Application Driven Networking in Dynamic Environments. Etat de l'art : SDN, virtualisation réseau et DDS

Slim Abdellatif, Lionel Bertaux, Pascal Berthou, Samir Medjiah, Akram Hakiri, Thierry Villemur, Thierry Gayraud, Michel Diaz, Thierry Monteil

Projet ANR- 13-ASTR-0024
Application Driven Networking in Dynamic Environments

Livrable 1.1

Etat de l'art : SDN, virtualisation réseau et DDS

Référence document	ANR-13-ASTR-0024-L1.1
Liste des auteurs	S. Abdellatif, L.Bertaux, P. Berthou, S. Medjiah, A. Hakiri, T. Villemur, T. Gayraud, M.Diaz, T.Monteil
Version	1.0
Date de livraison	1/10/2014

Résumé

Ce document a pour objectif de dresser un état de l'art des paradigmes réseaux émergents que nous envisageons de considérer dans le contexte du projet ADN, à savoir la virtualisation réseau, les réseaux programmables ou « Software-Defined Networks » (SDN), la virtualisation des fonctions réseau ou « Network Function Virtualization » (NFV) ainsi que le réseau dans le nuage ou « Cloud Networking ». Dans chacune des parties sont abordés : les principes généraux, un état des lieux des principales technologies ainsi que des cas d'utilisation représentatifs. Ce document introduit également le standard DDS (Data Distribution Service) de l'OMG (Object Management Group), middleware adopté dans le cadre de ce projet.

Mots-clés

Software Defined Networking (SDN), virtualisation réseau, Network Functions Virtualization (NFV), cloud networking, Data Distribution Service (DDS)

TABLE DES MATIERES

LISTE DES FIGURES

LISTE DES TABLEAUX

TERMINOLOGIE ET ABREVIATIONS

Terme	Description
*-aaS	* as a Service
ADSL	Asymmetric Digital Subscriber Line
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
BGP	Border Gateway Protocol
BoD	Bandwidth on Demand
BRG	Bridged Residual Gateway
CAPEX	CAPital EXpenditure
CAT-iq	Cordless Advanced Technology—internet and quality
CCN	Content Centric Networking
CE	Control Element
DHCP	Dynamic Host Configuration Protocol
DLNA	Digital Living Network Alliance
DNS	Domain Name System
DPI	Deep Packet Inspection
FE	Forwarding Element
ForCES	Forwarding and Control Element Separation Protocol
FXS	Foreign eXchange Subscriber
GRE	Generic Routing Encapsulation
IaaS	Infrastructure as a Service
IDS	Intrusion Detection System
IP	Internet Protocol
ISP / FAI	Internet Service Provider / Fournisseur d'Accès à Internet
LAN	Local Area Network
LB	Load Balancer
LFB	Logical Function Block
LSP	Label Switched Path
MPLS	Multi-Protocol Label Switching
NaaS	Network as a Service
NAPT	Network Address and Port Translation

NCS	Network Control System
NETCONF	Network Configuration Protocol
NFV	Network Functions Virtualization
NREN	National Research and Education Network
NVGRE	Network Virtualization using Generic Routing Encapsulation
OF-Config	OpenFlow Management and Configuration Protocol
OPEX	OPerational EXpenditure
OTT	Over the Top
PaaS	Platform as a Service
PIP	Physical Infrastructure Provider
PoP	Point of Presence
QoE / QdE	Quality of Experience / Qualité d'Expérience
QoS / QdS	Quality of Service / Qualité de Service
RG	Residential Gateway
RO	Router Overlay
SaaS	Software as a Service
SDN	Software-Defined Network / Réseau Programmable
SNMP	Simple Network Management Protocol
SP	Service Provider
SR-IOV	Single Root Input Output Virtualization
STT	Stateless Transport Tunneling
TaaS	Testbed as a Service
TTM	Time to Market
VLAN	Virtual Local Area Network
VM	Virtual Machine
VMDQ	Virtual Machine Device Queue
VNO	Virtual Network Operator
VNP	Virtual Network Provider
VPN	Virtual private Network
VRF	Virtual Routing and Forwarding
VxLAN	Virtual Extensible LAN
WAN	Wide Area Network

A INTRODUCTION

Après la convergence des réseaux qui a marqué les décennies précédentes, de nouvelles directions d'évolution des réseaux émergent. L'ouverture des réseaux, la programmation des réseaux et la virtualisation des réseaux et de leurs fonctions ainsi qu'un modèle de mise en place et d'utilisation/consommation du réseau emprunté du "*Cloud Computing*" (i.e. « *cloud networking* ») sont les principes de conception clefs des réseaux émergents et futurs pour leur permettre d'adresser les défis qui leur sont posés. Ces défis sont variés et découlent des nouveaux besoins des opérateurs et de leurs usagers.
Pour les premiers, ils se traduisent par la nécessité d'avoir une gestion/exploitation simplifiée du réseau capable notamment de résister aux problèmes d'échelle que posent les réseaux de centres de données et que va poser l'explosion des communications Machine-to-Machine (M2M) et de l'Internet des Objets (IoT). Cela passe par la réduction de l'intervention humaine dans les phases d'exploitation et de gestion du réseau et se traduit, d'une part, par l'automatisation des procédures complexes ou processus critiques et, d'autre part, par davantage d'autonomie laissée au réseau avec le développement de capacités d'auto-adaptation pour des fins d'optimisation ou de résilience (encore plus d'intelligence déléguée au réseau). Faisant face à une baisse de revenu latente au profit des fournisseurs de services et de contenus, les opérateurs réseau cherchent, d'une part, à améliorer de manière constante l'utilisation de leurs ressources réseau tout en offrant aux usagers une qualité d'expérience élevée et, d'autre part, à diversifier leurs services par le développement de nouveaux services innovants. La mise en place simplifiée de ces nouveaux services avec des délais de mise sur marché réduits et l'optimisation de l'utilisation du réseau avec une qualité d'expérience usager irréprochable sont d'autres défis posés aux réseaux émergents.
Du point de vue des usagers, la principale exigence s'exprime par la mise à disposition de services flexibles et personnalisés qui colle au mieux aux besoins dynamiques des applications tout en réduisant les coûts. Cela implique que le fonctionnement du réseau (son contrôle) doive, dans une certaine mesure, être guidé par les applications.

Ce projet cherche à adresser ce dernier problème en essayant de tirer partie des concepts cités ci-avant en les couplant à un middleware de type DDS (Data Distribution Service) afin d'offrir des services applicatifs personnalisés.

A cette fin et dans le but d'établir un ensemble de prérequis au projet, ce premier livrable cherche à introduire les concepts de virtualisation réseau, virtualisation des fonctions réseau, SDN (Software Defined Network) comme principale approche pour la programmation des réseaux et le cloud networking. Il décrit également des cas d'utilisation qui illustrent les possibilités et apports qu'ils amènent. Une description du middleware DDS et de ses possibilités du point de vue de la qualité de service y est également incluse.

B VIRTUALISATION RESEAU

B.1 PRINCIPES GENERAUX

La virtualisation réseau [MCB10] fait référence à un ensemble de techniques et de technologies dont la finalité est de permettre de faire coexister sur une même infrastructure

réseau (typiquement, physique) plusieurs ***réseaux logiques isolés*** et ***indépendants***. Nous détaillons dans ce qui suit ces trois derniers termes. La Figure B.1.1 illustre l'exemple d'une infrastructure réseau constituée d'éléments physiques - 4 équipements réseau et quatre nœuds terminaux reliés via des liens physiques – qui supporte deux réseaux logiques (réseau logique 1 représenté en vert et le réseau logique 2 représenté en orange). Un réseau logique est un réseau dont certains éléments (nœuds intermédiaires ou terminaux et liens) sont logiques (virtuels est le terme le plus souvent utilisé). Un nœud virtuel est une abstraction représentant un nœud de communication (terminal ou intermédiaire) qui n'a pas d'existence physique mais qui est porté par un nœud physique (ou plus rarement par un ensemble de nœuds physiques reliés via des liens physiques : agrégation) via les ressources qu'il met à sa disposition. Les ressources consommées par ce nœud virtuel (dites ressources virtuelles) sont nécessaires pour lui permettre d'exécuter ses fonctions réseau (acheminement, routage, etc.). Elles correspondent potentiellement à des ressources de commutation, CPU, mémoire, matérielle des interfaces réseau et éventuellement de stockage. A titre d'exemple, l'équipement physique Ep_1 de la Figure B.1.1 héberge deux nœuds virtuels Nv_{11} et Nv_{21} appartenant respectivement aux réseaux virtuels 1 et 2.
Similairement, un lien virtuel est une abstraction représentant un lien de communication qui peut être porté par un seul lien physique ou plus souvent par un voire plusieurs chemins physiques (liens + équipements réseau traversés). Le lien Lv11 de la Figure B.1.1 est par exemple porté par les liens physiques Lp_1 et Lp_2 ainsi que Ep_1. Des ressources virtuelles (de transmission et de commutation) sont donc allouées au lien virtuel le long des liens physiques et équipements qu'il traverse.

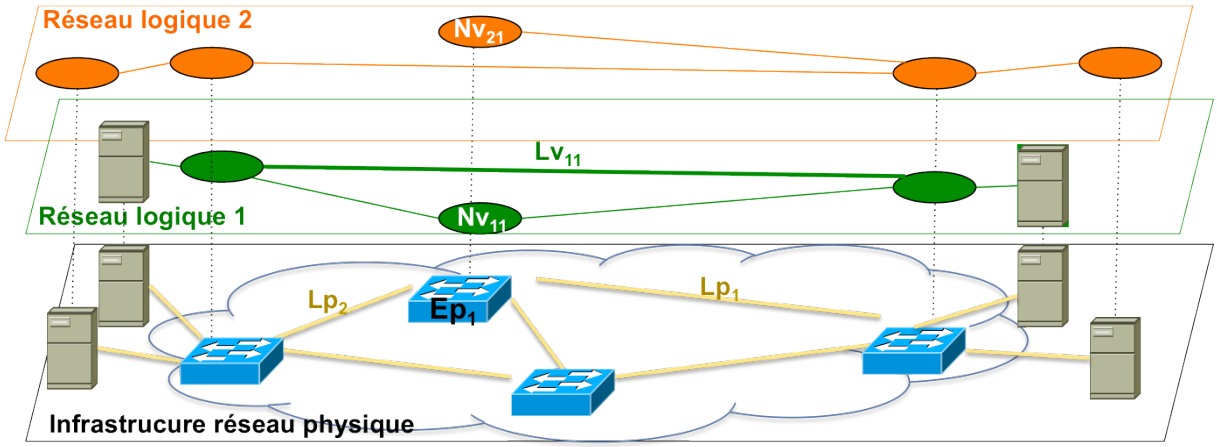


Figure B.1.1 Illustration des principes de la virtualisation réseau

L'idée clef sous-jacente à la virtualisation réseau est le partage des ressources d'une infrastructure réseau physique entre plusieurs réseaux logiques. Il est donc fondamental que les ressources virtuelles associées à un réseau virtuel soient ***isolées*** pour éviter toute interférence émanant d'autres réseaux virtuels avec des conséquences sur les performances et la sécurité. A titre d'exemple, l'isolation des ressources d'un réseau virtuel permet de le protéger, d'un point de vue performance, d'un dysfonctionnement de tout autre réseau virtuel qui serait amené par son état de charge à consommer la quasi-totalité des ressources physiques. Une seconde forme d'isolation concerne le trafic de données et de contrôle qui doit rester confiné dans son réseau virtuel d'appartenance et ne doit aucunement se retrouver dans un autre réseau virtuel.

Le principe d'***indépendance*** des réseaux virtuels explicité ci-avant stipule que chaque réseau virtuel (ou son fonctionnement) peut être customisé en toute indépendance et donc exhiber des caractéristiques (en termes de Qualité de Service (QdS), performances, sécurité, etc.) qui lui sont propres pour offrir des services spécifiques. Cette personnalisation peut s'exprimer de différentes manières en reposant sur des politiques / optimisations (QdS, sécurité, etc.) propres à chaque réseau virtuel, si ces derniers se partagent la même pile protocolaire. Elle peut même reposer sur l'utilisation de piles protocolaires spécifiques à chaque réseau virtuel.

En résumé, un réseau virtuel est une abstraction d'un réseau avec des nœuds et des liens. Les nœuds exécutent des fonctions réseau de bas niveau (fonctions d'acheminement du plan de données) mais potentiellement de haut niveau (typiquement du plan de contrôle et de gestion). Chaque nœud d'un réseau virtuel possède ses propres instances des fonctions réseau qu'il exécute ; il a également la possibilité de les customiser en toute indépendance. Ces fonctions réseau nécessitent des ressources mises à disposition par l'infrastructure physique. Ces ressources virtuelles sont obtenues par partitionnement (partage + isolation) des ressources de l'infrastructure physique par le biais de techniques de **virtualisation des équipements réseau** et de techniques de **virtualisation des liens** développées ci-après. Ainsi, les fonctionnalités offertes par un réseau virtuel vont de la simple connectivité avec éventuellement des garanties de performances ou de QdS au support de nouveaux protocoles avec de nouveaux services à valeur ajoutée.

Le concept de virtualisation réseau tel qu'introduit ci-avant est très général et peut s'instancier ou s'appliquer de différentes manières. Une multitude de technologies réseau historiques qui datent de plusieurs décennies relèvent de la virtualisation réseau. C'est notamment le cas des VLAN (Virtual LAN), les réseaux à commutation de circuits virtuels (tels qu'ATM), MPLS, VPN (Virtual Private Networks), et les réseaux overlays applicatifs. La technologie VLAN permet de faire coexister plusieurs réseaux virtuels Ethernet isolés (principalement de l'isolation du trafic et donc des domaines de diffusion) sur une infrastructure physique de type Ethernet commuté. Cette technologie repose principalement sur des techniques de virtualisation de liens avec une instanciation des fonctions réseau de niveau 2 par VLAN. La customisation d'un VLAN demeure très réduite et nécessairement basée sur de l'Ethernet. En revanche, une certaine flexibilité est possible dans la mesure où le rajout/retrait d'interfaces (et donc de stations terminales qui y sont reliées) peut se faire dynamiquement.
Les réseaux ATM (ou assimilés) et MPLS intègrent un aspect/forme important de la virtualisation réseau, à savoir la virtualisation des liens. En effet, un circuit virtuel ATM ou un LSP (Label Switched Path) MPLS n'est autre qu'un lien virtuel de bout-en-bout (du point de vue d'un nuage ATM ou MPLS). ATM garantit l'isolation des ressources consommées par chaque circuit virtuel grâce à la signalisation et aux mécanismes de réservation de ressources qu'il implémente. Ce niveau d'isolation est également possible avec MPLS avec des protocoles de distribution de labels orientés QdS. Les technologies ATM et MPLS n'implémentent qu'une forme de virtualisation réseau, à savoir la virtualisation des liens. Les possibilités de personnalisation et le niveau de flexibilité demeurent donc limités.
Les technologies VPN permettent de mettre en place des réseaux privés sur une infrastructure publique. De ce fait, elles correspondent à de la virtualisation réseau. Plusieurs

solutions VPN existent sur le marché, les plus populaires sont les solutions VPN de niveau 2 (Ethernet point à point ou multipoint) et de niveau 3 (VPN-IP avec la possibilité de fixer sa propre topologie). Les solutions VPN reposent principalement sur des techniques de « virtualisation de liens » en se basant sur des techniques de tunnelling (MPLS, IP, GRE, etc.). Certaines solutions reposent également sur des techniques de « virtualisation d'équipements réseau » assez basiques (par exemple, les VRF (Virtual Routing and Forwarding) utilisées dans les solutions VPN-IP déléguées au prestataire de services) et limitées aux nœuds de bordure (points de présence de ces services) et aucunement aux nœuds de cœur. Les solutions VPN introduisent un certain niveau de customisation avec la possibilité d'utiliser des protocoles différents au niveau de chaque VPN. Cette liberté demeure néanmoins limitée aux protocoles usuels (typiquement de l'architecture TCP/IP et/ou IEEE) et également aux protocoles supportés par les nœuds de l'infrastructure physique sous-jacente. Ces techniques souffrent également d'absence de flexibilité puisque statiques (du moins, pour les offres VPN actuelles).

Les réseaux overlay applicatifs sont une dernière forme de virtualisation réseau dont le but est d'offrir via l'Internet des services avec valeur ajoutée (en termes de QdS, résilience, sécurité, modes de communication, etc.). Un réseau overlay est constitué de *Routeurs Overlay* (RO) déployés sur des machines dédiées à l'extrémité de l'Internet (voir Figure B.1.2). Ces RO sont reliés entre eux par les liens virtuels traversant l'Internet (en utilisant des techniques de tunneling). Les usagers sont à leur tour reliés à un routeur overlay via un lien virtuel. Chaque réseau overlay peut utiliser sa pile protocolaire (réseau et couches supérieures) faite maison qui peut n'avoir aucun lien de parenté avec l'architecture TCP/IP. C'est l'un des attraits des réseaux overlays mais il est important de voir que cette liberté de personnalisation reste limitée à l'extrémité de l'Internet et ne touche aucunement le cœur du réseau Internet qui demeure conforme à l'architecture TCP/IP. Le contrôle qu'ont les routeurs overlay sur les tunnels demeure également limité ce qui impact le niveau de flexibilité qu'exhibe ce type de réseau.

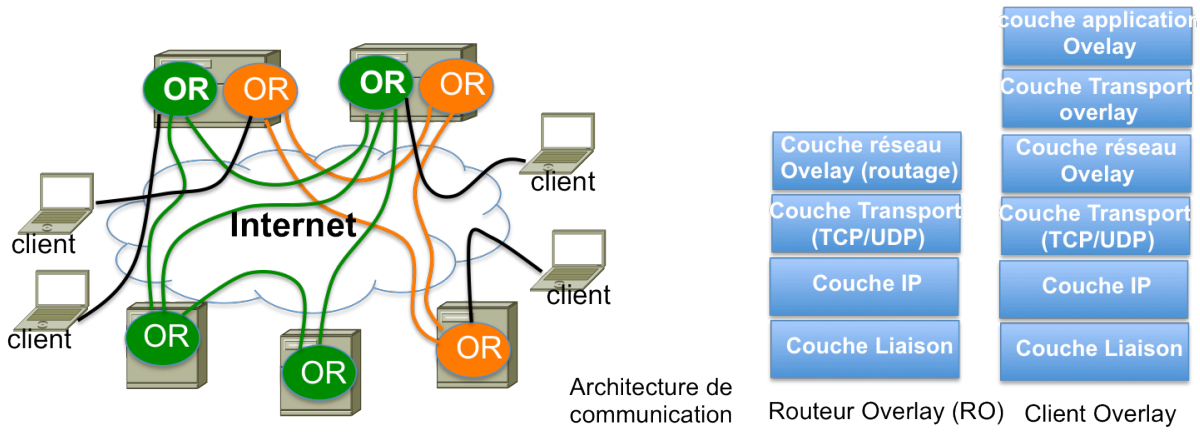


Figure B.1.2 Architecture générale des réseaux overlays

La virtualisation réseau fait dernièrement l'objet de beaucoup d'attention aussi bien de la part de la recherche académique qu'industrielle. Même si une multitude de technologies relevant de la virtualisation réseau existent sur le marché depuis bien longtemps, ces

dernières ne répondent pas complétement aux nouvelles attentes liées à la virtualisation réseau. Ces attentes s'expriment selon différentes directions :

1. Virtualisation partout et à tous les niveaux (« full virtualization ») : la virtualisation doit concerner tous les éléments d'une infrastructure physique et notamment les nœuds internes de ce réseau (aucune solution historique ne l'intègre). Elle doit potentiellement permettre de toucher toutes les ressources ainsi que toutes les fonctions réseau qu'elles soient de bas ou haut niveau ou qu'elles soient au niveau du plan de données ou de contrôle.
2. Indépendance et programmation : la virtualisation réseau doit permettre que chaque réseau virtuel puisse reposer sur une pile protocolaire ad-hoc de son choix (qui peut n'avoir aucun lien avec les architectures de communications historiques) voire même exhiber des possibilités de programmation très fines.
3. Provisionnement à la demande : la virtualisation réseau doit permettre le provisionnement à la demande d'un réseau virtuel avec des caractéristiques prédéfinies.
4. Flexibilité (Elasticité) : la virtualisation réseau doit permettre que le fonctionnement d'un réseau virtuel soit adapté à la demande en fonction des besoins durant sa phase opérationnelle.

