseau. Cependant, les deux plans réseau restent interconnectés et communiquent grâce à une interface, appelée *Southbound*. Ainsi, le contrôleur SDN envoie les règles/instructions d'acheminement du trafic aux switches, directement à travers cette interface, en utilisant un protocole *Southbound* (p. ex. *OpenFlow*). De plus, le CP est relié, à travers une interface *Northbound*, à une couche d'abstraction supérieure qui représente le plan de gestion ou Management Plane (MP). Le MP fournit un ensemble de fonctions, de langages de programmation et d'applications réseau, dédiées au fonctionnement du DP. Les différents éléments de l'architecture SDN sont illustrés dans la figure 4.9 et décrits ci-dessous :

- **Plan de données (Data Plane (DP)) :** représente l'infrastructure physique du réseau, composée d'un ensemble d'équipements/nœuds génériques (routeurs, switches, middlebox, etc.), pouvant être matériels ou virtuels et qui sont dénués de tout contrôle/intelligence sur le trafic transitant

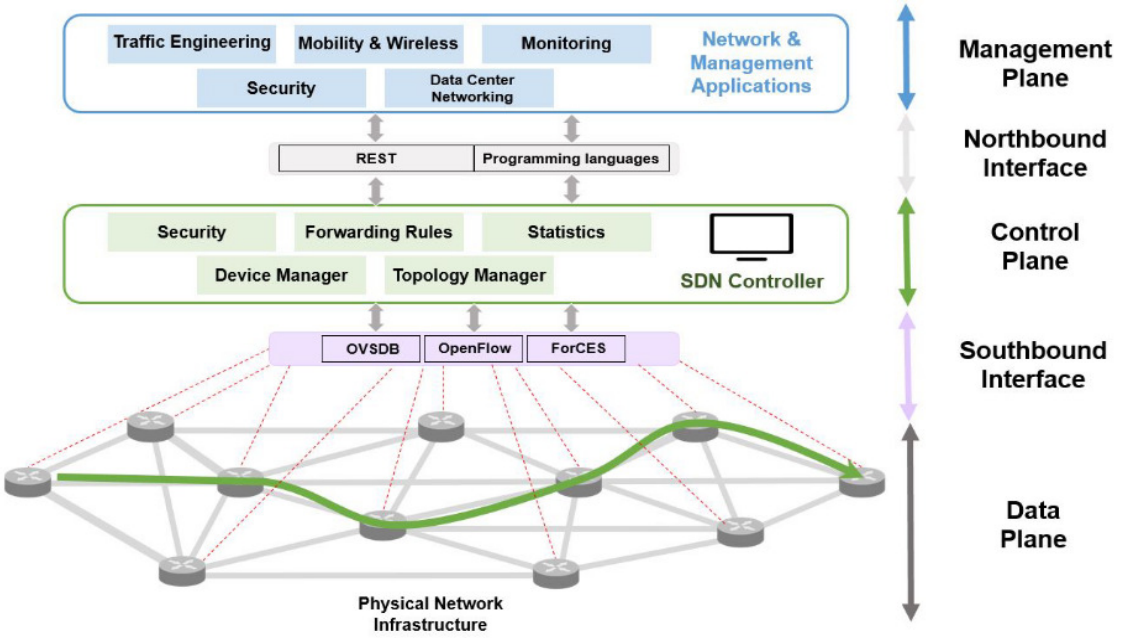

FIGURE 4.9 – Architecture SDN et ses différentes couches d'abstraction.

par le réseau. Ces équipements se chargent de l'acheminement des *flows* de données entre une source et une destination, selon des règles établies et transmises par le contrôleur SDN via l'interface *Southbound*. En outre, dans le cas d'une interface *OpenFlow*, chaque nœud réseau intègre : **1) des tables de *flows*** qui regroupent les règles d'acheminement du trafic ; **2) un canal sécurisé** connectant le nœud au contrôleur et **3) l'interface du protocole *OpenFlow*** (figure 4.10).

- **Interface de programmation *Southbound* :** représente le canal de communication entre le contrôleur SDN et les nœuds réseau. De ce fait, chaque switch envoie, à travers cette interface, des messages d'évènements produits dans le réseau (changement de ports, rupture d'une liaison, etc.), des statistiques relatives aux flows acheminés ou encore un paquet inconnu par le switch (message *Packet-in*). Dans l'autre sens, le contrôleur SDN transmet les règles et les instructions qui devront être appliquées par les switches pour le DP. L'objectif à travers cette interface est de fournir une interopérabilité entre tous les équipements réseau, indépendamment de leurs constructeurs. Bien que *OpenFlow* est le protocole *Southbound* le plus connu, il existe d'autres protocoles pouvant être utilisées, telles que OpenVSwitch DataBase (OVSDB) [70], Forwarding and Control Element Separation (ForCES) [71] ou encore Protocol-Oblivious Forwarding (POF) [72].
- **Plan de contrôle (Control Plane (CP)) :** consiste en une couche d'abstraction qui est gérée par un contrôleur (appelé également Network Operating System (NOS)) logiquement centralisé. Un contrôleur SDN intègre plusieurs fonctions telles que, la détection des nœuds connectés au réseau, la découverte et la gestion de la topologie (notamment la gestion de l'état des liaisons et des nœuds), la configuration et le contrôle de l'acheminement du trafic (calcul des plus courts

chemins, filtrage, etc.), la mise en place de mécanismes de sécurité, etc. En outre, un contrôleur peut être programmé, à travers des API et des plug-ins, pour supporter de nouvelles technologies ou des protocoles déjà établis, tels que Border Gateway Protocol (BGP), NETCONF, etc. Enfin, il existe deux architectures de contrôleur : **1) l'architecture centralisée** qui représente une seule entité physique d'un contrôleur, permettant de gérer des réseaux de moyenne taille. Un contrôleur centralisé est simple à implémenter et fournit des performances élevées. En contrepartie, étant un point individuel de défaillance, il demeure intolérant aux erreurs/pannes. De plus, le contrôleur centralisé est inefficace lorsqu'un trop grand nombre de nœuds sont déployés dans le réseau (faible scalabilité). Parmi les nombreuses implémentations de cette architecture, nous citons *Maestro* [73], *Beacon* [74] ou encore *Floodlight* [75] ; **2) l'architecture distribuée** consiste en un ensemble d'entités de contrôleurs, réparties dans différentes « régions » du réseau. Ces contrôleurs peuvent être gérés de manière centralisée à travers un *cluster* les regroupant, ou bien gérés de manière indépendante. Néanmoins, l'ensemble des contrôleurs garde une vision globale de la topologie, apportée par les différentes mises à jour échangées par les contrôleurs. L'architecture distribuée est tolérante aux fautes/pannes et supporte la scalabilité du réseau mais fournit dans certains cas des données de contrôle peu consistantes, en raison des nombreuses mises à jour appliquées sur les contrôleurs. Cette limitation est cependant corrigée par certains contrôleurs comme Open Network Operating System (ONOS) [76] ou ONIX [77].

- **Interface de programmation *Northbound* :** possède un niveau d'abstraction supérieur à la couche du CP. L'interface *Northbound* représente un environnement logiciel, constitué de langages de programmation et d'API, permettant d'une part, le développement d'applications réseau avancées et, d'autre part, de faire communiquer ces applications avec les couches d'abstractions sous-jacentes. Nous citons comme exemples les API REST [78] et SFNet [79].
- **Plan de gestion (Management Plane (MP)) :** cette couche applicative regroupe l'ensemble des applications réseau, déployées dans l'infrastructure sous-jacentes. Ces applications peuvent être classées dans cinq principales catégories [80] : **1) ingénierie de trafic** qui constituent les applications de load balancing, optimisation réseau, contrôle de congestion, etc. **2) mobilité et sans-fil** qui concerne les gestions de la mobilité des utilisateurs, des ressources radio, des opérations de handover ainsi que toutes les fonctions d'un réseau sans fil/mobile ; **3) monitoring et mesure** qui regroupe les applications d'évaluation des performances (débit, latence, etc.) et de traitement des statistiques transmises par les nœuds réseau ; **4) sécurité** qui intègre des applications de contrôle d'accès, firewalling, détection d'intrusion, etc. **5) Data Center** qui comprend des solutions de stockage, gestion des ressources, orchestration, etc.