Les deux premières attentes sont issues des programmes de recherche initiés durant ces dernières années (et dont une partie est encore en cours) sur les réseaux du Futur : les programmes *Future Internet Architecture* (FIA) [FIA] ou GENI (*Global Environment for Network Innovations*) [GENI] de la NSF (*National Science Foundation*), *l'European Future Internet Assembly* [EFIA] qui fédère plusieurs projets sur « *Future Internet Research* » et enfin le programme Japonais NGN (*New Generation Network*) [NGN]. En effet, le point de départ de ces programmes est que les approches incrémentales ne remettant pas en cause fondamentalement l'architecture TCP/IP ne peuvent permettre aux réseaux du futur de fournir des services diversifiés répondant aux besoins des applications courantes et à venir. La « *full virtualization* » et l'« *indépendance et programmation* » ont été portés comme solutions pour pouvoir adresser les applications à venir tout en offrant la possibilité de maintenir l'architecture TCP/IP actuelle au sein d'un réseau virtuel.
Les deux dernières attentes émanent du « cloud computing » qui pose de nouvelles exigences aux réseaux. La virtualisation réseau répond à l'exigence relative au support du « multi-tenancy ». La « mise en place du réseau virtuel à la demande » et l'« élasticité » reflètent simplement l'élargissement du modèle de consommation cloud des ressources informatiques aux réseaux de communications qui permettent de les relier, i.e. le cloud networking.

B.2 TECHNIQUES DE VIRTUALISATION RESEAU

B.2.1 VIRTUALISATION DES INTERFACES/CARTES RESEAU

La virtualisation des interfaces réseau est une brique de base de la virtualisation des machines. Elle vise à partager une interface réseau entre plusieurs machines virtuelles via des interfaces réseau virtuelles. Plusieurs solutions de virtualisation des interfaces existent et

sont poussées par les principaux acteurs industriels de la virtualisation des serveurs (tels que Vmware, Xen, Microsoft, etc.). Elles se distinguent par le niveau de performance et de compatibilité qu'elles permettent et peuvent être classées en deux familles :

- Solutions purement logicielles de virtualisation des interfaces

Elles reposent sur une approche classique dans laquelle l'hyperviseur assure entièrement le rôle d'isolation et de partage de(s) interface(s) physique(s) entre les interfaces virtuelles ainsi que l'acheminement du trafic entre interfaces virtuelles et entre interfaces virtuelles et physiques. Il émule donc le comportement d'un *switch* auquel sont reliées les interfaces virtuelles et physiques (voir Figure B.2.1) par l'interception et la retraduction des requêtes et commandes d'entrées/sorties ainsi que par des copies de (vers) les zones mémoires des machines virtuelles sources (resp. destinataires). Cela se traduit par un sur-débit inhérent à l'activation récurrente de l'hyperviseur pour pouvoir traiter les arrivées/départs de paquets et limite le débit maximal qui peut être atteint [SLF12].

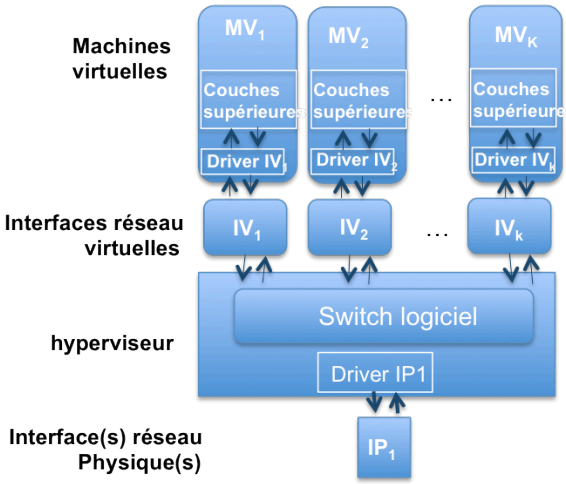


Figure B.2.1 Principe général des solutions logicielles de virtualisation des interfaces

Deux principales approches se distinguent : La première cherche à émuler parfaitement les interfaces physiques ce qui permet d'utiliser au niveau des machines virtuelles les drivers des interfaces physiques sans aucune modification. Le prix à payer se traduit à chaque transmission par une interruption de l'exécution d'une VM pour activer l'hyperviseur qui se charge de la relayer vers l'interface physique. Elle implique également la traversée de deux drivers identiques (de l'interface virtuelle et physique). La deuxième approche nécessite l'emploi d'un driver modifié au niveau des VM qui tient compte du fait que les interfaces sont virtuelles et qui travaille en concert avec l'hyperviseur. Il contrôle notamment plus efficacement le moment où une VM donne la main à l'hyperviseur pour pouvoir relayer ses paquets. Cette activation de l'hyperviseur peut concerner plusieurs transmissions et non pas une seule comme précédemment. Pour des raisons de performances, cette approche, connue sous le nom « *para-virtualized driver* », est la plus courante dans les solutions de virtualisation de machines (Xen, Vmaware et KVM).

- Solutions logicielles assistées par des extensions matérielles pour la virtualisation des interfaces

Ces solutions reposent sur le partitionnement de certaines ressources matérielles (mémoires, files, lignes d'interruptions, etc.) des interfaces réseaux physiques pour permettre aux machines virtuelles d'avoir un accès direct (sans passer systématiquement par l'hyperviseur) au matériel et donc réduire le sur-débit des solutions purement logicielles. Elles nécessitent donc des interfaces réseaux avec des spécificités/extensions liées à la virtualisation. Deux principales solutions existent [SLF12] : Virtual Machine Device Queue (VMDQ) [VMDQ09] et Single Root IO virtualization (SR-IOV) qui est devenu le standard industriel [SRIOV1]. La première repose sur l'exploitation au niveau de l'interface réseau de plusieurs lignes d'interruptions qui peuvent être associées à des VM différentes notamment pour leur signifier individuellement, sans faire intervenir l'hyperviseur, l'arrivée d'un paquet (la VM étant identifiée sur la base de l'adresse de destination). L'interface est également capable d'effectuer des accès DMA (Direct Memory Access) pour placer le paquet reçu dans la mémoire de la machine hôte. Au traitement de sa demande d'interruption, la VM cède la main à l'hyperviseur pour lui récupérer le paquet. Une interface SR-IOV lève cette dernière limite. A l'arrivée d'un paquet, en plus de signifier une demande d'interruption à la VM concernée, elle effectue un accès DMA dans la zone mémoire de celle-ci sans intervention de l'hyperviseur.

B.2.2 VIRTUALISATION DES EQUIPEMENTS RESEAU

Plusieurs solutions permettent de faire coexister plusieurs équipements réseaux virtuels sur un même équipement réseau. Elles se distinguent par le modèle d'isolation/partage de ressources adopté ainsi que par les ressources considérées par la virtualisation. Elles exhibent donc des degrés de virtualisation et des niveaux de performances différents. La Figure B.2.2 illustre ce concept de virtualisation d'équipements sur le cas particulier d'un routeur physique qui supporte deux routeurs virtuels Rv_A et Rv_B. Chaque routeur virtuel dispose de son propre plan de contrôle avec sa propre instance du protocole de routage, sa propre table de routage/acheminement et son propre plan de données (en charge de l'acheminement des paquets). La représentation des deux routeurs virtuels en deux blocs séparés laisse supposer qu'ils reposent sur des ressources dédiées (CPU et mémoire pour le plan de contrôle, commutation, mémoire, interfaces réseau et liens de transmission pour le plan d'acheminement) obtenus potentiellement par partitionnement. Cet exemple illustre le déploiement d'une solution de virtualisation développée où toutes les ressources physiques sont virtualisées par partitionnement et où chaque équipement virtuel dispose de ses propres instances des fonctions réseau. Toutes les solutions que nous présentons ci-après n'exhibent pas ce degré de virtualisation.

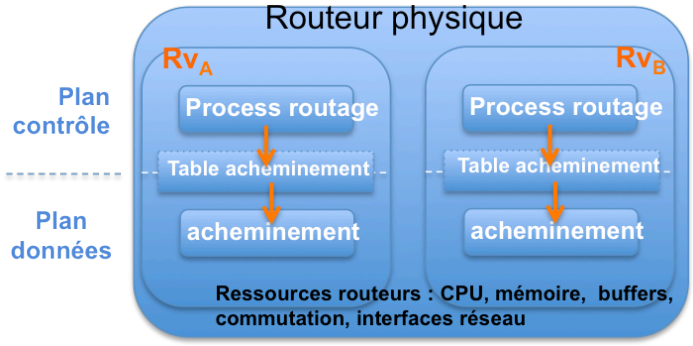


Figure B.2.2 Principe général de la virtualisation d'un routeur

Un premier type de solutions de virtualisation des équipements repose sur la virtualisation d'hôtes (ou virtualisation du système opératoire) en permettant de faire coexister plusieurs machines virtuelles (VM) avec leur instance du système opératoire sur le même équipement. Ces VM exécutent ensuite les fonctions réseau assurées par l'équipement virtuel (par exemple : fonctions de routage et fonctions d'acheminement du plan de données). Ces solutions purement logicielles se prêtent bien à des nœuds physiques terminaux où une multitude de solutions de virtualisation d'hôte reconnues existent (VMware, Xen, KVM, etc.). La solution Brocade Vyatta 5400 *vrouter* [Vrouter14] est un exemple. Elles s'appliquent également à des architectures matérielles plus spécifiques ciblant les équipements réseau avec quelques réserves sur les performances, par exemple les solutions de virtualisation d'Untangle [Untangle12] et Router OS de Mikrotik [RouterOS11].

Un deuxième type de solutions beaucoup plus courant - car disponible sur le marché depuis plusieurs années - se focalise sur la virtualisation d'un élément central des équipements réseau, à savoir la table d'acheminement/routage. Ces solutions connues sous le nom de « *Virtual Routing and Forwarding* » (VRF) permettent d'offrir plusieurs instances séparées de la table d'acheminement/routage, i.e. le nœud virtuel dispose de sa propre table. Le partage et surtout l'isolation entre nœuds virtuels des ressources nécessaires à l'acheminement et au routage demeurent basiques. D'ailleurs, certaines solutions utilisent un seul processus de routage pour alimenter les différentes tables d'acheminement des nœuds virtuels. D'autres, en revanche, offrent des instances de processus de routage propres à chaque nœud virtuel.

Un dernier type de solutions qui concernent les équipements réseaux les plus performants repose sur une isolation et un partitionnement de ressources par *hardware* et non par logiciel comme les solutions décrites ci-avant [HVR12].

B.2.3 VIRTUALISATION DES LIENS

La virtualisation des liens est une pratique très ancienne et très courante dans les réseaux qu'ils soient informatiques ou de transmission. Pour cette raison, nous ne l'aborderons que de manière très synthétique. Le composant de base de la virtualisation des liens est la virtualisation de la ressource de transmission matérialisée par la capacité de transmission d'un lien physique (i.e. débit). Cette dernière est mise en œuvre par les techniques de multiplexage ou d'accès multiple selon le type de lien considéré (point-à-point ou multipoints). Selon l'algorithme d'ordonnancement ou la technique d'accès multiple utilisés

différents niveaux d'isolation/partitionnement de ressources peuvent être obtenus. La virtualisation des ressources de transmission peut également impliquer l'agrégation de plusieurs liens physiques en un seul lien logique, pratique courante dans un contexte LAN (Local Area Network) avec les techniques d'agrégation des liens Ethernet dont celles basées sur le protocole LACP (Link Aggregation Control Protocol) de l'IEEE.
Un lien virtuel peut reposer sur un voire plusieurs chemins de données (i.e. il peut traverser plusieurs liens physiques interconnectés via des équipements réseau). On parle alors de virtualisation du chemin de données dans la mesure où les éléments des chemins de données (liens physiques et équipements réseau) sont partagés entre plusieurs liens virtuels. La virtualisation du chemin de données repose sur l'une ou l'autre des techniques suivantes :

- Une isolation des liens virtuels basée sur l'intégration de labels dans les paquets qui sont utilisés par les équipements réseau pour l'acheminement. Plusieurs technologies rentrent dans ce cadre : MPLS, ATM et Frame-Relay et l'utilisation des labels 802.1Q dans le contexte LAN Ethernet.
- Une isolation basée sur des techniques de tunneling avec l'encapsulation d'un paquet dans un autre, l'en-tête externe servant à l'acheminement et comme moyen d'isolation entre liens virtuels. Plusieurs protocoles de tunneling sont utilisés en pratique : IP-dans-IP, GRE (Generic Routing Encapsulation), tunnels MPLS (i.e. Label Switched Path (LSP)), L2TP (Layer 2 Tunneling Protocol), et plus récemment dans le contexte centre de données : VXLAN (Virtual eXtensible Local Area Networks), NVGRE (Network Virtualization using Generic Routing Encapsulation), STT (Stateless Transport Tunneling Protocol), etc.

B.3 ECOSYSTEME DE LA VIRTUALISATION RESEAU

Le concept de virtualisation réseau fait très simplement apparaître deux rôles (chacun en charge d'un certain nombre de fonctionnalités et devant assumer des responsabilités aussi bien technique qu'économiques/d'affaires) : le fournisseur de l'infrastructure physique (ou provider d'infrastructure) et le prestataire de service (provider de service).
Le provider d'infrastructure est responsable de la gestion et l'exploitation de son infrastructure, constituée de ressources physiques telles que les routeurs et les liens qui peuvent être virtualisés en plusieurs ressources virtuelles. Via des interfaces de contrôle offertes à ses clients (dans ce cas de figure, les prestataires de service), il offre la possibilité de mobiliser des ressources virtuelles localisées sur des éléments physiques différents pour leur permettre de constituer et mettre en place un réseau virtuel. Le prestataire de service est également responsable de la configuration, l'exploitation et gestion du réseau virtuel. C'est donc lui qui fixe les fonctions réseau exécutées dans le réseau virtuel. Il est, enfin, responsable de la mise en place des services applicatifs qu'il offre à ses clients. Ce modèle économique à deux rôles est le modèle historique proposé dans [FGR07].
Un second modèle de rôles avec des responsabilités atomiques beaucoup plus fines a été proposé par le projet européen FP7 4WARD [4WARD, SWPF09]. Ce modèle, décrit à Figure B.3.1, introduit quatre rôles différents avec certaines des responsabilités du provider de service renvoyées sur deux nouveaux rôles : le provider de réseaux virtuels et l'opérateur du réseau virtuel. Avec ce nouveau modèle, le provider d'infrastructure détient et gère des ressources réseau physiques qu'il virtualise et met à disposition de providers de réseaux virtuels via une interface de contrôle permettant d'établir des requêtes de ressources virtuelles auprès du provider d'infrastructure. Le provider de réseaux virtuels (VNP-Virtual Network Provider) construit un réseau virtuel (ou plus précisément une topologie réseau virtuelle pour insister sur le fait que le réseau virtuel ainsi créé n'est qu'une combinaison de ressources virtuelles élémentaires) en assemblant des ressources virtuelles émanant d'un ou plusieurs providers d'infrastructure. Le réseau virtuel ainsi construit est mis à disposition d'un opérateur de réseau virtuel (VNO-Virtual Network Operator) ou d'un autre provider de réseaux virtuels pour construire d'autres réseaux virtuels via une interface de contrôle. Le VNO repose sur la topologie virtuelle construite par le VNP pour mettre en place les fonctionnalités réseau permettant d'offrir le service de connectivité répondant aux besoins du provider de services (en termes de QdS, sécurité, etc.). Sur cette topologie virtuelle et via une interface (par exemple, type console), il instancie une architecture réseau au niveau du réseau virtuel par des opérations d'installation et de configuration puis assure sa gestion. En d'autres termes, le VNO transforme un assemblage de ressources virtuelles (topologie virtuelle) en un réseau virtuel opérationnel répondant aux attentes du provider de service. Le provider de service utilise un réseau virtuel opérationnel et customisé à ses besoins pour offrir ses services applicatifs à ses usagers.
Les deux nouveaux rôles introduits par le modèle 4ward permettent au provider de services de s'affranchir des responsabilités liées à l'assemblage de ressources virtuelles qui peuvent pour des raisons de localisation ou de capacité appartenir à plusieurs providers d'infrastructure. Le rôle du VNP est d'orchestrer les demandes de ressources virtuelles pour le compte du provider de service. Indépendamment de l'origine des ressources virtuelles, il est l'unique interlocuteur pour déployer une topologie virtuelle aussi complexe soit-elle. Le VNO permet au provider de service de déléguer ou d'externaliser l'exploitation et la gestion de son réseau virtuel pour lui permettre de centrer ses responsabilités sur la partie service.

Cette décomposition en rôles permet de délimiter les responsabilités. Un acteur économique peut endosser un rôle ou même combiner plusieurs rôles en même temps. C'est notamment le cas actuellement avec plusieurs prestataires endossant le rôle de provider d'infrastructure et de réseaux virtuels en relations économiques avec des opérateurs virtuels.

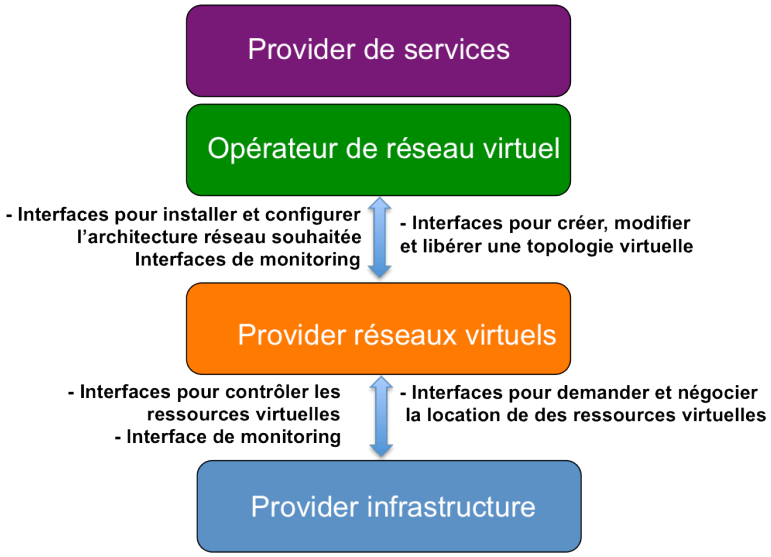


Figure B.3.1 Modèles de rôles/acteurs 4ward

B.4 EXEMPLES D'APPLICATION DE LA VIRTUALISATION RESEAU

D 'une manière générale et sans considérer des cas d'application spécifiques, la virtualisation réseau offre un certain nombre d'avantages :

- Pour le provider d'infrastructure, le moyen de mutualiser ses ressources et aller vers une utilisation optimale des ressources. Cela permet de rentabiliser ses coûts d'investissements et d'exploitation. Cet avantage est également valable pour un provider de réseaux virtuels qui à son tour peut partitionner ses ressources virtuelles et les partager entre plusieurs clients.
- Pour un client utilisateur d'un réseau virtuel, les avantages se traduisent par une simplification de la gestion du réseau et également par les possibilités de personnalisation du réseau virtuel qui lui est offert.

Plusieurs applications de la virtualisation réseau existent déjà sur le marché depuis plusieurs années. Nous nous focalisons dans la suite sur les scénarios d'application qui émergent et dont certains sont à l'origine des évolutions récentes de la virtualisation réseau (vers le « tout virtualisé », l'élasticité, etc.).

B.4.1 SUPPORT DE SERVICES DIVERSIFIES / RESEAUX VIRTUELS SPECIFIQUES AUX SERVICES

Il est actuellement admis que les solutions incrémentales bâties sur l'architecture TCP/IP ne peuvent constituer une réponse au support de la multitude et la diversité des services actuels et à venir [ITU12]. Certains services reposeront sur des architectures de communication complétement différentes de l'architecture TCP/IP et devront coexister avec celle-ci. La virtualisation réseau joue le rôle d'un catalyseur pour permettre le support efficace de ces

architectures et services dans la mesure où elle permet d'instancier plusieurs réseaux virtuels (chacun avec son architecture de communication et offrant un service spécifique) sur une même infrastructure physique. Cela évite la prolifération de réseaux spécialisés avec les coûts d'investissement et d'opération associés.

Plusieurs services actuels avec des besoins très spécifiques et qui ne peuvent se satisfaire des possibilités de l'Internet reposent sur l'utilisation de réseaux overlay applicatifs avec potentiellement des piles protocolaires spécifiques au niveau des routeurs overlay mais reliés via l'Internet avec peu de contrôle sur les liens virtuels qui les relient (se référer à Figure B.1.2) Ce cas d'application se distingue des solutions actuelles par le fait que la virtualisation et les architectures de communication peuvent toucher tous les nœuds réseau et pas uniquement des nœuds d'extrémité (Cela ouvre le champ à plus de possibilités et notamment au niveau des services offerts).

B.4.2 SUPPORT DE RESEAUX PLURI-GENERATIONNELS

Ce scénario est un cas particulier du cas d'application décrit dans la section précédente. L'idée est de se baser sur la virtualisation réseau pour pouvoir faire cohabiter sur une même infrastructure réseau plusieurs générations ou évolutions d'une même technologie réseau. A titre d'exemple, une même infrastructure réseaux cellulaire (ou une partie de celle-ci) peut être utilisée pour supporter en même temps plusieurs des technologies réseaux cellulaires (2G, 3G, 4G, ..) qui se sont multipliées ces dernières années.

B.4.2.1 MIGRATION DE SERVICES APPLICATIFS

La virtualisation des serveurs et d'hôtes se généralise particulièrement au niveau des centres de données. Certains composants d'une application (notamment des serveurs applicatifs) peuvent résider sur des machines virtuelles ce qui peut permettre d'envisager la possibilité de les faire migrer d'une localisation physique (machine) vers une autre. Cette migration du composant applicatif peut être motivée par des politiques propres au provider de l'application (gestion de ses ressources) et/ou par souci d'améliorer l'expérience utilisateur (Quality of Experience (QoE)) en rapprochant le composant (la machine virtuelle associée) de ce dernier. A titre d'exemple, un provider de jeux en réseau pour usagers mobiles peut décider de faire migrer la localisation d'un serveur de jeu pour se rapprocher d'un groupe d'usagers qui se sont déplacés vers une zone donnée pour suivre un événement sportif, culturel etc. Cette migration permet d'améliorer la qualité de l'accès au service (délais plus faibles) et donc la QoE des usagers et potentiellement réduire les coûts des communications. La flexibilité portée par les évolutions récentes de la virtualisation réseau permet une reconfiguration à la volée du réseau virtuel support du service applicatif pour répondre efficacement, tout en respectant des exigences de performances prédéfinies, à la mobilité de certains composants applicatifs. Cela introduit une certaine flexibilité au niveau des providers de service.

B.4.3 EXPERIMENTATION/TEST A GRANDE ECHELLE SUR RESEAUX OPERATIONNELS

Tester et évaluer un nouveau service réseau ou une nouvelle technologie à large échelle et sur un vrai réseau opérationnel est crucial pour un prestataire de service ou un opérateur réseau. De peur d'affecter le fonctionnement d'un réseau en opération, ces expérimentations s'effectuent à petite échelle sur des plateformes réseau dédiées ce qui peut se révéler ne pas être suffisamment représentatif de la réalité lorsque déployés sur un vrai réseau d'opérateur. La virtualisation réseau peut permettre d'effectuer des expérimentations à grande échelle en déployant un réseau virtuel de grande envergure pour ces fins d'expérimentation sur l'infrastructure physique d'un ou plusieurs réseaux opérationnels [GENI06]. De plus, il est possible de customiser les caractéristiques du réseau virtuel et son déploiement selon les besoins des expérimentations.

B.4.4 LOCATION A LA DEMANDE ET ELASTIQUES DE RESSOURCES VIRTUELLES

Un opérateur détenant des ressources (physiques ou virtuelles) peut louer à la demande et de manière élastique une partition de ses ressources. Cela peut concerner des ressources individuelles ou un assemblage de ressources (topologie virtuelle). La location de ressources est une pratique courante entre opérateurs qu'ils soient virtuels ou pas. Elle concerne très souvent les liens virtuels, ne peut être établi à la demande et n'est pas élastique. Ce scénario d'application cherche à répondre à ces limites en ouvrant les ressources aux équipements réseau, et en permettant que les ressources soient louées à la demande et de manière élastique. Cela inclut par exemple l'établissement à la demande d'un lien avec une de la bande passante élastique (Bandwidth on demand). Ce cas qui relève du « cloud networking » sera repris et détaillé ci-après dans la section relative à ce thème.

B.4.5 VIRTUAL NETWORKS AS A SERVICE

Ce cas d'application correspond à la situation où similairement aux usages qui prévalent dans le contexte du « cloud computing » et de l'« IaaS », l'on fournisse à un utilisateur suite à sa demande un réseau virtuel complet répondant exactement à ses besoins qui peuvent évoluer dans le temps. L'usager a la possibilité de configurer, gérer son réseau virtuel pour répondre à ses besoins. Plusieurs sous-cas d'application ont été proposés dans ce sens et concernent différents types de réseaux dont les réseaux IP, les réseaux mobiles, etc. Ces cas relèvent du « cloud networking »et seront donc repris et détaillés ci-après.

B.4.6 AGREGATION DES ACCES SANS FIL

Le paysage des technologies réseaux d'accès sans-fil est très hétérogène. Une multitude de technologies (cellulaire, satellitaire, Wi-Fi, WiMAx) - avec des caractéristiques spécifiques (en termes de débit, capacité, couverture, etc.) qui les rendent incontournables - existent et continueront à exister sur le marché. Sur une zone prédéfinie, plusieurs types d'accès sans-fil peuvent être offerts. Les zones de couverture de ces différents types d'accès peuvent se recouvrir et/ou se compléter. La virtualisation réseau permet aussi bien le partitionnement de ressources réseau que l'agrégation de ressources. Ce cas d'application repose sur ce dernier principe. L'idée est d'agréger les différents réseaux d'accès disponibles sur une zone

donnée en un seul réseau d'accès sans-fil virtuel. Du point de vue de l'usager, seul le réseau virtuel est visible avec peu ou pas de connaissance sur la ou les technologies réseaux supports. Si les zones de couverture des réseaux d'accès se complètent, la gestion de la mobilité est alors complétement transparente à l'usager et ce même si elle implique un changement du type de réseau d'accès utilisé (mobilité verticale). Si les zones de couverture se recouvrent, cela donne la liberté à l'opérateur de gérer de manière globale ses ressources (des différents types de réseau) tout en respectant les exigences de ses clients (performances, coût, consommation d'énergie,..).

B.4.7 MIGRATION DES SERVICES ET DES FONCTIONS RESEAU

Un nœud d'un réseau virtuel repose sur des ressources virtuelles et exécute un ensemble de fonctions réseau. La virtualisation réseau ouvre la porte à la possibilité de faire migrer à la volée et de manière transparente certaines fonctionnalités réseau d'un nœud physique à un autre. Couplée à la possibilité de reconfigurer à la volée certains liens et nœuds du réseau, la migration des fonctions réseau vient renforcer plusieurs cas d'application de la virtualisation réseau dont :

- Réseaux éco-conscients : un opérateur réseau dispose de nouvelles possibilités de gestion de ses ressources. En heures creuses, Il peut décider d'éteindre/libérer certains nœuds après avoir déplacé et concentré leurs fonctionnalités sur un nombre très réduit de nœuds ;
- Réseaux capables de survivre à des désastres : en situation d'un désastre imminent, il est possible de déclencher la migration des fonctions essentielles d'un réseau vers des zones moins exposées tout en reconfigurant certains liens pour continuer à offrir le plus longtemps possible une connectivité minimale dans les endroits sinistrés ;
- Réseaux « self-healing » : en réaction à des défaillances de certains éléments d'un réseau, la migration des fonctions peut être utilisée comme moyen de recouvrement.

C SOFTWARE-DEFINED NETWORKING

C.1 PRINCIPES GENERAUX

Dans un réseau classique, les fonctions de contrôle du réseau et d'acheminement des données sont effectuées au sein des équipements réseau (tels que les routeurs). Cette architecture monolithique est considérée comme un frein et des efforts ont été faits pour définir des architectures cherchant à introduire plus de flexibilité et de simplicité dans la gestion et le contrôle du réseau. La notion de "Software-Defined Networking" (SDN) [SDN12] regroupe les architectures visant à séparer l'acheminement des données et son contrôle, principalement en retirant l'intelligence des équipements réseaux matériels et en la disposant dans des composants logiciels sur des équipements dédiés.

La Figure C.1.1 illustre l'architecture SDN et la compare avec l'architecture d'un réseau classique. La séparation du plan de données du plan de contrôle permet une visualisation et une gestion plus globale du réseau. Le déploiement de nouveaux protocoles ou de nouvelles applications est alors facilité, l'intelligence du réseau étant logiquement regroupée dans une entité spécifique, appelée le contrôleur SDN. Comme on peut le voir sur la Figure C.1.1, le contrôleur devient le point central de l'architecture SDN. La partie supérieure englobe les applications agissant sur le réseau telles que le routage, le contrôle d'accès, la surveillance du réseau... L'interaction avec le contrôleur se fait via des API et les applications obtiennent ainsi une vision simplifiée du réseau sous-jacent. Le plan inférieur est composé d'éléments physiques chargés de l'acheminement des données, typiquement des commutateurs. Le contrôleur situé entre les applications et les équipements physiques peut alors, au travers de différents services, interpréter les consignes des applications et les répercuter sur le réseau. Idéalement, les interfaces Nord (Application-Contrôleur) et Sud (Contrôleur-Routeur) proposent des API ouvertes afin de permettre une meilleure interopérabilité entre les systèmes et matériels de différents constructeurs (voir Figure C.1.2 - Architecture de SDN vue par l'ONF [ONF 2013a]. Vision classique et haut-niveau de l'architecture SDN.qui présente la vision de l'Open Networking Foundation (ONF), principal promoteur des réseaux SDN).

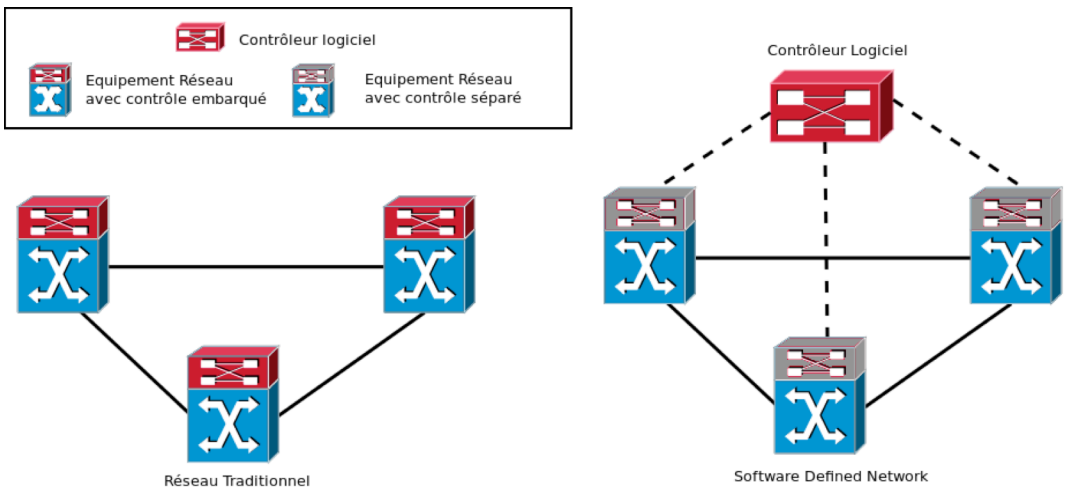


Figure C.1.1 – Architectures réseau traditionnelle et architecture d'un Software Defined Network (inspiré de [Nunes 2014]).

Pour l'instant, les efforts des chercheurs et des industriels se sont focalisés sur l'interface Sud car celle-ci est indispensable à la programmation du plan de données. Nous verrons par la suite que les approches sont légèrement différentes mais que le principe reste globalement le même. Nous détaillerons deux propositions qui nous semblent les plus abouties : OpenFlow [McKeown 2008] et ForCES [Doria 2010].

Si les travaux relatifs à l'interface Nord ne connaissent pas la popularité des protocoles de l'interface Sud (OpenFlow monopolisant les discussions sur les SDN), le succès futur des SDN passera pourtant par l'existence d'API clairement définies au niveau de l'interface Nord. En effet, ceci est indispensable pour le développement d'applications et, donc, une exploitation optimale des architectures SDN. Nous ferons un point sur les mécanismes actuels et sur les objectifs à respecter.

Bien que prometteuse, l'approche SDN soulève plusieurs problématiques et défis qui font actuellement l'objet de plusieurs travaux de recherche [Nunes 2014]. La centralisation de l'intelligence dans le contrôleur pose le problème du passage à l'échelle. Un nouveau flux entrant dans un commutateur doit être identifié par son contrôleur pour qu'il puisse établir une règle adéquate. Le nombre de nouveaux flux acceptés par seconde dépend alors directement des performances du contrôleur. Il a été démontré dans [Tootoonchian 2012] sur différents contrôleurs OpenFlow que ceux-ci pouvaient traiter plus de 50000 nouveaux flux par seconde dans différents scénarios. Une autre problématique est le placement du contrôleur, que ce soit par rapport au nombre de contrôleurs à utiliser ou à son emplacement dans le réseau physique. Les études menées dans [Phemius 2013] concluent que la bande-passante influe sur le nombre de nouveaux flux acceptés ainsi que le taux de pertes en cas de congestion sur le lien. La latence sur le lien contrôleur-commutateur va quand-à-elle influencer directement la latence introduite lors de l'acceptation d'un nouveau flux.

Dans une architecture SDN, les commutateurs sont délestés du contrôle pour se concentrer uniquement sur la transmission des données. Le plan de contrôle de plusieurs commutateurs pouvant être regroupé dans un seul contrôleur, celui-ci peut devenir un point faible pour le réseau. Il est donc nécessaire d'avoir une distribution ou une redondance en cas de panne du contrôleur. Pour lutter contre cela, des propositions d'architecture logiquement centralisée et physiquement distribuée ont été faites. Par exemple, Onix [Koponen 2010] et Hyperflow [Tootoonchian 2010] permettent d'avoir un plan de contrôle distribué tout en offrant aux applications une vision centralisée du réseau. Des compromis sont alors nécessaires afin de pouvoir offrir cette vue centralisée tout en gardant une certaine réactivité [Levin 2012]. Une approche intéressante est l'utilisation d'une architecture hybride comme Kandoo [Kandoo 2012] qui propose de hiérarchiser les contrôleurs en établissant deux couches: les contrôleurs de la couche inférieure ne sont pas interconnectés et agissent uniquement dans leur domaine local tandis la couche supérieure est un contrôleur logiquement centralisé qui connaît l'état de la globalité du réseau. Il est ainsi possible de limiter les interventions du contrôleur principal et d'obtenir une meilleure réactivité pour les applications locales.

Les problématiques citées ci-dessus sont directement liées à la structure de l'architecture SDN. En plus de cela, une solution de SDN doit définir comment sont définies les règles présentes dans les commutateurs: de manière réactive ou de manière proactive. La définition

des règles de façon proactive correspond à une architecture classique comme Ethane [Casado 2007] où les commutateurs vont renvoyer les nouveaux flux vers les contrôleurs. Comme expliqué précédemment, cela risque de dégrader les performances du contrôleur suivant le nombre de nouveaux flux par seconde. Afin de définir les règles de manières proactives, DIFANE [Yu 2010] propose qu'au lieu de transmettre les nouveaux flux vers le contrôleur, le routeur de bordure transmette ceux-ci vers des routeurs intermédiaires qui contiennent les règles de transmission adéquates. Le contrôleur est alors uniquement en charge de répartir ces règles entre les commutateurs intermédiaires.

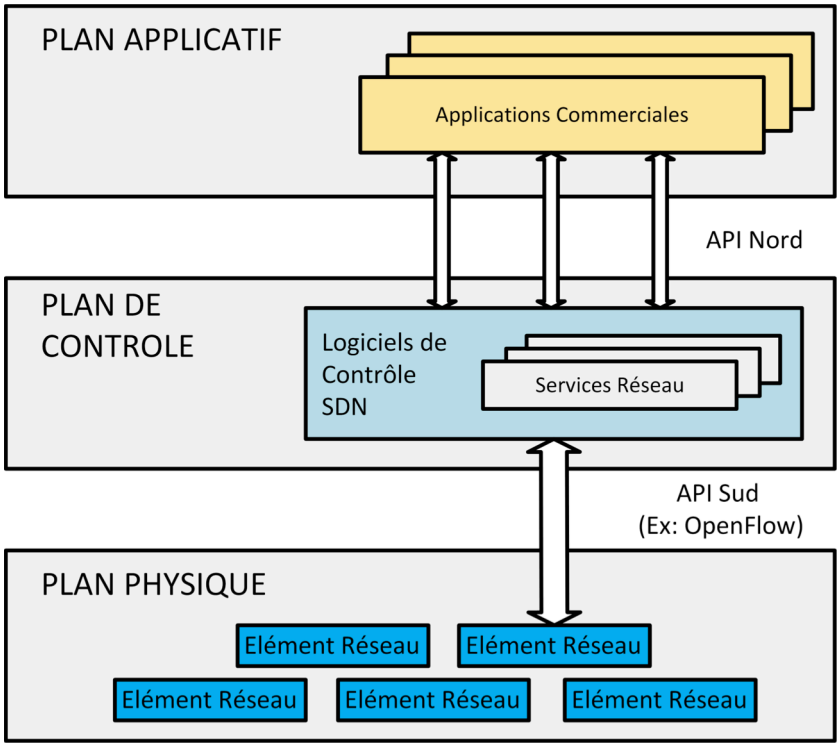


Figure C.1.2 - Architecture de SDN vue par l'ONF [ONF 2013a]. Vision classique et haut-niveau de l'architecture SDN.

C.2 L'Open Networking Foundation (ONF)

L'Open Networking Foundation (ONF) est une organisation à but non-lucratif crée en 2011. Son objectif est de promouvoir les SDN par le développement de standards ouverts. De nombreux acteurs soutiennent cette organisation, notamment Deutsche Telekom, Google, Facebook, Microsoft, … La réalisation la plus notable de cette organisation pour l'instant est le standard OpenFlow qui permet la programmation à distance du plan de données. Néanmoins, tous les groupes de travail actifs au sein de l'ONF ne se focalisent pas sur ce standard :

- « Architecture and Framework » : standardisation des SDN en essayant de définir les problèmes architecturaux liés aux SDN [ONF 2013b].
- « Configuration and Management » : étude des problèmes d'administration et de gestion liés au standard OpenFlow et définit notamment OF-CONFIG [ONF 2014].
- « Extensibility » : définition du protocole OpenFlow inclus dans les commutateurs [ONF 2013c] et développement d'extensions pour celui-ci.