OpenFlow

Nous adoptons la solution *OpenFlow* pour relier le contrôleur SDN aux différents nœuds déployés dans le FH de notre architecture C-RAN. Tel que nous l'avons mentionné précédemment, *OpenFlow* est un protocole de l'interface *Southbound* qui définit les règles d'échange d'information entre le DP et le CP dans un modèle SDN. Standardisé par Open Networking Foundation (ONF) [81], *OpenFlow* est considérée actuellement comme la solution référence et peut être supportée par la majorité des

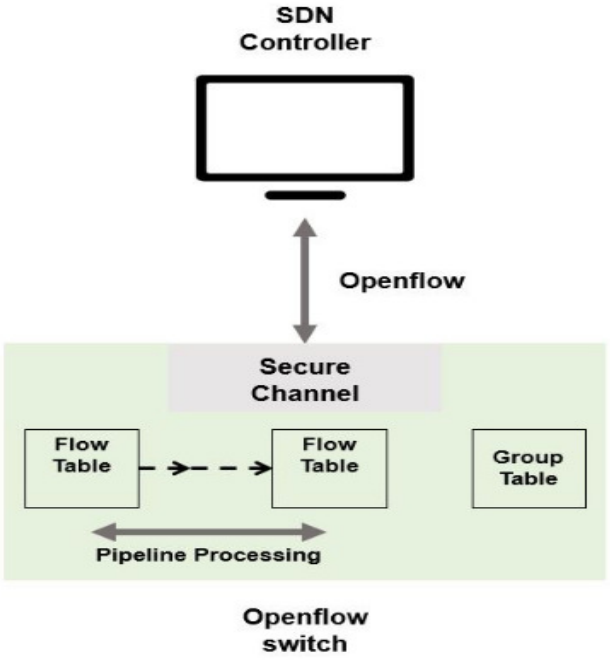

FIGURE 4.10 – Eléments d'un switch *OpenFlow*.

équipements réseau ainsi que des contrôleurs SDN existants. Par ailleurs, l'architecture de *OpenFlow* est constituée de trois principaux éléments :

- **Table de *flows* :** représente une liste d'entrées de *flows*, implémentée au sein du switch *OpenFlow*. Chaque entrée de la table est définie par trois attributs qui sont : **1) la/les règle(s) de matching** de l'entête d'un paquet (p.ex. paquet dont la valeur du port TCP source est égale à 80) ; **2) les actions/instructions** qui correspondent à la règle de matching (p. ex. acheminer des paquets avec le port TCP source 80 vers l'adresse IP 10.0.0.1) et 3) un compteur statistique qui relève le nombre de paquets entrés correspondant à la règle. A partir de la version 1.1.0 du protocole *OpenFlow*, les switches peuvent intégrer plusieurs tables de *flows*, reliées entre elles. De ce fait, le processus de recherche de matching (*lookup)* s'effectue en mode *pipeline*. De plus, un nouveau type de table a été introduit dans cette même version, appelé *Group Table*. Cette table particulière contient des instructions communes et applicables à de multiples *flows* [82]. Le schéma de la table de *flows* est illustré dans la figure 4.11.
- **Tunnel sécurisé :** utilise le protocole Transport Layer Security (TLS) pour sécuriser la communication entre le(s) contrôleur(s) SDN et les switches *OpenFlow* ainsi que pour transmettre les messages d'évènements, de *Packet-in* et les instructions du contrôleur.
- **Protocole de communication :** définit le format d'échange d'informations, via des messages, entre les plans de données et de contrôle.

A l'entrée d'un paquet dans le switch *OpenFlow*, ce dernier procède à une vérification, dans les tables de *flows*, de l'existence d'une règle qui correspond aux champs de l'entête du paquet, nous

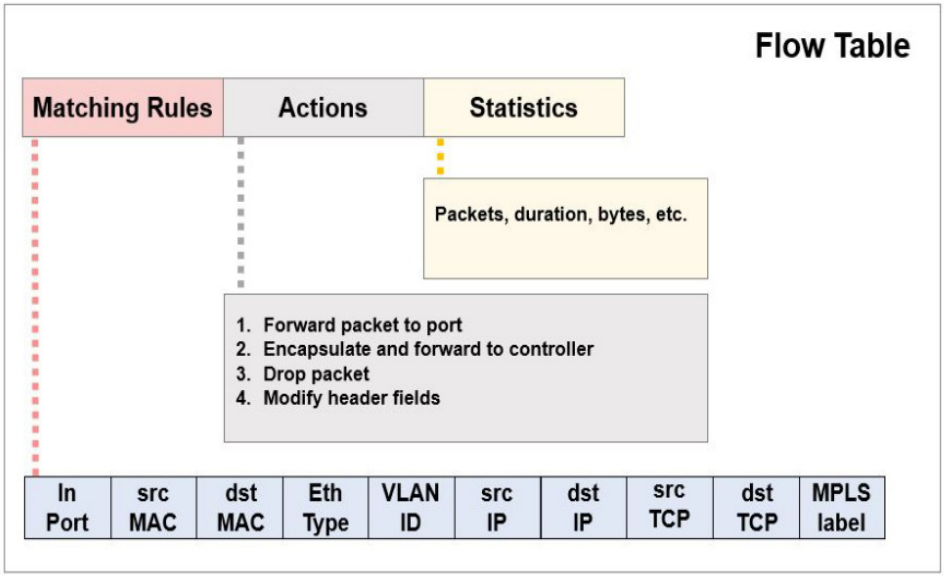

FIGURE 4.11 – Composants d'une table de *flows* dans *OpenFlow*.

parlons alors de *matching*. Ces règles concernent les adresses MAC/IP source et destination, le numéro de VLAN ou encore l'étiquette MultiProtocol Label Switching (MPLS) (à partir de la version 1.1.0 de *OpenFlow*). Dans le cas d'un *matching*, le switch applique les actions/instructions définies pour ce *matching*, telles que, l'acheminement vers un port de sortie en particulier, le filtrage du paquet, etc. En l'absence de *matching*, le paquet « inconnu » est encapsulé et transmit, via un message *Packet-in*, au contrôleur SDN qui va alors définir les actions à appliquer sur ce paquet ainsi que sur le *flow* auquel il correspond. Notons que ce processus de transmission vers le contrôleur doit être défini, à l'implémentation du réseau, comme règle par défaut [80]. Dans le cas contraire, le paquet est rejeté. Enfin, le compteur statistique est mis à jour dans la table des flows après l'application des instructions. Ces statistiques sont ensuite transmises vers le contrôleur SDN et vers les applications de monitoring, s'exécutant dans le MP

Avantages du modèle SDN

Une architecture SDN présente un grand nombre d'avantages tels que :

— grâce à la séparation des plans réseau (DP et CP), un modèle SDN apporte un suivi du fonctionnement global d'un réseau, facilitant ainsi la gestion, le contrôle, l'évolution et l'innovation de l'infrastructure réseau,
— la centralisation de la couche logique du CP simplifie la mise en place, la configuration et la modification de règles applicables sur le fonctionnement des switches/routeurs présents dans le réseau, notamment grâce à l'utilisation de langages de programmation de haut-niveau et

d'applications dédiées,
— une couche de contrôle programmable réagit de manière automatique, rapide et efficace aux changements, pannes ou erreurs produits sur la topologie du réseau,
— la centralisation du contrôle fournit une vue globale de l'état des nœuds sous-jacents, ce qui permet de développer des fonctions, des applications ainsi que des services réseau sophistiqués et plus adaptés à l'évolution des infrastructures/plateformes,
— les performances ainsi que la QoS du réseau sont significativement améliorées, à travers le développement d'une multitude d'applications d'ingénierie de trafic, d'optimisation réseau et de monitoring.