- « Forwarding Abstraction » : définition d'une couche d'abstraction « Hardware Abstraction Layer » (HAL) pour la prise en charge de commutateurs non-OpenFlow.
- « Market Education » : en charge de la promotion des SDN [ONF 2013a] mais aussi de la définition de cas d'étude [ONF 2013d].
- « Migration » : proposition de méthodes pour migrer des réseaux traditionnels vers des SDN basés sur le standard OpenFlow [ONF 2013e].
- « Northbound Interface » : définition des besoins, de l'architecture et d'une implémentation pour les interfaces entre contrôleurs et applications (NBI).
- « Optical Transport » : étude des apports des SDN dans les réseaux optiques et définition d'une architecture.
- « Testing and interoperability » : test et certification des implémentations d'OpenFlow faites par les équipementiers. Organisation de PlugFests.
- « Discussion Groups » : forums discutant des SDN. Réservé aux sociétés membres.
- « Wireless & Mobile » : définition de cas d'études et détermination des besoins protocolaires et architecturaux pour étendre les technologies de l'ONF dans les domaines sans-fil et mobiles.

Certaines propositions issues de ces groupes sont donc intéressantes pour l'ensemble des solutions SDN, même si celles-ci n'utilisent pas le standard OpenFlow. Par exemple, [ONF 2013b] offre la vision de l'ONF des composants constituants une architecture SDN (voir). Ces définitions permettent de mieux appréhender les éléments constituant l'architecture, quels sont leurs rôles et comment ils interagissent. En effet, en plus des couches application, contrôle et données, la Figure C.2.1 illustre les informations qui peuvent circuler entre les plans et quelles actions sont réalisables par les organismes de gestion et d'administration du système.

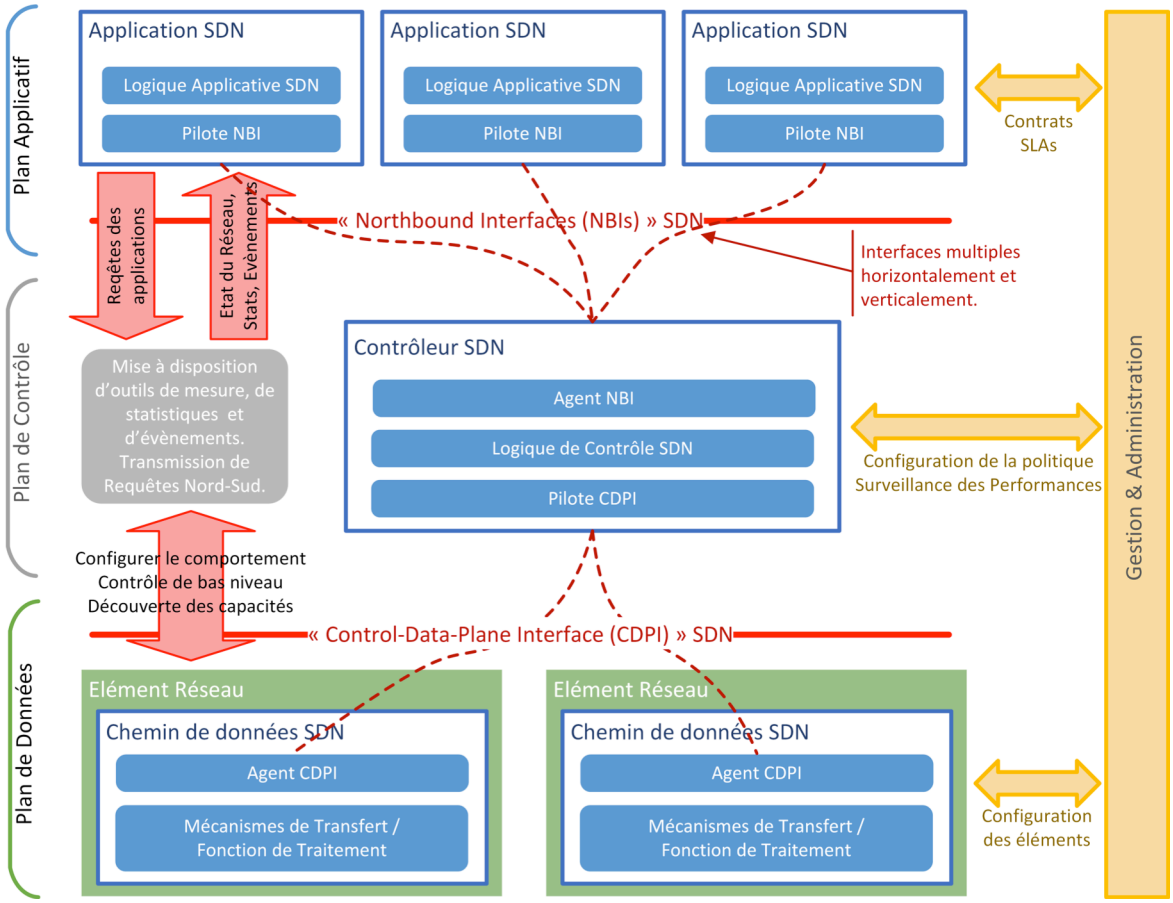


Figure C.2.1 – Composants d'une architecture SDN vue par l'ONF [ONF 2013b].

C.3 Protocoles et Standards pour l'Interface Sud

Si les SDN sont relativement récents, la configuration des commutateurs à distance existe depuis de nombreuses années et des protocoles existent. Le plus ancien est « Simple Network Management Protocol » (SNMP) [Case 1990] qui permet la configuration des équipements réseau depuis un poste de gestion. Néanmoins, la gestion n'est pas entièrement déportée puisque les équipements utilisent un agent interprétant les requêtes envoyées par le poste de gestion et qui va les répercuter sur l'équipement. Actuellement, SNMP est plus utilisé pour faire de la surveillance de réseau que de la configuration. Plus récemment, « Network Configuration Protocol » (NETCONF) [Enns 2011] permet aux équipements de proposer une API pour faciliter leur configuration. L'objectif principal de NETCONF est de permettre une configuration automatique d'équipements réseau. Pour cela, NETCONF utilise sur le langage de modélisation des données YANG [Bjorklund 2010] pour représenter la configuration des éléments du réseau, leur état mais aussi pour établir ses requêtes. Malgré cela, ces deux protocoles ne sont pas adaptés aux SDN : ils ne proposent pas de réelle séparation entre plan de contrôle et plan de données et ils ne sont pas entièrement programmables ; l'introduction de nouvelles fonctionnalités nécessitant une intervention directement sur l'équipement. Néanmoins, de tels protocoles peuvent être nécessaires pour configurer les équipements ou pour assurer une compatibilité avec les réseaux traditionnels. Ils sont donc supportés par la plupart des composants d'une architecture SDN (commutateurs ou contrôleurs).

C.3.1 LE STANDARD FORCES

Dans le domaine des SDN, plusieurs efforts ont été faits au sein de de l'IETF. Le groupe de travail « Forwading and Control Element Separation » (ForCES) a défini une architecture SDN en séparant le plan de contrôle du plan de transmission (premier document en 2003 [Khosvari 2003]). Par rapport à la vision OpenFlow des SDN, ForCES considère toujours le contrôle et l'acheminement au sein d'une même entité : le ForCES Network Element (NE) représenté sur la Figure C.3.1. ForCES définit deux nouvelles entités : le Forward Element (FE) et le Control Element (CE) [Yang 2004]. Le premier est chargé de l'acheminement des paquets et le second s'occupe des fonctions de contrôle et de signalisation. La communication entre les deux se fait via le protocole ForCES [Doria 2010] suivant un modèle maître-esclave (FE-CE). La Figure C.3.1 représente ces deux entités et situe le protocole ForCES. On peut aussi voir des exemples de fonctions qui peuvent être présentes dans ces deux entités. Contrairement à OpenFlow qui ne pose pas de contraintes sur le placement physique du contrôleur (hormis l'impact sur les performances), une architecture ForCES dans sa version de base spécifie que les différents éléments doivent être à un « hop » de distance maximum ; soit être sur le même réseau Ethernet par exemple. Néanmoins, nous verrons par la suite qu'il est envisageable d'augmenter cette distance en se reposant sur un protocole de Transport fiable.

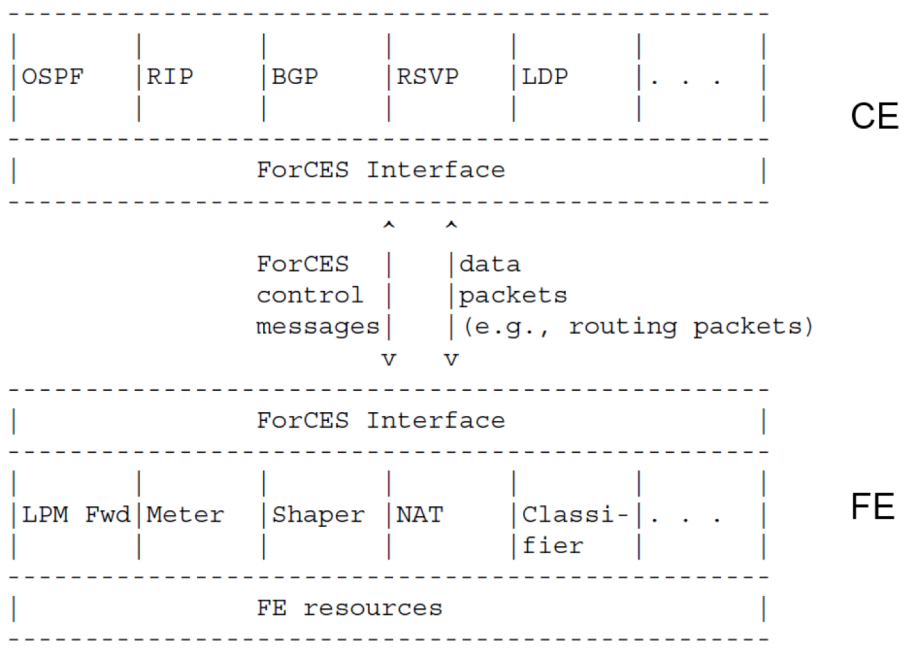


Figure C.3.1 – Exemples de fonctions présentes dans un CE et dans un FE (source : [Yang 2004]).

La Figure C.3.2 représente l'architecture ForCES avec un NE contenant plusieurs CE et FE. En plus de cela des gestionnaires « CE Manager » et « FE Manager » sont connectés respectivement aux CE et aux FE. Ces éléments non-obligatoires peuvent être utilisés pendant la phase de pré-association pour déterminer les différents NE ainsi que les CE et FE qui les constituent [Yang 2004]. Afin de stocker les différentes associations entre CE et FE, une base d'information « Management Information Base » a été définie dans [Haas 2010].

Cette base est stockée dans un unique CE si celui-ci est seul dans son NE, sinon elle peut être partagée entre le CE.
Si de nombreux liens de communication sont définis sur la Figure C.3.2, seul Fp est régi par le protocole ForCES. Les autres liens ne sont pas définis dans le document [Yang 2004] mais des indications sont données. Les communications Fi entre FE correspondent à des échanges entre commutateurs, que ceux-ci soient physiquement ou virtuellement séparés. La présence de plusieurs CE dans un NE peut se justifier par une volonté de fiabilité et le lien Fr représente ces communications. Il est à noter qu'un standard a été proposé permettant l'utilisation d'un CE de secours en cas de panne du principal à l'intérieur d'un NE [Ogawa 2014]. Enfin, les échanges entre les gestionnaires et le NE (notés Fl, Fc et Ff) sont relatifs à la configuration et doivent, pour le bon fonctionnement du système, reposer sur des protocoles fiables et sécurisés.
Les liens symbolisés par Fp sur la Figure C.3.2 permettent la communication entre les FE et les CE en se basant sur ForCES. Néanmoins, le protocole standardisé dans [Doria 2010] ne spécifie pas comment sont transportés les messages et préfère s'en abstraire en définissant une couche « Transport Mapping Layer » (TML) chargée de transporter les requêtes ForCES. Les besoins de ForCES pour cette TLM sont détaillés :

- fiabilité,
- sécurité,
- contrôle de la congestion,
- disponibilité des liens FE-CE,
- différentiation (8 niveaux de QoS souhaités),
- uni/multi/broadcast.

La manière d'obtenir ces besoins n'est pas spécifiée et peut faire intervenir des solutions de différents niveaux. Une implémentation de cette TML a été standardisée dans [Hadi 2010]. Les méthodes utilisées pour répondre aux différentes exigences de [Doria 2010] y sont détaillées. La TLM proposée est basée sur le protocole « Stream Control Transmission Protocol » (SCTP) [Stewart 2007]. SCTP est un protocole de Transport multi-domicilié

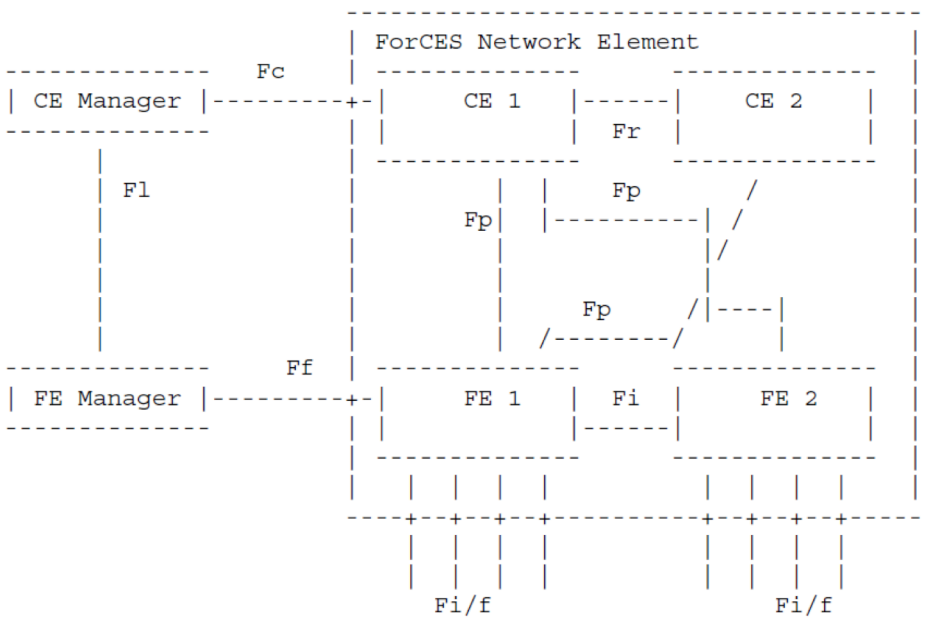


Figure C.3.2 – Architecture ForCES avec les principaux composants [Yang 2004].

proposant aux applications des services de fiabilité, contrôle de congestion, ... Il est donc possible d'exploiter directement de ces fonctionnalités pour transporter les requêtes ForCES.

Un composant essentiel de l'architecture ForCES est le « Logical Block Function » (LFB) défini par les standards sur la modélisation du FE [Halpern 2010] et la spécification du protocole ForCES [Doria 2010]. Il s'agit d'un block fonctionnel situé dans les FEs et commandé par les CEs. Un LFB peut être situé directement sur le chemin des paquets dans le FE et participer ainsi au traitement des paquets ou peut être un élément de contrôle et de configuration commandé par le CE. La Figure C.3.4 décrit un exemple de deux LFB présents sur le chemin de traitement des paquets. Comme pour les tables de flux d'OpenFlow, les blocs sont chainés afin d'obtenir la meilleure granularité possible. Les variables P et M représentent les paquets et leurs métadonnées associées entrants et sortants des blocs.
Le standard [Halpern 2010] introduit la spécification d'un modèle du FE, permettant ainsi de communiquer facilement ses capacités, son état et sa configuration à un CE via le protocole ForCES. Plusieurs LFB de même type peuvent être présentes dans le FE, ce document définit aussi des notions de classes de LFB. Il est ainsi plus facile pour le CE d'agir sur un ensemble

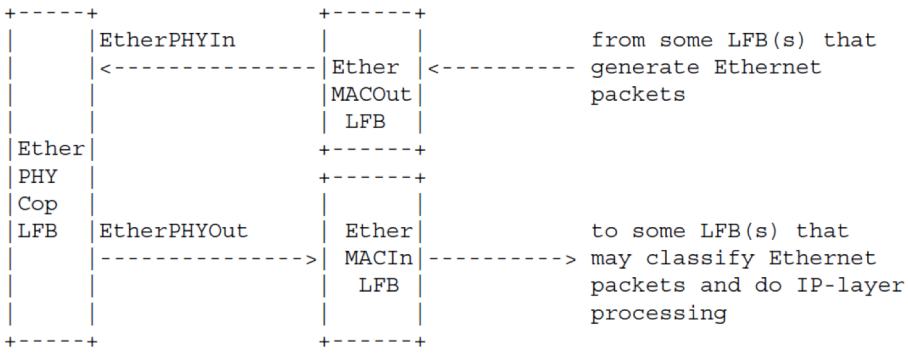


Figure C.3.3 – Exemple d'utilisation de LFB (source : [Wang 2013a]).

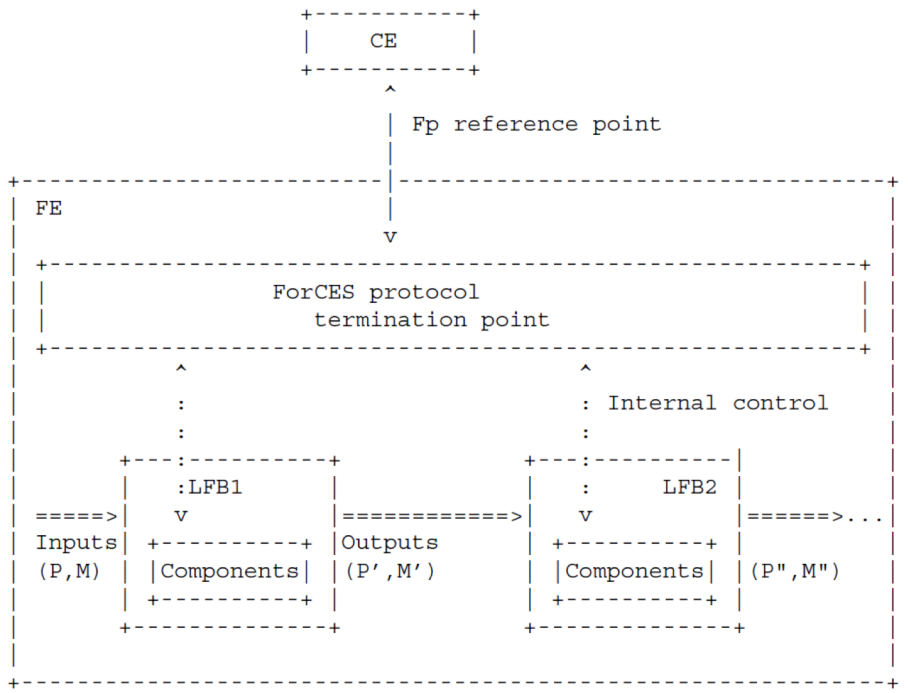


Figure C.3.4 – Exemple générique des blocs fonctionnels (LFB) de ForCES [Halpern 2010].

de LFB. Les classes permettant la configuration du FE sont notamment définies. Des spécifications plus précises sont apportées sous forme de librairie par le standard [Wang 2013a] avec la description de nouvelles classes ainsi que leur traduction en XML. Les différentes classes sont groupées suivant leurs rôles, correspondant à des fonctions basiques de routeurs :

- Traitement des paquets Ethernet : encapsulation, désencapsulation, entrée/sortie du paquet…
- Validation des paquets IP : IPv4 et IPv6.
- Transmission des paquets IP : sélection du saut suivant ou vérification de l'Unicast (IPv4 et IPv6).
- Redirection de paquets entre CE et FE : envoi vers le CE et injection dans le FE.
- Abstraction de traitements sur le paquet : redirection en fonction des métadonnées et ordonnanceur générique.