Fronthaul basé sur le modèle SDN

Au sein du FH, nous déployons un ensemble de switches SDN (virtuels et physiques) qui permettent de connecter les nœuds RRS au pool RCC. Ces switches sont gérés, via *OpenFlow*, par un contrôleur virtualisé au sein du pool RCC, offrant ainsi une vue globale du FH. Le choix du SDN comme solution de DP/CP est motivée par différentes raisons. D'abord, SDN est actuellement supporté par un grand nombre d'équipements déployés dans les infrastructures réseau existantes. Par conséquent, en exploitant ces équipements, nous pourrions réduire considérablement les CAPEX. En outre, grâce à sa programmabilité et sa flexibilité, un environnement SDN offre la possibilité d'implémenter des solutions de routage et d'ingénierie de trafic, adaptées à la charge de donnés, transitant par le FH. Enfin, SDN simplifie le contrôle et la détection de défaillances de l'infrastructure réseau, grâce à l'apport d'une vue globale du FH.

Nous adoptons le contrôleur SDN distribué, ONOS, qui fournit un ensemble d'API Java, d'applications et de couches d'abstraction, pouvant être exploitées pour le *mapping* des éléments réseau (switches, liaisons, ports, etc.), pour le contrôle du DP ainsi que pour la mise en place de politiques d'accès au réseau du FH. ONOS peut être déployé à travers plusieurs serveurs, formant alors un *cluster* qui permet d'exploiter les ressources CPU/mémoire de multiple machines physiques. Toutefois, ce *cluster* demeure logiquement centralisé et maintient une vue globale du réseau. En outre, grâce à son architecture distribuée, ONOS offre une forte disponibilité, une fiabilité, une tolérance de défaillances et peut s'adapter à la croissance du nombre de nœuds SDN.

4.4 Evaluation des performances

Dans cette partie, nous évaluons la fiabilité ainsi que les performances de notre architecture C-RAN. Pour cela, nous mesurons, sur les deux solutions de fronthaul proposées, les indicateurs suivants : bande passante digitale, débit de données, RTT et la variation de la latence (gigue).

Le nœud RRS est implémenté au sein d'une machine physique, équipée par un processeur *Intel Core i7* de 3.40 GHz et par une mémoire RAM de 8 Go. Par ailleurs, l'environnement Xen, hébergeant les différentes VM, est installé sur un noyau Linux 4.9 (avec la distribution Ubuntu 14.04). Ainsi mentionné précédemment, l'unité RAU est implémentée dans le *Dom0* et est directement connectée, à

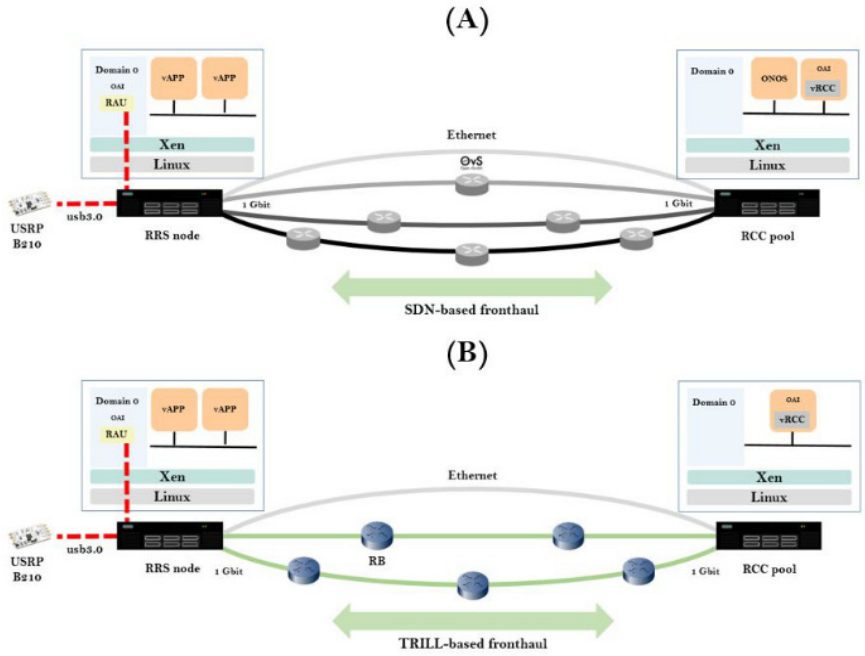

FIGURE 4.12 – Implémentations de l'architecture C-RAN avec les FH SDN et TRILL.

travers un port USB3.0, à une carte SDR de type USRP B210 (représentant la RRU). L'unité RCC ainsi que le contrôleur ONOS sont virtualisées dans des *DomU* séparés, au sein d'une machine physique, équipée par un processeur *Intel Celeron* de 1.90 GHz et par une mémoire RAM de 8 Go. Afin de réduire la complexité de notre expérimentation, nous mesurons les indicateurs de performance entre un seul nœud RRS et une seule unité RCC présente dans le pool. En outre, nous utilisons le système Single-Input Single-Output (SISO) pour la transmission/réception radio avec la méthode Frequency Division Duplexing (FDD) et une bande passante de 10 MHz. Enfin, les données IQ sont encapsulées dans des trames Ethernet compressées pour être transportées à travers le FH.

L'infrastructure du FH est composée d'un nombre variable de RB et de switches OVS physiques pour les solutions TRILL et SDN, respectivement. En effet, nous connectons le pool RCC au RRS à travers 1, 2 puis 3 OVS intermédiaires, déployés dans le FH-SDN, ainsi qu'à travers 2 puis 3 RB, déployés dans le FH-TRILL (TRILL nécessite au minimum deux RB pour fonctionner). Nous évaluons par la suite l'évolution des valeurs des indicateurs de performance, selon le nombre de switches présents dans le FH. Par ailleurs, une comparaison est réalisée entre ces différents scénarios et une connexion Ethernet directe reliant le nœud RRS au pool RCC. Il est à noter que nous utilisons sur tous nos équipements des cartes réseau *Intel* (I211, I217-LM, 82583V), possédant une capacité maximale de 1 Gbit/s. Le schéma de l'implémentation, décrite dans cette partie, est présenté dans la figure 4.12.

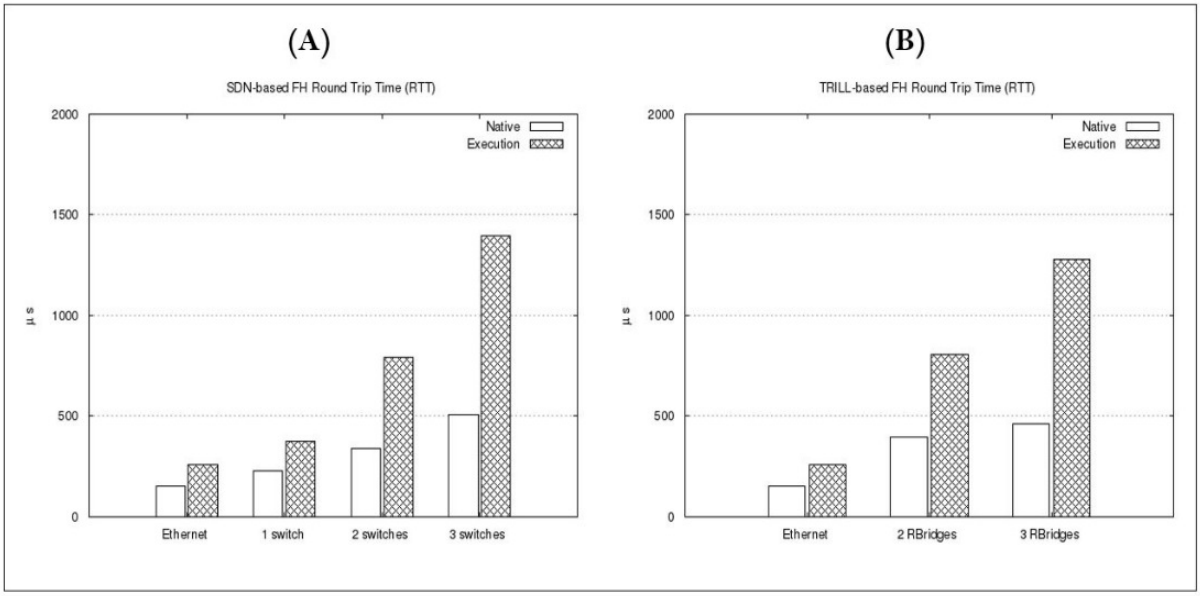

FIGURE 4.13 – Round Trip Time (RTT) moyen pour le FH-SDN et le FH-TRILL, en fonction du nombre de switches/routeurs intermédiaires.