Un exemple d'utilisation de ces différentes classes est illustré dans la Figure C.3.3. Les paquets des réseaux extérieurs entrent par la gauche dans le LFB « EtherPHYCop ». Après un traitement au niveau physique, ils sont transmis à LFB « EtherMACIn » qui fait les traitements de niveau 2. Les paquets nécessitant plus de traitement sont ensuite transmis aux LFB suivantes (effectuant des traitements de niveau IP par exemple). Une LFB souhaitant envoyer un paquet sur le réseau va le transmettre à la LFB « EtherMACOut » qui fera les traitements nécessaires de niveau 2 et le transmettra à la couche physique via la LFB « EtherPHYCop ». Deux autres exemples plus détaillés sont donnés dans [Wang 2013a] avec la transmission des paquets IP et le traitement des requêtes ARP.

Les efforts de l'IETF concernant les SDN proviennent essentiellement du groupe de travail ForCES. En 2011, un groupe de discussion informel a été formé mais aucun document ne semble avoir été produit. Seules les problématiques classiques des SDN ont été évoquées. Plus récemment, un document à caractère informatif a été publié [Boucadair 2014] et veut donner la vision des SDN pour les fournisseurs de services. Plus précisément, les auteurs essayent de fournir une taxonomie fonctionnelle des techniques utilisées dans les SDN.

C.3.2 LE PROTOCOLE OPENFLOW

OpenFlow se revendique comme le premier standard définissant l'interface de communication entre les plans de contrôle et de données d'une architecture SDN. Il définit les éléments du commutateur ainsi que le protocole permettant de gérer celui-ci depuis un contrôleur à distance [ONF 2013c]. Les différents composants (Figure C.3.5) d'un commutateur OpenFlow sont :

- Les « flow tables » et la « group table » chargées du traitement des paquets,
- Le « OpenFlow Channel », interface permettant la connexion avec le contrôleur.

Le protocole OpenFlow est utilisé par le contrôleur pour ajouter, mettre à jour ou supprimer des entrées dans les tables, de manière réactive (c.à.d. en réponse à un paquet) ou de manière proactive. En définissant une succession de tables, il est possible de réaliser plusieurs traitements successifs sur un paquet avant de l'éjecter ou le diriger vers un port de sortie, ce « pipeline » est illustré par la Figure C.3.6. Il est intéressant de noter que le standard définit deux catégories de commutateurs compatibles avec OpenFlow : ceux proposant uniquement OpenFlow et les hybrides. Dans le premier cas, tous les paquets doivent suivre le traitement d'OpenFlow par tables. Dans le cas hybride, le commutateur supporte OpenFlow et est capable de traiter les paquets de manière traditionnelle avec par exemple de la commutation

Ethernet ou du routage de niveau 3. Ces commutateurs doivent alors posséder un mécanisme de classification qui va diriger les paquets soit vers le traitement par les tables d'OpenFlow soit vers le traitement classique. En plus de cela, un commutateur hybride peut aussi permettre à un paquet d'aller du traitement OpenFlow vers le traitement classique.

Afin d'obtenir un traitement le plus fin et efficace possible, les tables constituant le « pipeline » d'OpenFlow peuvent être de différents types, chacune ayant un rôle distinct. Les tables de flux sont les tables de base permettant la transmission des données. Elles contiennent une collection d'entrées constituées notamment d'un champ de correspondance et d'instructions associées (Tableau C-1). Un paquet arrivant dans un commutateur est comparé aux différentes entrées de la première table du « pipeline ». Si une seule entrées correspond, alors l'instruction associée est éxécutée. Si plusieurs entrées correspondent, celle avec la plus haute priorité est choisie. Dans le cas où aucune entrée ne correspond, l'entrée par défaut de la table décide par exemple de supprimer le paquet, le remonter au contrôleur ou le faire suivre à la table suivante. Les instructions associées à chaque entrée contiennent soit des actions à exécuter sur le paquet soit la table à parcourir par la suite. Dans ce cas-là, des métadonnées sont associées au paquet afin d'assurer la communication entre les tables. Si l'instruction ne spécifie pas de table suivante, les actions sont exécutées. Ces actions peuvent correspondre, par exemple, à la suppression du paquet, à une opération d'encapsulation/désencapsulation sur le paquet, ou son renvoi vers la file d'attente d'un port

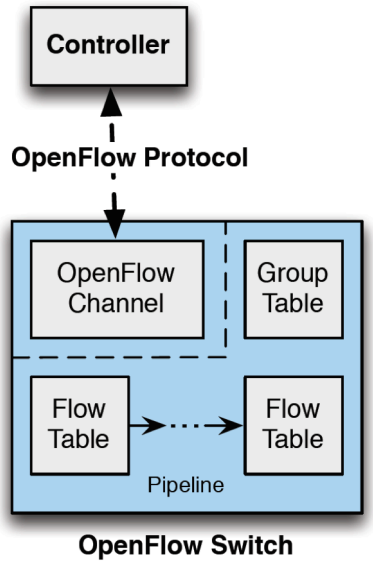


Figure C.3.5 – Composants d'un commutateur OpenFlow [ONF 2013c]

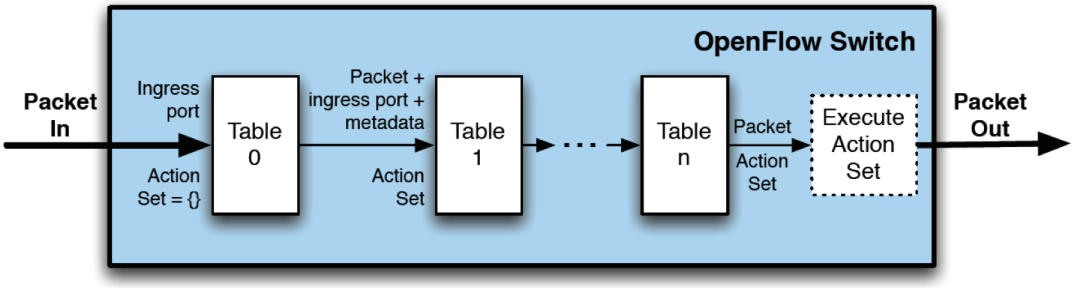


Figure C.3.6 – Pipeline de traitement d'un paquet dans un commutateur OpenFlow [ONF 2013c].

Match Fields	*Priority*	*Counters*	*Instructions*	*Timeouts*	*Cookie*

Tableau C-1 – Composants d'une entrée de table de flux dans un commutateur OpenFlow.

Group Identifier	*Group Type*	*Counters*	*Action Buckets*

Tableau C-2 – Composants d'une entrée de la table de groupe d'un commutateur OpenFlow.

Meter Identifier	*Meter Bands*		*Counters*
Band Type	*Rate*	*Counters*	*Type specific arguments*

Tableau C-3 – Composants d'une entrée de la table de mesure (en haut) et bande de mesure qui la compose (en bas).

de sortie, etc. Il est à noter qu'OpenFlow peut supporter plusieurs files d'attente par port mais qu'il ne les configure pas, ceci devant être fait par une solution extérieure (CLI, protocole propriétaire, OF-Config, etc.).

La « group table » (Tableau C-2) est une table contenant des entrées de groupes vers laquelle peuvent être dirigés les paquets depuis une table de flux classique. Une suite d'actions est alors exécutée sur le paquet. L'utilisation de ces groupes permet de définir des actions à exécuter sur de multiples flux.

Une instruction introduite par la version 1.3 d'OpenFlow permet de diriger le paquet vers une table de mesure (Tableau C-3) Les métriques qui y figurent peuvent être appelées par les entrées des tables de flux avant que les actions ne soient traitées. Pour une métrique donnée, plusieurs tables de bande sont disponibles et permettent de comparer le débit du flux à une valeur donnée par la table. Si celle-ci est dépassée, le paquet peut être supprimé ou le champ DSCP de l'entête IP peut être modifié. En se servant de ces métriques par flux, OpenFlow est donc capable d'opérations de QdS basiques comme la limitation du débit et cela en amont des files d'attentes qui peuvent être définies sur les ports de sortie.

Dans chaque table d'OpenFlow, un champ « Counters » permet de mémoriser des statistiques sur le/les flux de données qui la traverse et sur la table en elle-même. Par exemple, il est possible de stocker la durée écoulée depuis la définition d'une entrée, le nombre de paquets ayant transité par la table ou pour les ports des informations plus spécifiques comme le taux d'erreurs. Ces statistiques peuvent être récupérées par le contrôleur via l'envoi d'une requête au commutateur et permettent ainsi au contrôleur de se constituer une vision précise des flux circulant sur le réseau.

L'élément « OpenFlow Channel » de la Figure C.3.5 est l'interface de communication avec le contrôleur. Cette communication se fait via le protocole OpenFlow de manière sécurisée avec Transport Layer Security (TLS) ou en clair avec TCP. Il existe 3 types de messages, chacun comportant des sous-types :

- « Controller-to-switch » : envoyés par le contrôleur pour gérer et vérifier l'état du commutateur ou par exemple acquérir des statistiques,

- « Asynchronous » : envoyés par le commutateur pour informer le contrôleur d'évènements réseaux et de changements d'état,
- « Symmetric » : envoyés par le contrôleur ou le commutateur pour initier la connexion entre les deux éléments et la maintenir active.

Les deux premiers types de messages sont utilisés par exemple pour définir les entrées des différentes tables ou faire remonter/descendre les paquets de nouveaux flux présents dans aucune entrée. Des formats recommandés pour ces messages sont définis dans [ONF 2013c].
OpenFlow ayant subi de nombreuses évolutions, le choix de la version à utiliser lors d'un déploiement est important. Le Tableau C.4 résume certains apports des versions récentes (1.0 à 1.4), le détail des modifications étant inclus dans la spécification du protocole OpenFlow (Annexe B de [ONF 2013c]). Au vu des apports et modifications, il est clair que des versions différentes ne peuvent pas être compatibles. Nous verrons par la suite que l'ONF recommande la vérification des fonctionnalités de la version d'OpenFlow envisagée lors d'une migration. Par exemple, un réseau IPv6 nécessite obligatoirement le déploiement de la version 1.2 voire 1.3 pour que le support d'IPv6 soit complet.

Tableau C-4 - Apports et modifications apportées par les versions récentes d'OpenFlow.

Version	Date	Apports / Modifications
1.0 0x01	31/12/2009	• « Slicing » : support de plusieurs files d'attente par port • Correspondance avec les bits ToS/DSCP d'un paquet IP • Récupération des statistiques pour des ports individuels
1.1 0x02	28/02/2011	• « Multiple Tables » : détection de toutes les tables par le contrôleur, possibilité de détecter les tables Hardware • « Groups » : Introduction des groupes pour définir un ensemble de ports • Support avancé des VLAN et de MPLS • Support de ports virtuels (Ex : pour des tunnels)
1.2 0x03	5/12/2011	• Extension des champs de correspondance, plus de flexibilité • Support d'IPv6 • Support de plusieurs contrôleurs, mécanisme de choix en cas de panne
1.3 0x04	13/04/2012	• Refonte de l'expression des capacités d'un commutateur • Ajout de l'entrée table manquante et amélioration du comportement • Support des entêtes étendus d'IPv6 • Ajout des mesures par flux (tables de mesures)
1.3.1	6/09/2012	• Amélioration de la négociation de version
1.4 0x05	08/2013	• Mise à jour structures pour les ports, tables, files, … au format TLV • Support de ports Optiques (configuration, surveillance, description) • Ajout de la surveillance de flux pour un contrôleur (Ex : détecter des modifications par un autre contrôleur) • Amélioration du comportement en cas de tables pleines (et prévention)

Lors de la description des différents groupes de l'ONF, nous avons évoqué OF-Config [ONF 2014]. Il s'agit d'un protocole permettant la configuration et la gestion d'un « contexte » OpenFlow. Le contexte OpenFlow tel que défini par OF-Config regroupe l'ensemble des éléments OpenFlow communiquant entre eux ainsi que les communications OpenFlow établies entre ces éléments. Les mécanismes permettant de définir ce contexte (distribution des adresses IP entre routeurs et contrôleurs, configuration des capacités des commutateurs, …) ne sont pas inclus dans le protocole OpenFlow qui se concentre uniquement sur les communications entre le contrôleur et le commutateur, assumant que toute configuration préalable a été effectuée. Afin de ne pas avoir à effectuer ces actions « à la main » sur chaque élément du contexte OpenFlow (avec SNMP ou NETCONF par exemple), le protocole « OpenFlow Configuration » (OF-Config) [ONF 2014] a été introduit et permet de centraliser les configurations à l'aide d'un service appelé « OpenFlow Configuration Point ». Ce service peut être utilisé en amont de l'utilisation du contexte OpenFlow mais aussi pendant son utilisation pour reconfigurer un élément. Il est à noter qu'il est bien défini comme un service et non comme une entité physique, rendant son placement et son utilisation totalement libres. Si l'on souhaitait placer l'utilisation d'OpenFlow et OF-Config dans un modèle de rôle, OF-Config serait utilisé par l'acteur situé le plus bas qui définit le réseau et OpenFlow serait utilisé par un des acteurs supérieurs qui exploite le réseau. Afin d'établir sa vision globale du contexte OpenFlow, OF-Config définit deux entités:

- « OpenFlow Logical Switch »: abstraction d'un commutateur OpenFlow,
- « OpenFlow Capable Switch » : contexte opératoire correspondant à un ou plusieurs commutateurs OpenFlow. Equivalent à un élément physique ou virtuel qui héberge un ou plusieurs « OpenFlow Logical Switchs ».

La Figure C.3.7 présente un contexte OpenFlow avec d'un côté le point de configuration utilisant OF-Config et de l'autre les contrôleurs utilisant OpenFlow. Entre les deux, le commutateur OpenFlow est bien vu comme une abstraction, ce qui permet d'ignorer l'architecture exacte des éléments (physique, virtuelle, etc.). Les principales fonctions offertes par OF-Config pour la configuration d'un commutateur OpenFlow 1.3 sont :

- Affectation d'un ou plusieurs contrôleurs OpenFlow aux équipements OpenFlow du plan de données,
- Configuration des files d'attentes (Ex : débit max) et des ports du commutateur (Ex : sens),
- Configuration des certificats pour la communication sécurisée entre contrôleurs OpenFlow et commutateurs OpenFlow,
- Découverte des capacités d'un commutateur OpenFlow,
- Configuration de tunnels de types spécifiques : IP-in-GRE, NV-GRE, VxLAN.

Il est précisé dans la spécification d'OF-Config [ONF 2014] que celui-ci supporte OpenFlow 1.3 et les versions précédentes (de 1.0 à 1.2) mais rien n'est précisé pour le support d'OpenFlow 1.4. Néanmoins, celle-ci étant relativement récente, il est vraisemblable qu'OpenFlow 1.4 sera supporté dans les futures versions d'OF-Config. Il est à noter que la communication entre le point de configuration et les commutateurs se fait via le protocole NETCONF [Enns 2011] et le modèle de données utilisé est donc YANG [Bjorklund 2010]. Les détails sont donnés dans [ONF 2014].

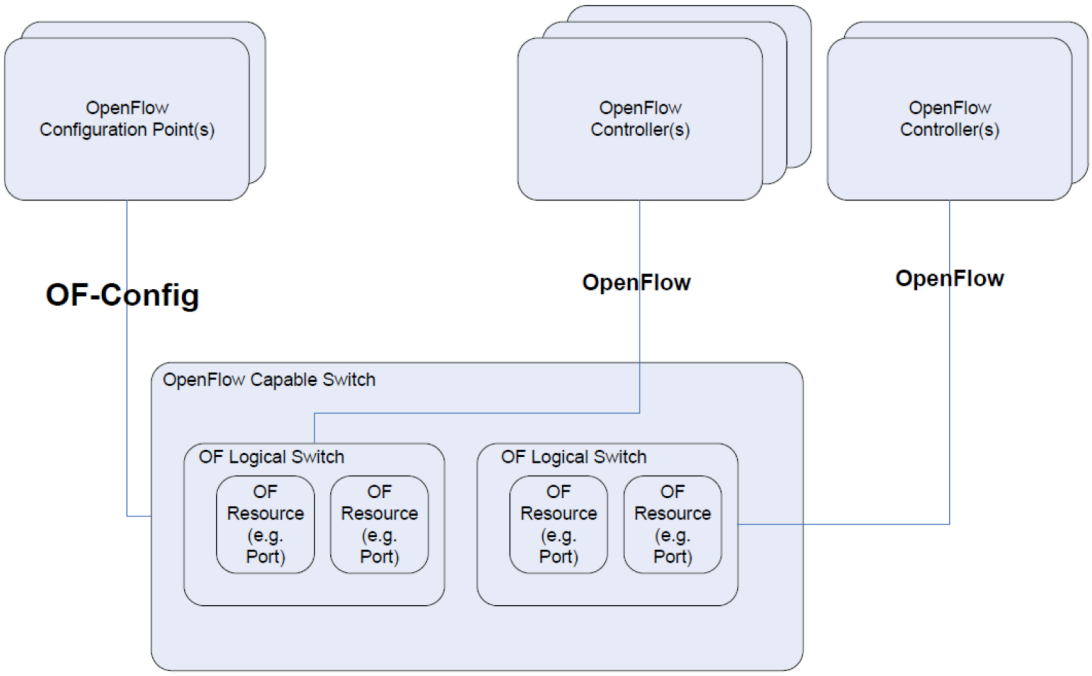


Figure C.3.7 – Entités OpenFlow et échanges tels que définis par OF-Config (source [ONF 2014].

C.4 L'INTERFACE NORD

Dans la section précédente, deux approches de SDN ont été détaillées. Néanmoins, les définitions faites pour ForCES et OpenFlow s'intéressent uniquement à l'interface Sud. Nous allons essayer ici de faire un point sur l'état actuel de l'interface Nord du contrôleur dans les architectures SDN basées sur OpenFlow, celui-ci étant le plus répandu. Néanmoins, les approches devraient être les mêmes quel que soit l'architecture SDN et idéalement, les interfaces devraient être compatibles.

Si le contrôleur a la visibilité du réseau entier, les applications qui lui sont connectées sont la réelle intelligence du réseau en proposant des fonctions de gestion avancée, de sécurité et de surveillance du réseau. Les API définies au niveau de l'interface Nord permettent d'offrir une abstraction des couches inférieures et simplifient ainsi la vision des applications. Néanmoins, chaque type d'application nécessite des informations plus ou moins détaillées. Le groupe de travail « North Bound Interfaces » de l'ONF a été lancé en 2013 et tente d'éclaircir les travaux effectués. La Figure C.4.1 présente leur vision des différents types d'interafce (avec une indication sur le niveau d'abstraction dans lesquels ils se placent). Le niveau d'abstraction le plus faible correspond à une programmation quasiment directe du commutateur, le contrôleur ne servant alors que d'intermédiaire. A l'opposé, le plus haut niveau d'abstraction revient à fournir des services purs comme le fait par exemple OpenStack. Pour l'instant, deux approches semblent prendre de l'importance : l'architecture MD-SAL d'OpenDaylight et le langage de haut-niveau Frenetic (ou Pyretic pour la version Python).

L'abstraction de la couche service avec une approche modélisation (MD-SAL) d'OpenDaylight se base sur une modélisation des données avec YANG, lui-même basé sur XML. Il est ainsi possible d'obtenir des données accompagnées de leur description ce qui facilite la tâche du contrôleur et des applications. Une architecture REST permet l'accès aux machines distantes.

Frenetic [Foster 2013] est un langage haut-niveau qui vise à simplifier la programmation d'un SDN. En partant du constat que la gestion d'un contrôleur et la configuration d'un SDN nécessitent des connaissances poussées sur les flux mais aussi sur le matériel présent, Frenetic a été proposé pour abstraire totalement le protocole utilisé sur l'interface Sud.