4.4.1 Round Trip Time (RTT)

Afin de mesurer le RTT moyen des FH-SDN et FH-TRILL (figure 4.13) pour chacun des scénarios présentés dans cette partie, nous exécutons un ping à partir du nœud RRS vers le pool RCC, avant l'établissement de la connexion RAU-RCC (*Native*) (c.-à-d. avant le lancement du C-RAN) et durant cette même connexion (*Execution*). Nous notons que dans le modèle de division B de OAI, 277 Mbits/s d'échantillons de données IQ sont générés et transmis continuellement à travers le réseau FH, lorsque la connexion RAU-RCC est établie [62].

Nous observons, à partir de la figure 4.13, que le RTT moyen, dans le cas *Native*, varie entre 154 μs (Ethernet) et 506/460 μs (3 OVS/RB) pour les FH SDN/TRILL, ce qui représente des valeurs inférieures au délai de transmission (latence) de 250 μs, préconisé dans une architecture C-RAN, par l'alliance Next Generation Mobile Networks (NGMN) [83]. Néanmoins, le RTT moyen augmente considérablement lorsque la connexion RAU-RCC est établie, passant à 257 μs (+67%) pour la liaison Ethernet et à 1400/1280 μs (+177/178%) pour les réseaux SDN/TRILL. Cela peut impacter négativement le fonctionnement de certaines applications, nécessitant une très faible latence (< 1 ms), telle que les opérations robotiques [84]. Notons que TRILL fournit de meilleures performances que le SDN, lorsque le nombre de switches/RB intermédiaires est supérieur à 2.

En résumé, notre architecture C-RAN peut fournir une faible latence, à travers un réseau à moyenne capacité (1 Gbit/s), quel que soit la technologie utilisée (SDN/TRILL). Par ailleurs, le FH implémenté, durant ces expérimentations, peut satisfaire les exigences en terme de latence, imposées par la 5G (1 ms) [5], dans la majorité des scénarios présentés plus haut.

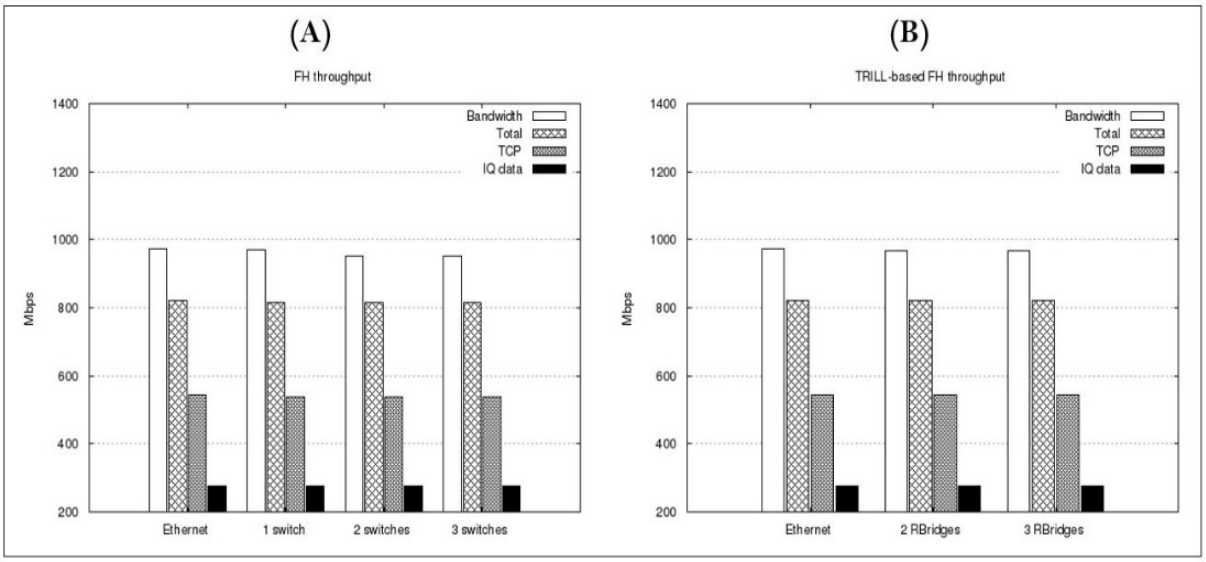

FIGURE 4.14 – Débit moyen pour le FH-SDN et le FH-TRILL, en fonction du nombre de switches/RBridges intermédiaires.

4.4.2 Débit moyen de données

Dans la figure 4.14, nous comparons, au sein des solutions FH-SDN et FH-TRILL, les valeurs moyennes de quatre indicateurs qui sont : la bande passante digitale, le débit de données, le trafic TCP ainsi que le débit des données IQ. Chacun de ces indicateurs est présenté ci-dessous :

- **Bande passante digitale (*Bandwidth*) :** constitue la vitesse de transmission maximale, pouvant être atteinte entre le nœud RRS et le pool RCC. Cette valeur est mesurée avant l'établissement de la connexion RAU-RCC. Tel que nous l'observons sur la figure 4.14, la bande passante digitale décroit légèrement de 973 Mbit/s (Ethernet) à 951/967 Mbit/s (3 OVS/RB) pour les FH SDN/TRILL, selon le nombre de nœuds intermédiaires présents dans le réseau.
- **Débit de données IQ (*IQ data*) :** représente des échantillons de données radio, compressés et encapsulés dans des trames Ethernet. Ainsi mentionné précédemment, 277 Mbit/s de données IQ sont continuellement générées (139 Mbit/s en montant (*uplink*) et 138 Mbit/s en descendant (*downlink*)) entre les unités RAU et RCC. De plus, cette valeur demeure constante et n'est pas impactée par le changement de la topologie du FH.
- **Trafic TCP (*TCP*) :** cet indicateur représente le débit maximal de paquets TCP pouvant être transportés simultanément (et avec succès) avec les données IQ au sein du FH. A travers cette expérimentation, nous souhaitons simuler le trafic de données généré par des utilisateurs ou des VM. Cette valeur est mesurée à partir d'un client *Iperf*, installé dans une VM se trouvant au niveau du nœud RRS. *Iperf* permet de générer et transmettre du trafic TCP vers le pool RCC. Nous constatons que le débit TCP moyen baisse légèrement de 543 Mbit/s (Ethernet) à 538/542 Mbit/s (3 OVS/RB).
- **Débit moyen total (*Total*) :** est calculé à partir de l'addition des débits de données IQ et TCP

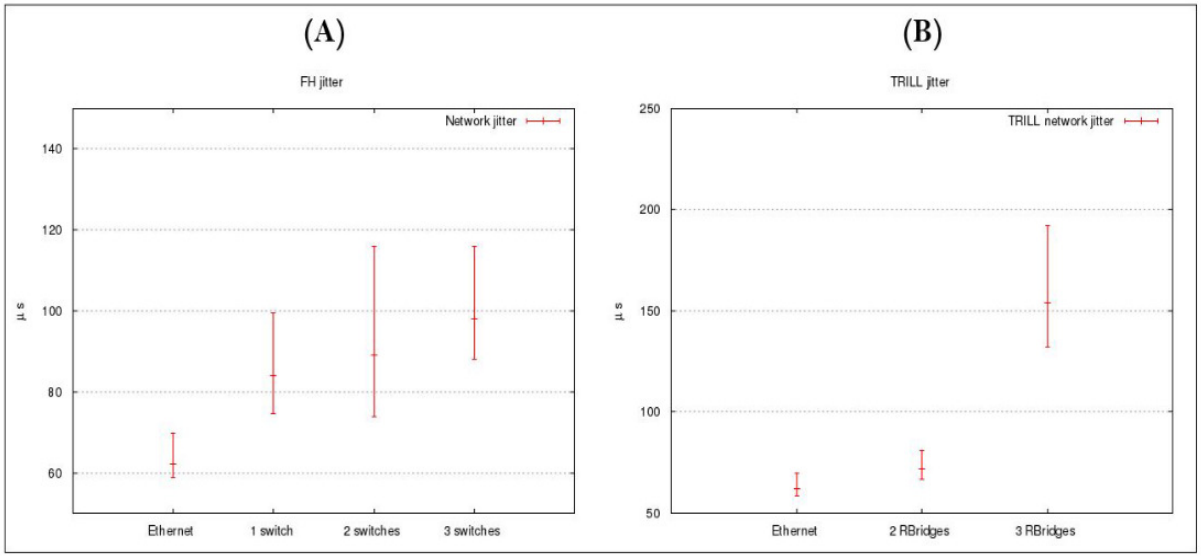

FIGURE 4.15 – Gigue moyenne pour le FH-SDN et le FH-TRILL, en fonction du nombre de switches/routeurs intermédiaires.

mesurés dans le réseau FH. Le débit total varie entre 815 et 820 Mbit/s, selon le nombre de OVS/RB intermédiaires déployés dans le FH. Cela représente 84-85% de la bande passante digitale.

Nous observons, à partir de ces mesures, que le débit moyen est relativement stable, quel que soit la topologie ou la solution réseau déployées dans le FH, bien que TRILL fournit des performances légèrement supérieures à SDN, lorsque le nombre de switches/RB est supérieur à 2. En outre, nous notons que le FH supporte le transport, en simultané, de plusieurs types de données, transmises par différentes sources.

4.4.3 Variation de latence (gigue)

Réduire la gigue dans les réseaux mobiles pour répondre aux contraintes imposées par certaines applications sensibles à la variation de délai, fait partie des objectifs de la 5G. Ainsi, il est nécessaire de mesurer la gigue de notre architecture C-RAN. Les résultats obtenus sont présentés dans la figure 4.15. Nous observons que la gigue moyenne varie entre 59 (Ethernet) et 116 μs (3 OVS) pour le FH-SDN et entre 60 (Ethernet) et 190 μs (3 RB) pour le FH-TRILL. SDN fournit une meilleure gigue que TRILL dans notre architecture C-RAN, avec une moyenne inférieure à 100 μs (et inférieure à 160 μs avec TRILL), quel que soit la topologie déployée dans le FH. Néanmoins, les deux solutions réseau proposées garantissent une faible gigue qui reste largement inférieure aux 30 ms requis par les applications vidéo et VoIP.

4.5 Conclusion

Concevoir une nouvelle génération de réseau d'accès radio, plus performante, efficiente (en terme d'énergie) et plus fiable, tout en réduisant les CAPEX/OPEX, représente un défi majeur pour les opérateurs et les manufacturiers mobiles qui sont à la recherche d'un moyen de réadapter leurs infrastructures réseau afin de répondre aux exigences imposées par la 5G. Dans cette optique, nous avons proposé une nouvelle architecture C-RAN, basée sur l'interface NGFI, qui intègre un réseau switché dans le FH. Pour cela, nous avons implémenté deux solutions réseau distinctes pour les DP/CP qui sont SDN ainsi que TRILL. En outre, nous avons décrit les avantages apportés par la virtualisation des éléments RRS et RCC au sein de la plateforme MNet, en termes de flexibilité et de partage des ressources. Enfin, nous avons montré, à travers nos expérimentations, que notre implémentation C-RAN, utilisant la solution logicielle OAI, peut supporter le transport simultané des données TCP et IQ à travers plusieurs nœuds réseau se trouvant dans le FH. Par ailleurs, nous avons observé que les solutions SDN et TRILL permettent de fournir les performances requises par les réseaux mobiles 5G.

Cependant, nous avons établi nos expérimentations sur des NIC à faible capacité (1 Gbit/s) qui sont largement déployées dans les équipements réseau actuels mais qui devraient être remplacées dans les années à venir par des cartes plus puissantes (10 Gbit/s). Par conséquent, nous souhaiterions effectuer nos expérimentations sur des NIC pouvant fournir des débits de 10-40 Gbit/s pour de meilleures performances. Par ailleurs, nous souhaiterions mettre en place, à travers ONOS, des applications de monitoring et d'ingénierie de trafic au sein du FH pour une meilleure gestion du réseau et une meilleure QoS.

Chapitre 5

Conclusion

Sommaire

Prévue pour l'horizon 2020, la nouvelle génération de réseaux cellulaires (5G) devrait relever de nombreux défis, en premier lieu, technologiques, en introduisant de nouvelles solutions (MEC, NFV, SDN, etc) pour une meilleure disponibilité, un meilleure fiabilité et une amélioration des performances, mais également économiques, avec la nécessité de réduire les CAPEX/OPEX qui n'ont cessé de croitre ces dernières années. Le C-RAN est de ce fait, défini pour devenir l'architecture d'accès radio standard pour la 5G, apportant une gestion des ressources radio optimisée, une gestion de la mobilité des utilisateurs améliorée, une efficience énergétique et un coût de déploiement réduit grâce à son architecture centralisée et à sa flexibilité permettant de réutiliser les infrastructures physiques présentes dans le réseau. Dans ce travail de thèse, nous avons conçu une architecture C-RAN partiellement centralisée, basée sur l'interface Next Generation Fronthaul Interface (NGFI). Notre but a été de proposer une implémentation flexible et « low-cost », pouvant supporter différentes technologies réseau et pouvant être implémentée sur des équipements génériques. Pour ce faire, nous avons intégré les composants : Remote Radio System (RRS) et Radio Cloud Center (RCC), développés sur l'environnement Open Air Interface (OAI), au sein de la plateforme de virtualisation Metamorphic Network (MNet), plateforme qui embarque un environnement Xen et qui est implémentée sur un ensemble de nœuds réseau interconnectés, appelés MNetBox.

La première contribution de cette étude consistait à concevoir et à implémenter une nouvelle architecture Xen pour MNet qui intègre le framework d'accélération du traitement de paquets OpenData-Plane (ODP), au sein d'un domaine virtuel privilégié appelé *Domain Driver (DD)*, auquel nous avons

instancié un certain nombre de cœurs de processeur virtuels (vCPU). Cette solution permet notamment d'exploiter les fonctionnalités d'ODP pour un traitement accéléré du trafic de données transitant par la plateforme, sans occasionner de surcharge sur les ressources physiques du processeur. A travers les expérimentations réalisées, nous avons observé que cette nouvelle architecture Xen apporte un gain de performance (nombre de paquets traités par seconde, débit, etc.) de 15%, lorsque le nombre de cœurs vCPU était supérieur à 1, comparée à la précédente architecture de MNet. Cette architecture nous a servi de base pour l'implémentation du C-RAN.