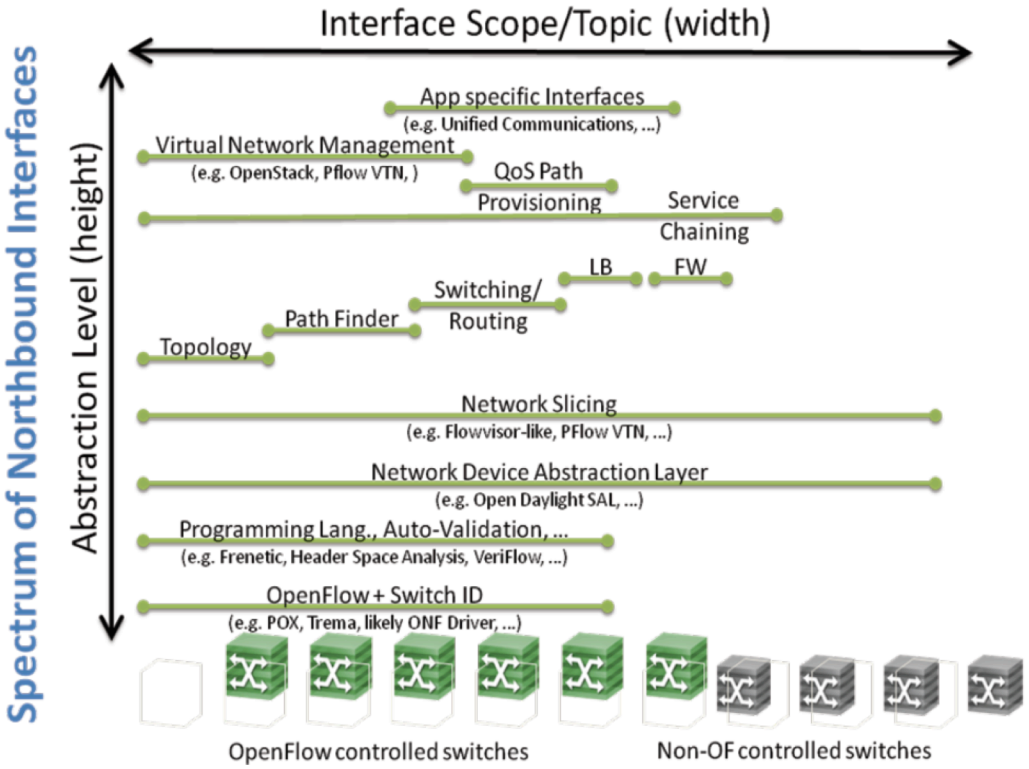


Figure C.4.1 – Aperçu des interfaces Nord avec leur niveau d'abstraction (source : [ONF 2013g]).

Frenetic/Pyretic permettent aux applications de se concentrer uniquement sur leur domaine d'action : la définition globale des différentes politiques du réseau que ce soit en termes de connectivité, de sécurité ou de surveillance.

C.5 TECHNOLOGIES SDN

Dans les sections précédentes, nous avons vu deux protocoles permettant le déploiement d'une architecture SDN : OpenFlow et ForCES. Ces dernières années, OpenFlow n'a cessé de gagner en popularité et semble devenir le protocole de référence pour la communication entre le plan de contrôle et le plan de données. Les matériels et logiciels que nous allons lister dans cette section se basent donc sur OpenFlow pour proposer une solution de SDN. Les informations présentes dans les tableaux sont en partie reprises de [Nunes 2013] et complétées ou mises à jour par nos soins.

C.5.1 COMMUTATEURS MATERIELS OPENFLOW

Le fort engouement pour OpenFlow a poussé les industriels à inclure des agents OpenFlow dans leurs commutateurs. Des modèles compatibles avec OpenFlow sont listés dans le Tableau C-5. La version d'OpenFlow supportée est donnée à titre indicatif, les agents OpenFlow présents dans les commutateurs pouvant être mis-à-jour en même temps que le logiciel embarqué du commutateur. En toute logique, les fabricants devraient donc proposer des mises à niveau assurant la compatibilité de leur matériel avec les dernières versions d'OpenFlow.

En plus des logiciels embarqués (ou « firmwares ») proposés par les fabricants, il est aussi possible de flasher le commutateur et d'y incorporer un commutateur logiciel. Un tutoriel[1] propose par exemple de flasher un commutateur Pronto et d'y mettre Indigo qui est un commutateur logiciel. Ainsi, il est possible d'ajouter des fonctionnalités dans les commutateurs sans avoir à les changer.

Les commutateurs constituent un point essentiel lors du déploiement d'une plateforme SDN, il est intéressant de regarder la compatibilité avec d'autres méthodes de programmation comme SNMP ou NETCONF. En effet, la compatibilité avec de tels protocoles peut faciliter l'intégration dans une plateforme réseau déjà en place et qui utiliserait déjà des mécanismes de programmation des commutateurs.

C.5.2 COMMUTATEURS LOGICIELS OPENFLOW

En plus des commutateurs matériels, il existe des commutateurs logiciels qui ne sont pas liés à un équipement particulier ou à un fabricant. Ces logiciels sont prévus pour tourner soit comme une application dans l'espace utilisateur sur une machine classique soit directement comme logiciel embarqué sur des commutateurs matériels.

Une liste de commutateurs logiciels est donnée dans le Tableau C-5 avec les versions d'OpenFlow supportées. Ces dernières sont données à titre indicatif, les logiciels pouvant évoluer et devenir compatibles avec des versions plus récentes d'OpenFlow.

Néanmoins, les commutateurs logiciels ne se limitent pas à cette liste, les commutateurs open-source sont parfois liés à des implémentations propriétaires. L'entreprise peut alors être un contributeur majeur de la version open-source. Par exemple, le commutateur logiciel

[1] "Up and Running on the 3290 or 3780"
http://www.openflowhub.org/display/Indigo/Up+and+Running+on+the+3290+or+3780

Tableau C-5– Liste non exhaustive de commutateurs logiciels compatibles OpenFlow.

Software Switch	Implementation	Aperçu	OF Version
OpenvSwitch	C / Python	Open source software Can be port to ASICs	1.0
Pantou/OpenWRT	C	Firmware that can be installed on commercial swtiches	1.0
Ofsoftswitch13	C / C++	User-space software switch implementation	1.3
Indigo	C	Disponible sur Broadcom et Linux	1.0
NetFPGA	FPGA-based Approach		0.8.9 ?
Click Modular Router	C++	SDN via l'OpenFlow Element Click	1.0
XDPD	C / C++	BISDN	1.3
LINC (Rhinoceros)	Erlang	RasperyPi	1.?

indigo contient les bases du commutateur « Switch Light » développé par « Big Switch networks[2]».

En plus du commutateur logiciel, certaines implémentations présentées dans le Tableau C-6 proposent des outils permettant d'obtenir des informations supplémentaires. Par exemple, Open vSwitch (OVS) comporte une base de données « Open vSwitch DataBase (OVSDB) » pour gérer et configurer le commutateur et qui peut être manipulée à distance via un protocole de gestion pour OVSDB (voir [Pfaff 2013]). En plus de la configuration, la base de données contient des statistiques sur le système, ce qui peut être intéressant pour évaluer une plateforme. Nous reviendrons par la suite sur les performances d'Open vSwitch (OVS) puisque celui-ci est utilisé dans certains émulateurs de SDN.

C.5.3 CONTROLEURS OPENFLOW

[2] Big Switch networks - http://www.bigswitch.com/

Le contrôleur est le composant essentiel d'une architecture SDN. En plus de contenir la partie « intelligente » du réseau (le contrôle), il doit être capable d'interagir avec les applications des couches supérieures et de leur offrir une vision centralisée du réseau sous-jacent. De nombreux contrôleurs pour OpenFlow ont été développés (voir Tableau C-7), la plupart sont des logiciels open-source.
Comme pour les commutateurs logiciels, des versions propriétaires de ces contrôleurs libres existent. Par exemple, OpenDaylight sert de base au contrôleur XNC de Cisco et il est aussi présent dans le « SDN Virtual Environment » d'IBM. FloodLight est la base du contrôleur utilisé par Big Switch Networks qui, en retour, participe activement au développement de la version open-source.
Si la plupart de ces contrôleurs se contentent uniquement de fonctionnalités basiques leur permettant de programmer les commutateurs, certains proposent des fonctionnalités supplémentaires.
FlowVisor est un contrôleur OpenFlow particulier qui permet de virtualiser le réseau partitionnant ses ressources en « *slices* ». Chaque *slice* est dédié à un réseau openflow virtuel différent et peut être gérée par un contrôleur différent. FlowVisor agit alors comme un proxy entre des commutateurs et plusieurs contrôleurs OpenFlow. Cette approche permet notamment la mise en place d'un réseau expérimental dans un réseau réel et a été utilisée dans le réseau de Stanford (cas d'étude présenté par la suite). Afin de définir les différentes les *slices* associés à chaque réseau virtuel, FlowVisor se base sur des informations des 4 couches basses du modèle OSI, par exemple : le port du commutateur, les adresses Ethernet, les adresses IP, les ports TCP ou UDP… Théoriquement, il est même possible de hiérarchiser les contrôleurs FlowVisor et définir ainsi des tranches à l'intérieur de tranches. Les détails d'implémentation et d'expérimentation sont dans le rapport technique [Sherwood 2009] ainsi que dans l'article [Sherwood 2010]. Les analyses effectuées montrent entre autre que FlowVisor peut permettre l'isolation de certains flux, qu'il introduit une légère latence supplémentaire dans l'établissement des nouveaux flux mais qu'il ne semble pas trop souffrir du passage à l'échelle.

Tableau C-6 – Liste non exhaustive de Commutateurs matériels compatibles OpenFlow.

Fabricant	Modèle	OF version
Hewlett-Packard	8200zl, 6600, 6200zl, 5400zl and 3500/3500yl	1.0
Brocade	NetIron CES 2000 Series	1.0
IBM	RackSwitch G8264	1.0
NEC	PF5240, PF5820	1.0
Pronto	3290 et 3780	1.0
Juniper	Junos MX-Series	1.0
Pica8	P-3290, P-3295, P-3780 et P-3920	1.3
Dell / Force 10	S4810, Z9000	1.3

RouteFlow [Nascimento 2011] n'est pas à proprement parler un contrôleur OpenFlow mais plutôt une solution complète permettant de virtualiser la fonction de routage IP avec l'utilisation de commutateurs OpenFlow. Pour cela, RouteFlow comporte un serveur connecté à des contrôleurs OpenFlow. Les contrôleurs lui remontent l'état des commutateurs OpenFlow rencontrés avec notamment des informations sur les ports connectés à d'autres commutateurs OpenFlow. En se basant sur ces informations, le serveur RouteFlow peut établir une topologie du réseau plus ou moins abstraite suivant le mode de fonctionnement choisi. Cette topologie est utilisée pour faire tourner des algorithmes de routage qui servent à déterminer les tables de flux des commutateurs.

Plus qu'un simple contrôleur, OpenDaylight[3] est un projet open-source concentré autour d'un contrôleur modulaire. Des nombreuses interfaces sont proposées, que ce soit côté nord (OSGi, REST, etc.) ou bien côté sud (OpenFlow, SNMP, NETCONF, …). Trois éditions d'OpenDaylight sont disponibles, le choix dépend du contexte (applications et réseau physique). L'édition basique comprend les fonctionnalités minimales permettant la mise en place du contrôleur. L'édition Virtualisation est orientée centre de données et contient pour cela des services facilitant l'intégration dans OpenStack via le module Neutron. Enfin, l'édition restante vise les fournisseurs de services et propose pour cela plus de choix au niveau des interfaces Sud avec, par exemple, la compatibilité avec le trafic BGP.

OpenDaylight n'est pas le seul contrôleur du Tableau C-7 à supporter d'autres mécanismes qu'OpenFlow pour programmer les commutateurs. Par exemple, Ryu supporte SNMP, NetConf, xFlow... De tels contrôleurs peuvent donc s'intégrer plus facilement dans des plateformes existantes et commander des commutateurs non-OpenFlow.

Tableau C-7 – Liste non exhaustive des Contrôleurs OpenFlow.

Controleur	Implementation	Open-Source	Developpeur	OF version
POX	Python	Oui	Nicira	1.0
NOX	Python / C++	Oui	Nicira	1.0
MUL	C	Oui	Kulcloud	1.0 et 1.3
Maestro	Java	Oui	Rice University	?
Trema	Ruby / C	Oui	NEC	1.0
Beacon	Java	Oui	Stanford	?
Jaxon	Java	Oui	Indep Dev	?
Helios	C	Non	NEC	?
Floodlight	Java	Oui	BigSwitch	1.0
SNAC	C++	Non	Nicira	?
Ryu	Python	Oui	NTT, ORSG group	1.4
NodeFlow	Javascript	Oui	Indep Dev	1.0
ovs-controller	C	Oui	Indep Dev	?
Flowvisor	C	Oui	Stanford/Nicira	?
RouteFlow	C++	Oui	CPQD	?
OpenDaylight	Java	Oui	OpenDaylight Project	1.3

[3] Description technique d'OpenDaylight – avril 2014: http://www.opendaylight.org/project/technical-overview

C.5.4 OUTILS D'EMULATIONS ET DE SIMULATIONS DE RESEAUX SDN

Dans le cadre d'expérimentation sur les SDN, il est intéressant de posséder un outil permettant la simulation et/ou l'émulation. En effet, que ce soit pour la conception d'une architecture ou le développement d'un protocole, ces méthodes permettent de réaliser des premiers tests sans avoir à réaliser d'implémentation sur un système réel.

Mininet[4] est un émulateur de réseau écrit en Python et maintenu par ON.LAB. Présenté dans [Lantz 2010], il permet sur un seul noyau linux d'émuler un réseau complet :

- Nœuds : gérés par un groupe de processus limités en utilisation du CPU,
- Commutateurs : « Linux Bridge » (Pont) ou instance d'Open vSwitch,
- Liens : utilisation de « Traffic Control » (tc) pour forcer les caractéristiques (délai, débit, …).

Le principal avantage de Mininet est la possibilité de lancer sur chaque nœud des instances des applications présentes dans le système hôte. Les communications se font sur un lien Ethernet simulé avec une bande-passante et un délai donnés. L'avantage de l'émulation au sens où l'entend Mininet est la facilité de portage de code : les mêmes applications pouvant être exécutées sur un réseau Mininet et sur un réseau physique.
L'émulation de SDN est possible dans Mininet : les commutateurs Mininet supportent OpenFlow et il est possible d'y connecter un contrôleur OpenFlow. La version d'OpenFlow supportée est dépendante de la version actuelle d'Open vSwitch qui est utilisée par l'émulateur pour introduire les SDN dans les commutateurs créés. Par défaut, Mininet propose plusieurs contrôleurs : le contrôleur OpenFlow de référence, le contrôleur d'Open vSwitch et le contrôleur NOX. Il est aussi possible d'utiliser d'autres contrôleurs en les faisant tourner directement sur la machine hôte et en les connectant au réseau Mininet comme des contrôleurs distants. Au vu de ces capacités, Mininet parait donc être une solution intéressante pour tester le comportement d'applications sur un SDN, vérifier le comportement et la compatibilité d'un nouveau contrôleur, etc. En revanche, la facilité d'utilisation de Mininet devient son principal inconvénient : une émulation sur un système unique est limitée par les performances de l'hôte et les instances d'Open vSwitch deviennent rapidement gourmandes en ressources si la bande-passante simulée est élevée.

Network Simulator 3 (ns-3)[5] est un des simulateurs réseau le plus utilisé dans le monde. Il supporte la version 0.8.9 d'OpenFlow qui est relativement ancienne mais permet ainsi d'effectuer des simulations de SDN. Les contrôleurs implémentés dans ns-3 ne correspondent pas à des contrôleurs existants tels que POX ou NOX, il s'agit de contrôleurs permettant des fonctions basiques comme refuser tous les flux ou accepter tous les flux. Le simulateur faisant de nombreuses abstractions sur le réseau, un contrôleur implémenté dans ns-3 n'est pas assurée d'obtenir les mêmes performances dans le monde réel. Ceci a fortement freiné le développement de contrôleurs dans ns-3 et, en conséquence, a diminué l'intérêt pour la simulation d'OpenFlow avec ns-3. Le projet d'OpenFlow sur ns-3 est actuellement suspendu et il semble peu probable que de nouvelles versions soient développées dans un avenir proche.

[4] Mininet - http://www.mininet.org
[5] ns-3 - http://www.nsnam.org

« EstiNet OpenFlow Network Simulator and Emulator » [Wang 2013] est comme son nom l'indique un émulateur et un simulateur de réseaux utilisant OpenFlow. Dans le mode simulation, il est possible de faire tourner des contrôleurs tels que NOX, POX, Floodlight, OpenDaylight ou Ryu sur un nœud contrôleur simulé. Dans le mode émulation, il est possible d'utiliser un contrôleur physique pour contrôler à distance les commutateurs simulés par EstiNet. Développé par EstiNet Technologies[6], ce logiciel se revendique comme le meilleur émulateur et simulateur de réseau SDN avec comme atouts : la simulation de liens Ethernet et Wi-Fi, l'utilisation de vrais contrôleurs OpenFlow et de vraies applications, l'utilisation d'une vraie pile TCP/IP, etc. Un des points fort d'EstiNet est son moteur de simulation qui va directement utiliser l'horloge de la simulation pour commander les applications externes, les contrôleurs, et les commutateurs plutôt que de laisser leur contrôle au système d'exploitation comme le fait Mininet. Ainsi, les interactions entre composants sont censées mieux respecter les contraintes temporelles du réseau simulé. Les détails sur l'implémentation d'EstiNet sont donnés dans [Wang 2013] conjointement à une comparaison avec les caractéristiques de Mininet et de ns-3. Une analyse de performances montre qu'EstiNet serait moins gourmand en ressources que ne l'est Mininet et serait aussi plus précis dans ses résultats. En effet, nous allons voir que l'architecture même de Mininet le limite sur certains aspects.

La qualité des résultats obtenus lors d'expériences en émulation ou en simulation est fortement dépendante des modèles utilisés pour représenter le monde réel. Afin de simuler des commutateurs compatibles avec OpenFlow, la plupart d'émulateurs de SDN utilisent Open vSwitch (OVS). Si celui-ci est performant en tant que commutateur logiciel et peut en effet simuler un serveur faisant tourner une instance d'Open vSwitch, il est normal de se questionner sur sa capacité à simuler un commutateur matériel compatible avec OpenFlow. Cette question est étudiée dans [Huang 2013] et il est clair que les capacités de base de OVS et des commutateurs matériels sont différentes (voir Tableau C-8). Il est important de noter que les mesures sur OVS ont été faites en utilisant uniquement 2 ports, contexte qui lui est clairement avantageux. Afin d'aller plus loin, les auteurs tentent de simuler ces commutateurs matériels grâce à OVS. Les résultats de leurs travaux sont clairs : pour obtenir une simulation précise, il est nécessaire d'avoir des modèles exacts des commutateurs matériels que l'on souhaite simuler. Les performances d'OVS étant dépendantes du CPU, il est possible que OVS soit plus rapide que les commutateurs matériels dans certaines cas mais aussi qu'il soit plus lent si la charge devient importante (Ex : fort taux d'arrivées). Il est notamment souligné que le comportement d'un contrôleur OpenFlow (latences, taux d'acceptation de nouveaux flux) est dépendant du matériel et du logiciel embarqué et que ceux-ci varient d'un commutateur à un autre.

[6] EstiNet – http://www.estinet.com

Tableau C-8 - Comparaison des caractéristiques de 3 commutateurs physiques avec des mesures faites sur une version du commutateur logiciel Open vSwitch (source : [Huang 2013]).