Dans la seconde contribution, nous avions eu pour objectif d'implémenter l'architecture C-RAN-NGFI au sein de la plateforme MNet. Pour cela, nous avons virtualisé sur le domaine principal de Xen, *Domain 0 (Dom0)*, les fonctions de la Radio Aggragation Unit (RAU), à laquelle nous avons connecté une antenne radio (Remote Radio Unit (RRU)). Les unités RAU et RRU ont été implémentées au sein d'une même machine physique, appelée « RRS node » qui constitue alors la station de base de notre architecture C-RAN. Les unités de traitement RCC ont été virtualisées (Xen) sur un serveur distant nommé pool RCC, dans lequel les ressources physiques (mémoire, CPU, interfaces réseau, etc.) ont été partagées entre ces unités et d'autres applications virtualisées. Afin de connecter les nœuds RRS au pool RCC, nous avons déployé un réseau Fronthaul (FH) composé d'un ensemble de switches (nœuds) interconnectés. Nous avons également implémenté deux solutions réseau, chacune possédant un modèle de plan de contrôle différent, le modèle distribué avec le protocole de couche 2 Transparent Interconnection of Lots of Links (TRILL) et le modèle centralisé avec l'architecture Software Defined Network (SDN). Une comparaison des performances réseau a été réalisée entre TRILL et SDN, selon des mesures de Round Trip Time (RTT), de débit de données moyen et de gigue et selon la variation du nombre de switches intermédiaires se trouvant entre un nœud RRS et un pool RCC. Les résultats obtenus ont montré que les deux solutions réseau garantissent les performances requises par la 5G dans la plupart des scénarios traités, notamment en termes de latence et de gigue. De plus, notre réseau FH permet de transporter en simultané des données radio (IQ) ainsi que du trafic d'utilisateur (TCP) sur plusieurs sauts (plusieurs switches) en utilisant jusqu'à 85% de la bande passante digitale (débit de données maximal dans le réseau). Enfin, nous avons observé que TRILL offrait de meilleures performances de débit et de RTT lorsque le nombre de switches intermédiaires était supérieur à 2.

A travers notre architecture C-RAN, nous avons souhaité atteindre les objectifs introduits par NGFI qui sont : 1) l'atténuation des contraintes de latence, de synchronisation et de bande passante, habituellement exigées pour le transport des données radio à travers le FH, notamment dans une centralisation complète ; 2) l'adoption d'Ethernet comme médium de transport radio entre les stations de base et le pool RCC, pour réduire les CAPEX/OPEX ; 3) l'exploitation des avantages de la virtualisation pour concevoir un réseau flexible et performant qui garantit un meilleur partage des ressources de calcul (côté pool RCC) ainsi qu'un meilleur déploiement des applications/services à l'extrémité du réseau (côté nœud RRS) et 4) le déploiement d'un réseau FH switché et ouvert à différents opérateurs, fournisseurs de services ou encore tierces parties, pouvant transporter, sur plusieurs sauts, tout type de trafic de données. A l'opposé d'une architecture C-RAN complètement centralisée, dans laquelle une RRU est connectée au pool BBU à travers une liaison point-à-point dédiée.

5.1 Travaux futurs

La nouvelle génération de réseaux mobiles n'a pas encore été standardisée, malgré les nombreux travaux menés pour définir le spectre radiofréquence, les techniques de multiplexage, les technologies réseau, etc., qui seront adoptés dans le C-RAN et plus généralement au sein de la 5G. Par conséquent, un certain nombre de pistes restent encore à explorer avant son lancement officiel en 2020. Dans cette section, nous allons présenter les travaux qui pourraient être réalisés dans les trois prochaines années pour améliorer notre architecture C-RAN.

Dans ce travail de thèse, nous avons effectué nos expérimentations sur des interfaces réseau possédant des capacités limitées à 1 Gbit/s. Bien que ces dernières soient largement déployées dans les équipements réseau actuels, elles devraient être remplacées par des cartes Ethernet plus puissantes, pouvant assurer des débits de 10 à 40 Gbit/s. Pour ce fait, nous projetons d'augmenter la capacité du réseau FH, afin de fournir des performances de débit qui correspondent au mieux aux besoins de la 5G.

Par ailleurs, il est nécessaire de mettre en place un réseau FH robuste qui permettrait de fournir de meilleures performances, une sécurité et une qualité de service (QoS) requises pour le C-RAN. Pour cela, il est envisageable d'intégrer, au sein du contrôleur SDN (plan de management), des applications dédiées à la gestion du trafic de données transitant par le FH en terme d'ingénierie de trafic avec la mise en place de load balancers programmables et des mécanismes de multi-pathing, en terme de monitoring et de contrôle avec l'utilisation d'outils statistiques permettant de mesurer différents facteurs réseau (performance, type de trafic, etc.) et enfin en terme de sécurité avec l'ajout d'applications de firewalling, de contrôle d'accès et de détection d'intrusions. Par ailleurs, il serait intéressant d'observer l'impact positif ou négatif de ces différentes fonctionnalités sur le transport des données radio entre les stations de base et le pool RCC.

La virtualisation des éléments du C-RAN dans un environnement Xen représentait un défi majeur de cette étude que nous avons réussi à relever. Néanmoins, intégrer l'unité RAU dans le *Dom0* présente certains inconvénients comme l'impossibilité de migrer cette unité vers d'autres machines physiques. De ce fait, nous souhaiterions virtualiser la RAU au sein du domaine secondaire *Domain User (DomU)*, à l'instar de l'unité RCC. Cela offrira plus de flexibilité à notre architecture C-RAN et nous permettra de déployer les unités RAU comme un service ou une fonction réseau.

Enfin, nous souhaiterions associer, au sein de la plateforme MNet, les deux outils d'accélération de traitement de paquets ODP et *Intel DPDK* pour fournir des performances réseau supérieures et une variété de librairies permettant de programmer de applications de packet processing personnalisées.

5.2 Publications

Nous présentons dans cette section les différents documents rédigés durant cette thèse.

5.2.1 Conférences

Publications au sein de conférences internationales :

— RABIA, Tarek, BRAHAM, Othmen, et PUJOLLE, Guy. **Accelerating packet processing in a Xen environment With OpenDataPlane**. In : Advanced Information Networking and Applications (AINA), 2016 IEEE 30th International Conference on. IEEE, 2016. p. 408-413.
— RABIA, Tarek, BRAHAM, Othmen, et PUJOLLE, Guy. Partially centralized C-RAN architecture using TRILL protocol. In : Network of the Future (NOF), 2016 7th International Conference on the. IEEE, 2016. p. 1-3.

Article accepté qui sera publié au courant de l'année :

— RABIA, Tarek, BRAHAM, Othmen. **A New SDN-based Next Generation Fronthaul Interface for a Partially Centralized C-RAN**. n : Advanced Information Networking and Applications (AINA), 2016 IEEE 32th International Conference on. IEEE, 2018.

5.2.2 Rapports techniques

Délivrables rédigés pour le compte du projet Elastic Network :

— Tarek RABIA, Othmen BRAHAM, **"Techniques de virtualisation des HeNodeB, RRH et BBU"**, Délivrable 2.2, FUI Elastic, Novembre 2016.
— Tarek RABIA, Othmen BRAHAM, **"Rapport d'installation"**, Délivrable 4.1, FUI Elastic, Novembre 2016.
— Tarek RABIA, Othmen BRAHAM, **"Implémentation et intégration du C-RAN"**, Délivrable 3.4, FUI Elastic, Octobre 2017.

5.2.3 Livres blancs

— Tarek RABIA, **"Metamorphic Networks (MNET) : La nouvelle solution d'accès Cloud"**, Avril 2015.

Bibliographie

[1] STATS, I. W. Internet world stats : Usage and population statistics. Retrieved from Internet World Stats : http ://www.internetworldstats.com/stats. htm, 2017.

[2] INTELLIGENCE, G. S. M. A. Global Mobile Economy Report 2017. 2017.

[3] INDEX, Cisco Visual Networking. Global Mobile Data Traffic Forecast Update, 2015–2021 White Paper. link : http ://goo. gl/ylTuVx, 2017.

[4] GHOSH, Amitava, RATASUK, Rapeepat, MONDAL, Bishwarup, et al. LTE-advanced : next-generation wireless broadband technology. IEEE wireless communications, 2010, vol. 17, no 3.

[5] ANDREWS, Jeffrey G., BUZZI, Stefano, CHOI, Wan, et al. What will 5G be ?. IEEE Journal on selected areas in communications, 2014, vol. 32, no 6, p. 1065-1082.

[6] CHIOSI, Margaret, CLARKE, Don, WILLIS, P., et al. Network functions virtualisation introductory white paper. In : SDN and OpenFlow World Congress. 2012.

[7] CHIH-LIN, I., ROWELL, Corbett, HAN, Shuangfeng, et al. Toward green and soft : a 5G perspective. IEEE Communications Magazine, 2014, vol. 52, no 2, p. 66-73.

[8] BRAHAM, Othmen et PUJOLLE, Guy. The metamorphosing network (M-Net). In : Global Information Infrastructure and Networking Symposium (GIIS), 2012. IEEE, 2012. p. 1-4.

[9] BARHAM, Paul, DRAGOVIC, Boris, FRASER, Keir, et al. Xen and the art of virtualization. ACM SIGOPS Operating Systems Review, 2003, vol. 37, no 5, p. 164-177.

[10] OpenDataPlane. URL http ://www.opendataplane.org.

[11] China Mobile, Next generation fronthaul interface. White Paper, Oct, 2015.

[12] PERLMAN, Radia. Rbridges : transparent routing. In : INFOCOM 2004. Twenty-third Annual-Joint Conference of the IEEE Computer and Communications Societies. IEEE, 2004. p. 1211-1218.

[13] com4innov, http ://www.com4innov.com/.

[14] NIKAEIN, Navid, MARINA, Mahesh K., et al. OpenAirInterface : A flexible platform for 5G research. ACM SIGCOMM Computer Communication Review, 2014, vol. 44, no 5, p. 33-38.

[15] INTEL, D. P. D. K. Data Plane Development Kit. URL http ://dpdk. org.

[16] RIZZO, Luigi. netmap : A Novel Framework for Fast Packet I/O. In : USENIX Annual Technical Conference. 2012. p. 101-112.

[17] KRAWCZYK, Hugo, CANETTI, Ran, et BELLARE, Mihir. HMAC : Keyed-hashing for message authentication. 1997.

[18] BELLARD, Fabrice. QEMU, a fast and portable dynamic translator. In : USENIX Annual Technical Conference, FREENIX Track. 2005. p. 41-46.

[19] Intel DPDK, Poll Mode Driver, Data Plane Development Kit : Programmer's guide, http ://dpdk.org/doc/guides/prog_guide/poll_mode_drv.html.

[20] "PF_RING ZC API," http ://www.ntop.org/pfring_api/pfring__zc_8h.html.

[21] HAN, Sangjin, JANG, Keon, PARK, KyoungSoo, et al. PacketShader : a GPU-accelerated software router. In : ACM SIGCOMM Computer Communication Review. ACM, 2010. p. 195-206. MLA

[22] Solarflare. Openonload. http ://www.openonload.org/.

[23] BONELLI, Nicola, DI PIETRO, Andrea, GIORDANO, Stefano, et al. On multi–gigabit packet capturing with multi–core commodity hardware. In : International Conference on Passive and Active Network Measurement. Springer, Berlin, Heidelberg, 2012. p. 64-73.

[24] PAOLINO, Michele, NIKOLAEV, Nikolay, FANGUEDE, Jeremy, et al. Snabbswitch user space virtual switch benchmark and performance optimization for nfv. In : Network Function Virtualization and Software Defined Network (NFV-SDN), 2015 IEEE Conference on. IEEE, 2015. p. 86-92.

[25] EMMERICH, Paul, WOHLFART, Florian, RAUMER, Daniel, et al. Comparison of frameworks for high-performance packet IO. In : Architectures for Networking and Communications Systems (ANCS), 2015 ACM/IEEE Symposium on. IEEE, 2015. p. 29-38.

[26] BARBETTE, Tom, SOLDANI, Cyril, et MATHY, Laurent. Fast Userspace Packet Processing. In : Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for networking and communications systems. IEEE Computer Society, 2015. p. 5-16.

[27] MAHALINGAM, Mallik, DUTT, Dinesh, DUDA, Kenneth, et al. Virtual extensible local area network (VXLAN) : A framework for overlaying virtualized layer 2 networks over layer 3 networks. 2014.

[28] GROSS, Jesse, SRIDHAR, T., GARG, P., et al. Geneve : Generic network virtualization encapsulation. Internet Engineering Task Force, Internet Draft, 2014.

[29] ORAN, David. OSI IS-IS intra-domain routing protocol. 1990.

[30] KAPADIA, Amar, CHASE, Nicholas, 2017, Understanding OPNFV : Accelerate NFV Transformation using OPNFV. Sunnyvale : Mirantis.

[31] FRASER, Keir, HAND, Steven, NEUGEBAUER, Rolf, et al. Safe hardware access with the Xen virtual machine monitor. In : 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OASIS). 2004. p. 1-1.

[32] MENON, Aravind, COX, Alan L., et ZWAENEPOEL, Willy. Optimizing network virtualization in Xen. In : USENIX Annual Technical Conference. 2006.

[33] MARTINS, Joao, AHMED, Mohamed, RAICIU, Costin, et al. ClickOS and the art of network function virtualization. In : Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation. USENIX Association, 2014. p. 459-473.

[34] CERRATO, Ivano, ANNARUMMA, Mauro, et RISSO, Fulvio. Supporting fine-grained network functions through Intel DPDK. In : Software Defined Networks (EW), 2014 Third European Workshop on. IEEE, 2014. p. 1-6.

[35] HWANG, Jinho, RAMAKRISHNAN, K. K., et WOOD, Timothy. NetVM : high performance and flexible networking using virtualization on commodity platforms. Network and Service Management, IEEE Transactions on, 2015, vol. 12, no 1, p. 34-47.

[36] DONG, Yaozu, YANG, Xiaowei, LI, Jianhui, et al. High performance network virtualization with SR-IOV. Journal of Parallel and Distributed Computing, 2012, vol. 72, no 11, p. 1471-1480.

[37] KOURTIS, Michail-Alexandros, XILOURIS, Georgios, RICCOBENE, Vincenzo, et al. Enhancing VNF performance by exploiting SR-IOV and DPDK packet processing acceleration. In : Network Function Virtualization and Software Defined Network (NFV-SDN), 2015 IEEE Conference on. IEEE, 2015. p. 74-78.

[38] HAN, Sangjin, JANG, Keon, PANDA, Aurojit, et al. SoftNIC : A Software NIC to Augment Hardware. UCB Technical Report No. UCB/EECS-2015, 2015.

[39] RIZZO, Luigi et LETTIERI, Giuseppe. Vale, a switched ethernet for virtual machines. In : Proceedings of the 8th international conference on Emerging networking experiments and technologies. ACM, 2012. p. 61-72.

[40] GARZARELLA, Stefano, LETTIERI, Giuseppe, et RIZZO, Luigi. Virtual device passthrough for high speed VM networking. In : Architectures for Networking and Communications Systems (ANCS), 2015 ACM/IEEE Symposium on. IEEE, 2015. p. 99-110.

[41] LIN, Yonghua, SHAO, Ling, ZHU, Zhenbo, et al. Wireless network cloud : Architecture and system requirements. IBM Journal of Research and Development, 2010, vol. 54, no 1, p. 4 : 1-4 : 12.

[42] Ericsson AB, Huawei Technologies Co. Ltd., NEC Corporation, Nokia Networks, Alcatel-Lucent, CPRI Specification V6.1(2014-07-01) : Common Public Radio Interface (CPRI) ; Interface Specification, <http ://www.cpri.info>, 2014.

[43] OPEN BASE STATION ARCHITECTURE INITIATIVE, et al. Reference Point 1 Test Messages. Version, 2005, vol. 1, p. 21.

[44] ETSI, ETSI ORI (Open Radio Interface), [Online]. Available : <https ://portal.etsi.org/tb.aspx ?tbid=738&SubTb=738>.