Commutateur	Logiciel Embarqué	CPU	Lien	Entrées d'une table de flux	Taux de mise en place
HP ProCurve J9451A	K.15.06.5008	666MHz	1Gbps	1500	40 flux/s
Fulcrum Monaco Reference	Open vSwitch 1.0.1	792MHz	10Gbps	511	42 flux/s
Quanta LB4G	Indigo-1.0-web-1b4g-rc3	825MHz	1Gbps	1912	38 flux/s
Open vSwitch (Xeon X3210)	Version 1.7.0	2.13GHz	1Gbps	65000	408 flux/s

C.6 EXEMPLES D'APPLICATION DE L'APPROCHE « SOFWARE DEFINED NETWORKING»

Actuellement, les réseaux programmables ne sont plus un concept seulement théorique. Les protocoles déjà définis ainsi que les matériels disponibles permettent le déploiement de SDNs. L'annexe 0 résume le document [ONF 2013f] qui porte sur la migration des réseaux traditionnels vers les réseaux programmables avec OpenFlow. Les cas d'applications présentés dans ce document sont de bons exemples de déploiement des SDN dans un contexte de production.

C.6.1 RESEAU D'ACCES SUR UN CAMPUS

Le réseau d'accès d'un campus est soumis à de fortes variations au cours de la journée : le nombre d'utilisateurs, leurs points de connexion, le nombre de requêtés, etc. La flexibilité et l'autonomie sont des points clés permettant les performances d'un tel réseau. Une architecture de type SDN permet d'apporter ses caractéristiques tout en fournissant les moyens de mesurer les performances et la qualité d'expérience.
Le cas d'application du campus de Stanford a servi de preuve de concept pour les SDN et pour OpenFlow en particulier. Il est intéressant de noter que la transition vers un SDN a pu se faire en parallèle au réseau traditionnel et sans perturber son fonctionnement, ce qui est indispensable au déploiement des SDN.

C.6.2 RESEAU ETENDU (WAN) ENTRE PLUSIEURS SITES

Les WAN traditionnels manquent de dynamicité et sont fréquemment surdimensionnés. Il en découle une forte sous-utilisation moyenne (30-40%) qui implique un cout 2 à 3 fois plus élevé que leur coût réel en bande-passante. Dans un contexte où les différents sites reliés par le WAN sont maitrisés par une seule et unique entité, les applications présentes sur le réseau ainsi que leurs contraintes sont connues et maitrisées. Les SDN peuvent alors permettre d'obtenir une architecture plus souple basée sur un plan de contrôle programmable.

La vision globale du réseau ainsi que le contrôle complet du plan de données facilite le déploiement d'applications réseau dédiées à la surveillance et à la gestion du WAN. Dans le cas du WAN reliant les centres de données de Google, une application d'ingénierie du trafic a été développée. Cette application permet de contrôler le plan de données en se basant sur des mesures de performances, les connaissances des applications et les possibilités d'OpenFlow.

C.6.3 RESEAU DE FOURNISSEUR D'ACCES

En plus des connexions Internet classiques, les fournisseurs d'accès proposent des services avancés tels que le déploiement de WAN entre différents sites. Néanmoins, le déploiement d'un tel service nécessite fréquemment une intervention humaine chez le fournisseur d'accès que ce soit l'établissement d'un contrat ou pour la configuration d'équipement réseau. Il n'est pas alors possible d'avoir un système autonome et vraiment réactif. En simplifiant la gestion du plan de données par domaine et en facilitant l'intégration de logiciels, les SDN permettent d'apporter cette flexibilité.
Le fournisseur d'accès NTT se base sur OpenFlow pour la gestion de ses routeurs de bordure, ce qui lui permet de déporter les services BGP et de les centraliser dans une seule entité. La configuration de nouvelles sessions et la gestion du routage peuvent alors être faits pour tout le domaine géré par le fournisseur d'accès.

C.6.4 DEPLOIEMENT DE POLITIQUES DE QUALITE DE SERVICE

La récupération de statistiques sur les flux ainsi que l'utilisation des tables de mesures permettent le déploiement de règles de QdS via OpenFlow. Néanmoins, OpenFlow n'est uniquement capable de différencier les flux et de limiter leur bande-passante. Une application gérant les différentes politiques de QdS est alors nécessaire et celle-ci doit être capable de les faire appliquer en utilisant OpenFlow. Si cette approche est efficace, elle présente néanmoins des faiblesses. La suppression de paquets d'OpenFlow est agressive et implique l'introduction de « burst » de données pour les connexions TCP [Mohan 2013]. Se baser uniquement sur OpenFlow limite les possibilités de configuration de l'équipement [Sonkoly 2012], l'exemple le plus flagrant étant la configuration des files d'attentes connectées aux ports de sortie d'un commutateur qui doivent être configurés via OF-Config ou un autre protocole.
Une autre approche pour le déploiement de règles de QdS avec OpenFlow est de faire du « flow-splitting » en assignant chaque flux applicatif à un chemin spécifique. Couplé avec un algorithme d'optimisation, il est alors possible d'améliorer la qualité des communications, notamment dans le cas de la diffusion de vidéo [Egilmez 2013].

D VIRTUALISATION DES FONCTIONS RESEAU

D.1 PRINCIPES GENERAUX

Historiquement, le monde des télécoms a toujours privilégié l'utilisation d'équipements dédiés pour assurer les fonctions du réseau. Toutefois, ce choix conduit inévitablement à des délais de mise sur le marché et d'installation longs et implique des coûts d'investissements (CapEx) opérationnels (OpEx) importants ce qui est devenu handicapant dans un monde des télécoms en perpétuelle mutation et sujet à une compétition accrue. Ce modèle est remis en question par le concept de virtualisation des fonctions réseau (Network function virtualization (NFV)) qui plaide pour la virtualisation des fonctionnalités réseau sous forme de modules logiciels qui s'exécutent sur des infrastructures informatiques conventionnelles (par exemple, des serveurs de centres de données) et qu'il est possible d'assembler, de chainer, pour déployer les services offerts par le réseau. Cette approche est connexe aux techniques de virtualisation connues dans l'industrie informatique dans le sens ou une fonction virtuelle peut-être implémentée dans une ou plusieurs machines virtuelles.

L'approche NFV a initialement été proposée par un groupe d'opérateurs proches de l'ETSI (AT&T, BT, Deutsche Telekom, Orange, Telecom Italia, Telefonica, NTT, China Mobile, etc.) lors de la conférence « SDN and Openflow World Congress » de 2012 [Chio 2012]. Depuis, l'approche est portée par l'ETSI au travers d'un groupe (SIG) NFV [GS NFV 001, GS NFV 002, GS NFV 003, GS NFV 004, GS NFV-PER 002]. Depuis, cette approche suscite un engouement grandissant, puisque la virtualisation du réseau est aussi poussée au travers des futurs réseaux 5G [Deme2013] et sera surement un des concepts clés de la 5G. Le Broadband Forum mène à son tour une initiative nommée « Flexible Service Chaining ».

D.2 EXEMPLES D'APPLICATION DE L'APPROCHE NFV

L'essentiel des exemples d'application de l'approche NFV émanent du SIG NFV de l'ETSI.
Le SIG (Specific Interrest Group) NFV de l'ETSI, lancé en octobre 2013 et initialement composé de 7 membres, est aujourd'hui fort de 150 membres, principalement des opérateurs télécoms et réseaux, des équipementiers et fournisseurs de solutions informatiques[7].

L'objectif de ce groupe est de définir les besoins et l'architecture pour la virtualisation des fonctions réseau et d'adresser les défis technologiques suivants :

- Garantir que la virtualisation du réseau simplifiera l'opération des réseaux
- Définir des applications réseau portables entre différents équipementiers et avec différents hyperviseurs.
- Assurer la coexistence entre réseau classique et virtualisé, ainsi qu'assurer une transition douce vers des réseaux entièrement virtualisés en s'appuyant sur l'existant business et opérationnel.
- Permettre la gestion et l'orchestration des services réseau tout en les protégeant d'attaques ou de mauvaises configurations.
- Assurer la stabilité du réseau durant la mise en place de fonctions et services.
- Assurer un niveau suffisant de robustesse matérielle et logicielle
- Permettre la création de services réseaux qui pourraient idéalement être déployés en ligne sans recompilation sur n'importe quel hyperviseur ou équipement.

[7] http://www.etsi.org/technologies-clusters/technologies/nfv

Le groupe a identifié 9 principaux cas d'application des NFV (certains sont détaillés dans les sections suivantes) :

1. Network Function Virtualisation Infrastructure as a Service
2. Virtual Network Function as a Service (VNFaaS)
3. Virtual Network Platform as a Service (VNPaaS)
4. Virtualisation of Mobile Core Network and IMS
5. Virtualisation of Mobile base station
6. Virtualisation of Home Environment
7. Virtualisation of CDNs (vCDN)
8. Fixed Acess Network Functions Virtualisation

D.2.1 Network Function Virtualisation Infrastructure as a Service

Il s'agit de services de mise en place à la demande d'une infrastructure NFV pour permettre le déploiement de fonctions réseau virtuelles. Une infrastructure NFV repose aussi bien sur une infrastructure informatique qu'une infrastructure réseau. La catégorie de services « NFV Infrastructure as A service » repose donc sur des services de type « Infrastructure as a Service (IaaS) » et « Network as a Service (NaaS)» (Figure D.2.1).

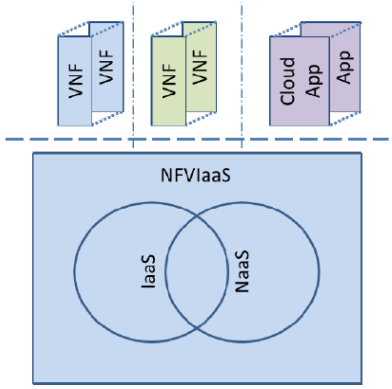


Figure D.2.1 - architecture NFVIaaS (© ETSI)

D.2.2 Virtual Network Function as a Service (VNFaaS)

Il s'agit de services de mise en place à la demande de fonctions réseau en remplacement à des équipements réseau dédiés. Les fonctionnalités réseau suivantes ont été identifiées comme étant virtualisables et donc pouvant être offertes à la demande:

a. AR - Enterprise Access Router / Enterprise CPE
b. PE - Provider Edge Router
c. FW - Enterprise Firewall
d. NG-FW - Enterprise NG-FW
e. WOC - Enterprise WAN optimization Controller
f. DPI - Deep Packet Inspection (Appliance or a function)
g. IPS - Intrusion Prevention System and other Security appliances

h. Network Performance Monitoring

D.2.3 VIRTUAL NETWORK PLATFORM AS A SERVICE (VNPAAS)

Il s'agit de services permettant de fournir une plateforme réseau virtuel ainsi que l'environnement informatique nécessaire au développement et à l'exécution de services réseau.

D.2.4 VIRTUALISATION OF MOBILE CORE NETWORK

Ce cas d'utilisation s'intéresse à l'application de la virtualisation aux réseaux de mobiles. Pour illustrer ce cas d'application, nous décrivons ci-après les résultats de travaux qui ont porté sur les réseaux LTE Evolve Packet Core (EPC) [Bas2013] et dont le but est d'identifier les fonctions disponibles sur les Serving-Gateway (S-GW) et Packet Gateway (P-GW) qui pourraient être virtualisées.

Ces deux Gatewas sont en charge de ces sept fonctions :

- **Control Signalling :** Réception et déclenchement des messages de signalisation (S-GW et P-GW).
- **Resources Management Logic** : Gestion des ressources physiques (plan de données) du nœud. Administre la mise en forme du trafic utilisateur et de sa régulation puisqu'il est en charge de la réservation de ressource en fonction de l'expression des besoins en QoS de d'utilisateur (Quality Class Identifier QCI). (S-GW et P-GW).
- **Data-plane Forwarding Rules** : Règles de commutation pour le plan de données, applique les politiques de mise en forme fournies par le *Resources Management Logic*. (S-GW et P-GW).
- **Data Plane Forwarding :** Eléments physiques de commutation des paquets à proprement parler. (S-GW et P-GW).
- **GTP Matching** : GPRS Tunneling Protocol (GTP) est un groupe de protocoles basés sur IP qui est utilisé pour le transport des paquets GPRS dans les réseaux de téléphonie mobile. Cette fonction à pour rôle l'encapsulation/desencapsulation des paquets GTP. (S-GW et P-GW).
- **Data plane Filtering and Classification** : Identification des paquets utilisateurs à partir des profils et règles sur les liens montants et descendants. (S-GW et P-GW).
- **Charging Control** : Facturation online et offline (P-GW).

Le tableau suivant résume, au travers de 8 scénarios LTE, la sollicitation de ces fonctions dans les S-GW et P-GW.

Tableau D-1 – Liste des fonctions virtualisées sollicitées dans 8 scénarios LTE

EPC Scenario	EPC Node	Signaling	Resources Management	U-plane Forwarding Rules	U-plane Forwarding	GTP	Filtering	Charging
UE Attach	S-GW	✔	✔	✔	✔	✔	-	-
	P-GW	✔	✔	✔	✔	✔	✔	✔
UE Detach	S-GW	✔	✔	✔	-	-	-	-
	P-GW	✔	✔	✔	-	-	-	-
S1-U Bearer Release	S-GW	✔	✔	✔	-	-	-	-
	P-GW	-	-	-	-	-	-	-
Service Request	S-GW	✔	✔	✔	✔	✔	-	-
(Default Bearer)	P-GW	-	-	-	-	-	-	-
Service Request	S-GW	✔	✔	✔	✔	✔	-	-
(Dedicated Bearer)	P-GW	✔	✔	✔	✔	✔	✔	✔
Tracking Area Update	S-GW	✔	✔	✔	-	-	-	-
	P-GW	✔	✔	✔	-	-	-	-
Intra Handover	S-GW	✔	✔	✔	✔	✔	-	-
with/without X2 Support	P-GW	-	-	-	-	-	-	-
Handover	S-GW	✔	✔	✔	✔	✔	-	-
from LTE to 2G/3G	P-GW	✔	✔	✔	✔	✔	✔	✔

L'approche visée pour la mise en œuvre des fonctions virtuelles est basée sur le concept des SDN. La plateforme utilisée est basée sur 3 switches OpenFlow (nec et pica8) plus un contrôleur SDN (Floodlight). L'étude montre que l'implémentation des fonctions du plan de contrôle suivantes peut se faire facilement par l'ajout d'un module au contrôleur SDN : « **Control Signalling, Resources Management Logic».** Les Fonctions « **Data-plane Forwarding Rules** », « **Data Plane Forwarding** », « **Data plane Filtering and Classification** » du plan de donnée sont possibles avec Openflow. Toutefois, les fonctions nécessitant du traitement sur les paquets (**GTP Matching**) sont plus complexes à réaliser. Quatre approches différentes sont proposées (voir Figure D.2.2):

a) transférer au contrôleur tous les paquets GPRS qui se chargera de l'encapsulation (peut-être lourd) ;

b+c) utiliser une middlebox ou un matériel spécialisé (limite l'approche de virtualisation) ;

d) étendre les fonctionnalités d'openflow en ajoutant un middleware pour le déploiement de fonctions virtuelles.

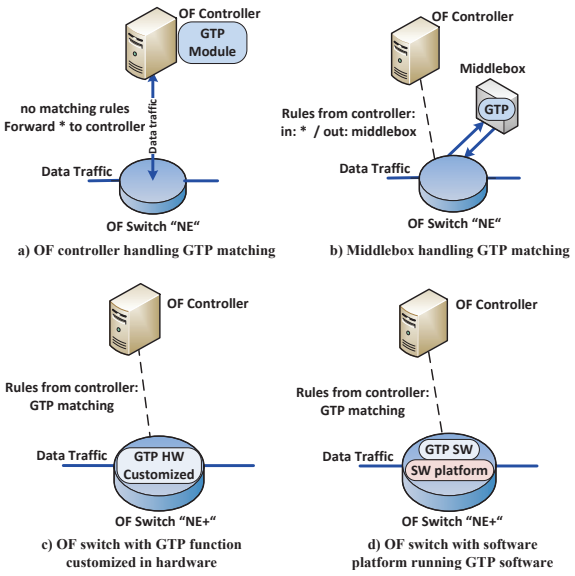
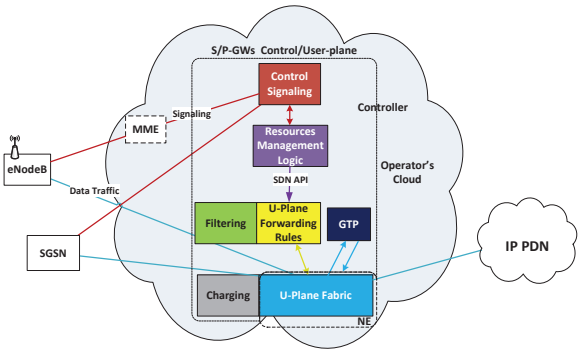


Figure D.2.2 - Différentes approches de mise en œuvre de la fonction « *GTP matching* »

L'étude a enfin mis en évidence la difficulté de faire une facturation en ligne (fonction « **Charging Control** ») du fait qu'il n'existe pas d'indication de l'état des flux dans OpenFlow.

Concernant l'architecture de déploiement, quatre architectures sont possibles : d'une architecture complétement virtualisée (Figure D.2.3) à une architecture peu virtualisée (Figure D.2.4). La première solution offrant la meilleure flexibilité mais pose des problèmes de performance, en particulier pour tout ce qui est traitement du plan de données. La dernière solution (Figure D.2.5) est mixte en déployant certaines fonctions sur les deux côtés (cloud et équipement), tout en garantissant leur synchronisation par les remontées de statistiques d'OpenFlow.



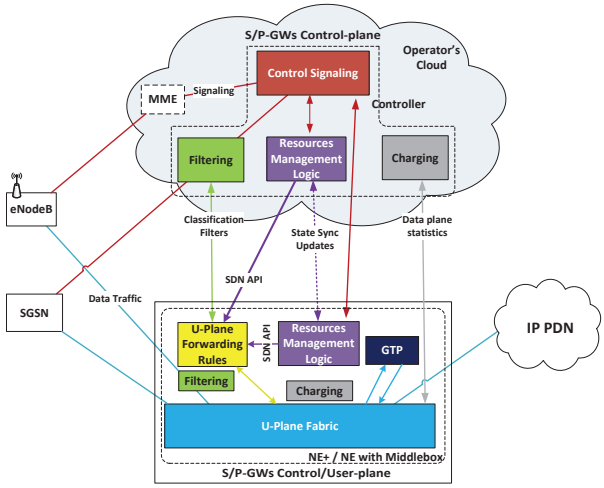
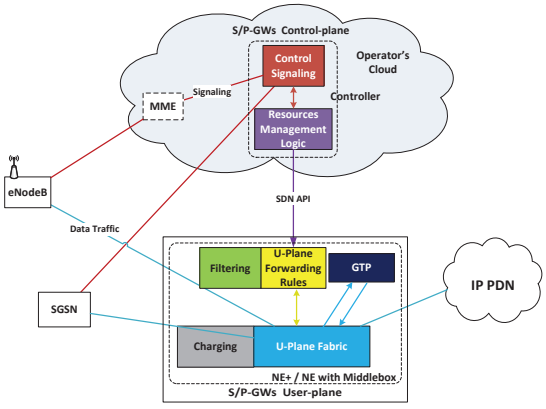


Figure D.2.3 - Architecture S-GW, P-GW virtualisée

Figure D.2.4 - Architecture S-GW, P-GW peu virtualisée



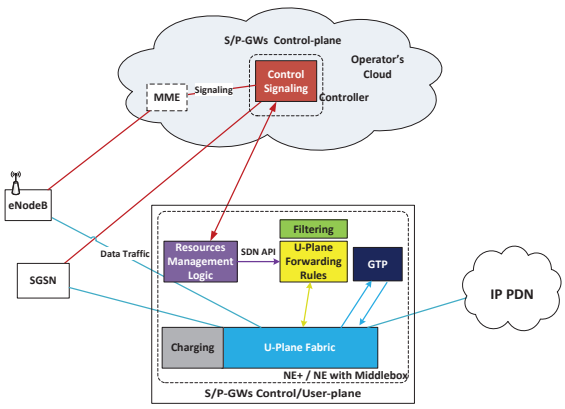


Figure D.2.5 - Architecture S-GW, P-GW mixte

D.2.5 CHAINAGE FLEXIBLE DE FONCTIONS RESEAU

Ces cas d'application sont issus du Broadband Forum qui a créé un groupe de travail relatif aux NFV dont l'objectif est d'étudier l'apport des NFV dans le contexte des accès xDSL et optiques.