[45] CHECKO, Aleksandra, CHRISTIANSEN, Henrik L., YAN, Ying, et al. Cloud RAN for mobile networks—A technology overview. IEEE Communications surveys & tutorials, 2015, vol. 17, no 1, p. 405-426.

[46] GOMES, Nathan J., CHANCLOU, Philippe, TURNBULL, Peter, et al. Fronthaul evolution : from CPRI to Ethernet. Optical Fiber Technology, 2015, vol. 26, p. 50-58.

[47] LI, Jian, CHEN, Dageng, WANG, Yi, et al. Performance evaluation of cloud-ran system with carrier frequency offset. In : Globecom Workshops (GC Wkshps), 2012 IEEE. IEEE, 2012. p. 222-226.

[48] CHECKO, Aleksandra. Cloud Radio Access Network architecture. Towards 5G mobile networks. 2016. PHD thesis. Technical University of Denmark.

[49] GHEBRETENSAÉ, Zere, LARAQUI, Kim, DAHLFORT, Stefan, et al. Transmission solutions and architectures for heterogeneous networks built as C-RANs. In : Communications and Networking in China (CHINACOM), 2012 7th International ICST Conference on. IEEE, 2012. p. 748-752.

[50] SEGEL, J. et WELDON, M. Lightradio portfolio-technical overview. Technology White Paper, 2011, vol. 1.

[51] HANSRYD, Jonas et EDSTAM, Jonas. Microwave capacity evolution. Ericsson review, 2011, vol. 1, p. 22-27.

[52] LAW, David, DOVE, Dan, D'AMBROSIA, John, et al. Evolution of Ethernet standards in the IEEE 802.3 working group. IEEE Communications Magazine, 2013, vol. 51, no 8, p. 88-96.

[53] FERRANT, J.-L., GILSON, Mike, JOBERT, Sebastien, et al. Synchronous Ethernet : A method to transport synchronization. IEEE Communications Magazine, 2008, vol. 46, no 9.

[54] EIDSON, John et LEE, Kang. IEEE 1588 standard for a precision clock synchronization protocol for networked measurement and control systems. In : Sensors for Industry Conference, 2002. 2nd ISA/IEEE. IEEE, 2002. p. 98-105.

[55] MILLS, David L. Network time protocol (NTP). Network, 1985.

[56] eCPRI. eCPRI Specification V1.0. Common Public Radio Interface : eCPRI Interface Specification, 2017.

[57] MARSCH, Patrick et FETTWEIS, Gerhard P. (ed.). Coordinated Multi-Point in Mobile Communications : from theory to practice. Cambridge University Press, 2011.

[58] MCKEOWN, Nick, ANDERSON, Tom, BALAKRISHNAN, Hari, et al. OpenFlow : enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 2008, vol. 38, no 2, p. 69-74.

[59] DÖTSCH, Uwe, DOLL, Mark, MAYER, Hans-Peter, et al. Quantitative analysis of split base station processing and determination of advantageous architectures for LTE. Bell Labs Technical Journal, 2013, vol. 18, no 1, p. 105-128.

[60] CHIH-LIN, I., YUAN, Yannan, HUANG, Jinri, et al. Rethink fronthaul for soft RAN. IEEE Communications Magazine, 2015, vol. 53, no 9, p. 82-88.

[61] HUANG, Bo-Syuan, CHIANG, Yi-Han, et LIAO, Wanjiun. Remote radio head (RRH) deployment in flexible C-RAN under limited fronthaul capacity. In : Communications (ICC), 2017 IEEE International Conference on. IEEE, 2017. p. 1-6.

[62] CHANG, Chia-Yu, NIKAEIN, Navid, KNOPP, Raymond, et al. FlexCRAN : A Flexible Functional Split Framework over Ethernet Fronthaul in Cloud-RAN. In : Communications (ICC), 2017 IEEE International Conference on. IEEE, 2017. p. 1-7.

[63] DEIß, Thomas, COMINARDI, Luca, GARCIA-SAAVEDRA, Andres, et al. Packet forwarding for heterogeneous technologies for integrated fronthaul/backhaul. In : Networks and Communications (EuCNC), 2016 European Conference on. IEEE, 2016. p. 133-137.

[64] DE LA OLIVA, Antonio, PÉREZ, Xavier Costa, et al. Xhaul : toward an integrated fronthaul/backhaul architecture in 5G networks. IEEE Wireless Communications, 2015, vol. 22, no 5, p. 32-40.

[65] C. Y. Chang, R. Schiavi, N. Nikaein, T. Spyropoulos and C. Bonnet, "Impact of packetization and functional split on C-RAN fronthaul performance," 2016 IEEE International Conference on Communications (ICC), Kuala Lumpur, 2016, pp. 1-7.

[66] HU, Yun Chao, PATEL, Milan, SABELLA, Dario, et al. Mobile edge computing—A key technology towards 5G. ETSI White Paper, 2015, vol. 11, no 11, p. 1-16.

[67] PERLMAN, Radia, GHANWANI, Anoop, EASTLAKE 3RD, Donald, et al. Routing bridges (RBridges) : Base protocol specification. 2011.

[68] PERLMAN, Radia. Challenges and Opportunities in the Design of TRILL : a Routed layer 2 Technology. In : GLOBECOM Workshops, 2009 IEEE. IEEE, 2009. p. 1-6.

[69] HOPPS, Christian E. Analysis of an equal-cost multi-path algorithm. 2000.

[70] PFAFF, Ben et DAVIE, Bruce. The open vSwitch database management protocol. 2013.

[71] DORIA, Avri, SALIM, J. Hadi, HAAS, Robert, et al. Forwarding and control element separation (ForCES) protocol specification. 2010.

[72] SONG, Haoyu. Protocol-oblivious forwarding : Unleash the power of SDN through a future-proof forwarding plane. In : Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. ACM, 2013. p. 127-132.

[73] NG, Eugene, CAI, Z., et COX, A. L. Maestro : A system for scalable openflow control. Rice University, Houston, TX, USA, TSEN Maestro-Techn. Rep, TR10-08, 2010.

[74] ERICKSON, David. The beacon openflow controller. In : Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. ACM, 2013. p. 13-18.

[75] Project Floodlight, “Floodlight,” 2012.[Online]. Available : http ://floodlight.openflowhub.org/.

[76] BERDE, Pankaj, GEROLA, Matteo, et al. ONOS : towards an open, distributed SDN OS. In : Proceedings of the third workshop on Hot topics in software defined networking. ACM, 2014. p. 1-6.

[77] KOPONEN, Teemu, CASADO, Martin, GUDE, Natasha, et al. Onix : A distributed control platform for large-scale production networks. In : OSDI. 2010. p. 1-6.

[78] FIELDING, Roy. Representational state transfer. Architectural Styles and the Design of Netowork-based Software Architecture, 2000, p. 76-85.

[79] YAP, Kok-Kiong, HUANG, Te-Yuan, DODSON, Ben, et al. Towards software-friendly networks. In : Proceedings of the first ACM asia-pacific workshop on Workshop on systems. ACM, 2010. p. 49-54.

[80] KREUTZ, Diego, RAMOS, Fernando MV, VERISSIMO, Paulo Esteves, et al. Software-defined networking : A comprehensive survey. Proceedings of the IEEE, 2015, vol. 103, no 1, p. 14-76.

[81] Open Networking Foundation (ONF), 2014. [Online]. Available : https ://www.opennetworking.org/

[82] LARA, Adrian, KOLASANI, Anisha, et RAMAMURTHY, Byrav. Network innovation using openflow : A survey. IEEE communications surveys & tutorials, 2014, vol. 16, no 1, p. 493-512.

[83] ALLIANCE, N. G. M. N. Further study on critical C-RAN technologies. Next Generation Mobile Networks, 2015.

[84] HAERICK, W. et GUPTA, M. White Paper : 5G and the Factories of the Future. 5G-PPP, Tech. Rep, 2015.