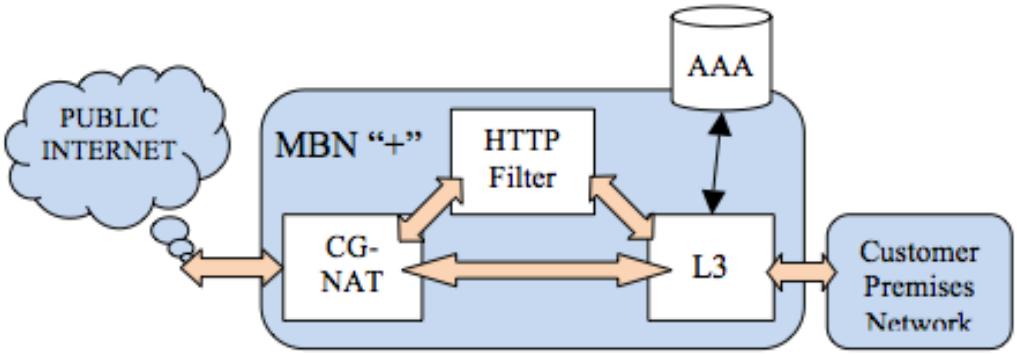


Figure D.2.6 - Accès Internet avec NAT et filtrage

Les cas d'utilisation résumés ci-après sont actuellement définis dans leur document de travail [Broad2014] :

- Accès Internet, Carrier Grade NAT, et filtrage web : Ce premier cas d'utilisation définit 3 fonctions chainables, avec comme première fonction, une fonction liée aux traitements de niveau 3 à la sortie du Customer Premise Equipement (CPE). A noter que cette fonction se trouve, ainsi que les autres, dans le réseau de l'opérateur. Viennent ensuite, optionnellement le filtrage http, ainsi que le NAT (voir Figure D-2.6);
- Accès Internet et contrôle parental. Ce deuxième cas d'utilisation se propose de définir une fonction de contrôle parental élaborée, capable de traiter y compris du trafic sécurisé.
- Interception judiciaire et Deep Packet Inspection. Cet UC propose l'utilisation de 3 fonctions, LI (Interception Judiciaire), DPI et L3 (routage).
- Distributed Service Functions : Ce cas d'utilisation a pour objectif de proposer un cas similaire au précédent mais avec éventuellement une distribution des fonctions LI et DPI au niveau du POP métropolitain (voir Figure D-2.7).

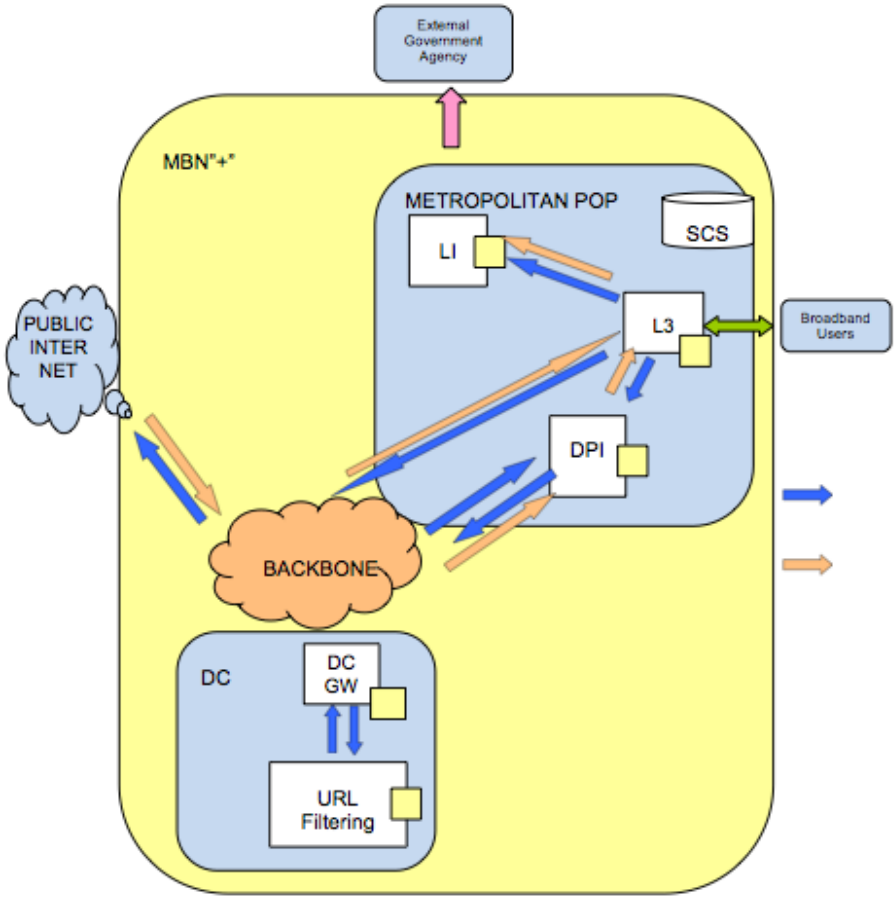


Figure D.2.7 - Distribution des fonctions sur le POP métropolitain

D.3 TECHNOLOGIES NFV

Fabricant	Produit
BROADCOM	Open NFV platform
ALCATEL-LUCENT	Cloud-Band NFV Lab
ONF	OpenStack
CloudNFV	OpenStack
INTEL	Open Network Platform (ONP)
CYAN	Blue Planet SDN Platform

Tableau D-2 - Produits et solutions commerciales pour l'intégration des NFV.

E CLOUD NETWORKING: NETWORK AS A SERVICE

E.1 PRINCIPES GENERAUX

Le Cloud Computing est un modèle émergent dans la distribution et la consommation de ressources informatiques étant donné l'attractivité et l'agilité de son modèle économique. De plus en plus d'entreprises, petites ou grandes, migrent vers le Cloud Computing pour leurs charges de travail.

Malgré l'adoption grandissante de ce modèle, la migration vers un tel modèle pose toujours des défis comme le manque d'un contrôle fin de la sécurité, la confidentialité, l'audit, mais aussi une performance imprévisible ou une efficacité faible. En plus de ces faiblesses, on retrouve aussi

l'absence de contrôle ou un contrôle limité qui est mis à disposition des clients dans les environnements Cloud Computing actuels. En effet, le modèle réseau du cloud s'est grandement contenté d'offrir une connectivité basique en utilisant des adresses IP statiques ou dynamiques allouées aux machines virtuelles des utilisateurs, avec un service de firewall basique au niveau de chaque serveur. De ce fait, plusieurs services réseaux importants ne sont pas disponibles. Parmi ces services, on peut citer :

- L'isolation fine du réseau pour une différenciation de service ou du niveau de sécurité,
- Le routage basé sur des politiques en implémentant des middleboxes dans le réseau (détection d'intrusion, DPI, etc.)
- Le contrôle de l'adressage ;
- La mise en place d'un cache distribué pour augmenter les performances, …

La dématérialisation du réseau et sa mise à disposition à la demande et en libre service via un modèle de type « Network as a Service (NaaS) » Constitue la pièce manquante au puzzle du Cloud Computing. La possibilité de spécifier et d'instancier des réseaux à la demande constitue un des plus importants avantages de la virtualisation réseau. Les réseaux virtuels peuvent être crées en fonction des différentes exigences comme la bande passante, le délai de bout-en-bout, la sécurité ou les protocoles à utiliser (voir Figure E.1.1). Cette virtualisation apporte aussi la capacité de reconfigurer le réseau en temps réel sans interrompre la connectivité pour changer par exemple déplacer un nœud virtuel d'un endroit à un autre.

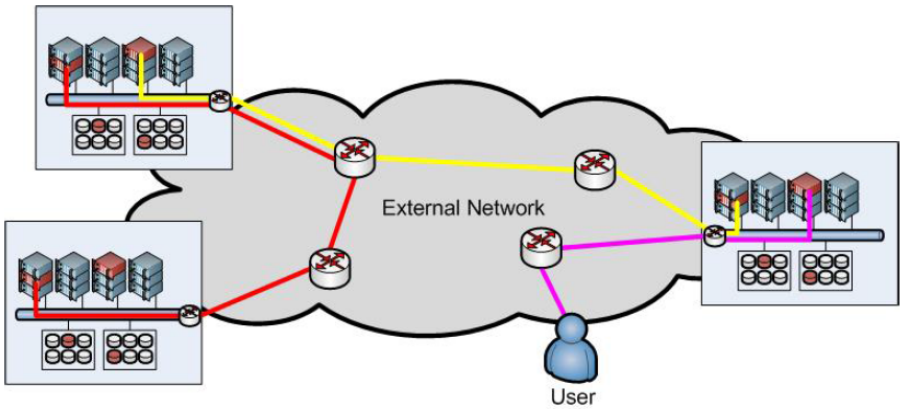


Figure E.1.1 - Un réseau virtuel dynamique pour connecter un service distribué

Les services de type NaaS intègre un certain nombre de fonctionnalités intéressantes notamment pour permettre le support de services personnalisés pour les applications :

- **Visibilité du réseau (*Network Visibility*):** Plusieurs applications qui bénéficient du Cloud Networking reposent sur un réseau overlay. Pour assurer une performance élevée, un effort considérable doit être fait pour optimiser le mapping entre les topologies physiques et logiques. Vu que les datacenters ont un niveau élevé d'utilisateurs, la prise en compte de l'endroit d'un rack dans l'overlay peut avoir un impact significatif sur les performances de l'application. Plusieurs solutions ont été proposées pour palier à ce problème. Par exemple, Orchestra [CZM 11] utilise un protocole de clusterisation complexe pour découvrir la topologie du data center et utiliser cette information pour construire efficacement sa topologie réseau.

Un datacenter a une connaissance très précise de sa topologie réseau et pourrait mettre cette information à disposition des utilisateurs sans coût supplémentaire. Cela permettrait, ainsi, aux utilisateurs d'allouer efficacement les nœuds virtuels aux machines virtuelles.

- **Forwarding personnalisé (*Custom Forwarding*) :** Une visibilité du réseau améliorerait grandement les performances des applications basées sur des overlays. Cependant, il existe quelques limitations à vouloir utiliser des réseaux d'overlay. En effet, dans un data center, souvent les serveurs ont une seule carte réseau. Ainsi, un simple arbre multicast ne pourra pas être mappé sur un réseau physique d'une manière efficace.
  De ce fait, la seconde fonctionnalité doit être la possibilité de contrôler l'acheminement des paquets au niveau des switches. Ceci permettrait l'implémentation de protocoles de routage personnalisés. Ainsi, les applications qui font appel à la duplication de paquets ou à la décision d'acheminement pourraient bénéficier cette fonctionnalité.
  - Le broadcast/multicast,
  - Content-based networking,
  - Content-centric networking,
  - Load-aware anycast,
  - Multipath
- **Traitement des paquets dans le réseau (*In-Network Processing*)** : Un des avantages du Cloud Networking réside dans son offre de capacités de traitement de paquets dans le réseau comme une partie intégrante d'une plateforme de Cloud. Par exemple, des applications distribuées comme MapReduce [DG 04] ou les applications de streaming en temps réel font intervenir de larges quantités de données qui sont généralement regroupées entre plusieurs phases de traitement. En utilisant le traitement des paquets dans le réseau, il est possible de réduire le trafic total qui est envoyé sur le réseau, et ainsi de réduire significativement la durée d'exécution de l'application considérée. Cependant, comme ces fonctions d'agrégation sont spécifiques aux applications, elles ne peuvent pas être fournies comme des services réseau traditionnels (cache, multicast, …).
  Une autre application qui peut bénéficier de cette fonctionnalité est le service de cache distribué. Par exemple, il est serait possible d'implémenter des stratégies de cache opportunistes qui prennent en compte combien de fois un paquet (ou une collection de paquets) a été « remarqué » par un certain switch.

Pour permettre une offre de NaaS en utilisant le principe de Cloud networking, les exigences sont donc :

- **Intégration avec les équipements du Datacenter :**
  Les infrastructures des datacenters actuels constituent un investissement significatif. L'utilisation d'équipements réseau de base peu ou pas programmables réduit le cout de déploiement d'un data center.
- **Modèle de programmation haut niveau :**
  Un modèle de programmation simple d'usage doit être mis à disposition aux développeurs de logiciels. Ce modèle de programmation doit cacher tous les détails réseau de bas niveau comme le traitement des paquets. Il ne doit pas aussi exposer toute la complexité de la topologie réseau physique déployée au niveau du data center.
- **Mise à l'échelle et isolation multi-utilisateur :**

Comparé à des solutions existantes de routeurs logiciels [DEA 09][HJP 10], un NaaS doit pouvoir supporter une multitude d'applications différentes, développées par des organisations différentes et qui s'exécutent conjointement sans être conscientes les unes des autres. Ainsi, le modèle NaaS requiert une forte isolation des différentes ressources réseaux mises à disposition des utilisateurs (tenants)

Comparé à une architecture classique d'un datacenter, le NaaS exige que les équipements réseaux soient capables d'exécuter le code des utilisateurs. Pour séparer d'une manière logique les fonctionnalités traditionnelles d'acheminement des paquets des fonctionnalités avancées du NaaS, le « composant de contrôle » peut être implémenté dans un équipement séparé mais connecté via des liens à très grande passante aux switchs, ou il peut être intégré dans le switch.

Afin de supporter la superposition de plusieurs réseaux virtuels, les nœuds physiques de l'infrastructure doivent être capables de supporter plusieurs nœuds virtuels, et doivent être capables de traiter les paquets des différents réseaux sans en altérer les performances. Parmi les technologies qui peuvent être utilisées, on peut citer :

- **Switchs OpenFlow :**
  Un choix naturel pour implémenter le principe du NaaS est d'utiliser des commutateurs OpenFlow. Cependant, OpenFlow suppose que seulement une petite portion des paquets est traitée par logiciel (par exemple, les paquets SYN). Ainsi, la plupart des commutateurs commerciaux offrent une petite bande passante pour le contrôleur OpenFlow (par exemple, le switch HP 5406zl supporte seulement 17Mbps [CMT 11]). Aussi, la nature d'expression des règles OpenFlow est un peu limitée pour implémenter des matching complexes comme ceux exigés pour permettre le routage basé sur le contenu (CCN).
- **NetFPGA :**
  Les NetFPGA sont très efficaces en ce qui concerne les performances de traitement des paquets. En reposant sur la programmabilité hardware, les NetFPGA peuvent supporter des traitements atteignant les 40GBps. Cependant, les langages de programmation utilisés par les NetFPGA comme le VHDL ou le Verilog, qui sont assez bas niveau, demandent une expérience considérable afin d'assurer un traitement performant. Plus encore, la notion de partage de NetFPGA par plusieurs clients n'est pas encore supportée et reste un problème de recherche ouvert.
- **Routeurs logiciels :**
  Une technologie prometteuse est représentée par les routeurs logiciels qui peuvent supporter des débits de dizaines de Gbps. Ils peuvent être utilisés seuls ou en remplacement de routeurs traditionnels, ou en combinaison des deux modes.

Le modèle NaaS introduit de nouveaux défis à relever. Contrairement à un ensemble de services implémentés par des routeurs logiciels, les équipements NaaS doivent supporter beaucoup plus d'applications appartenant à des clients différents. En plus, ces applications, écrites par des clients différents, sont en compétition pour les ressources du nœud physique. Ainsi, les équipements NaaS doivent être en mesure d'exécuter du code malicieux ou du code mal écrit sans altérer les performances pour les autres applications. De ce fait, les techniques de virtualisation traditionnelles ne sont pas très adaptées, mais des techniques plus « légères » couplées à des mécanismes d'isolation au niveau réseau doivent être développées.

E.2 TECHNOLOGIES NAAS

E.2.1 OPESTACK NEUTRON

OpenStack est un système de gestion de Cloud qui permet de contrôler un large ensemble de ressources (traitement, stockage, et réseau) à travers un datacenter (voir E.2.1). Toutes ces ressources sont gérées à l'aide d'un tableau de bord qui donne un contrôle complet aux administrateurs, tout en permettant aux utilisateurs de gérer leurs ressources à l'aide d'une interface web.

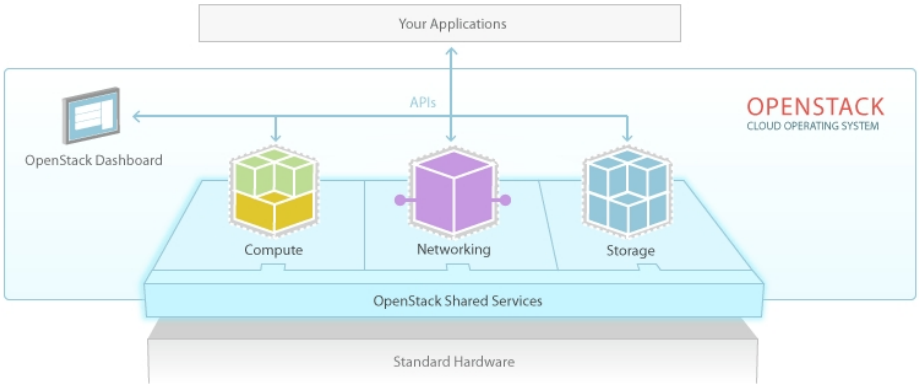


Figure E.2.1 Aperçu de l'architecture OpenStack

OpenStack repose sur une architecture modulaire, avec des noms de code pour chacun de ses composants. Neutron, anciennement Quantum, est un projet OpenStack dont le but est de fournir du NaaS entre les différentes interfaces virtuelles (vNICs) et qui est géré par d'autres services OpenStack, comme par exemple le service Nova.

Neutron offre aux utilisateurs une API pour construire des topologies réseaux avancées et configurer leurs politiques. Les fonctionnalités offertes par Neutron sont diverses, on peut citer les exemples suivants :

- Création d'une topologie réseau pour une application web multi-tier.
- Utilisation du tunneling L2-in-L3 pour contourner les limites du VLAN.
- Fourniture des garanties de qualité de service de bout-en-bout.
- Construction de services au-dessus des réseaux des utilisateurs : LB-aaS, VPN-aaS, IDS-aaS, Firewall-aaS, etc.
- Création/Suppression de réseaux L2/L3 à partir d'une interface graphique
- Démarrage de VM dans un certain réseau Neutron.

La Figure E.2.2 illustre l'architecture réseau d'OpenStack Neutron. Dans cette architecture on peut identifier trois types de nœuds :

- Nœud de traitement (Compute Node)
- Nœud réseau (Network Node)
- Nœud contrôleur (Controller Node)

On peut remarquer aussi, la présence de 3 réseaux :

- Réseau de gestion (Management network) : ce réseau est relié à la première interface de chaque nœud (*eth0*)
- Réseau interne des VMs (VM Internal Network) : utilisé pour toutes les communications entre les VMs reliés à la deuxième interface (*eth1*) et ne concerne que les nœuds de traitement et les nœuds réseaux.
- Réseau externe des VMs (VM External Network) : utilisé pour toutes les communications des VMs vers Internet. Ce réseau ne concerne que les nœuds réseaux et est rattaché à la troisième interface (*eth2*).

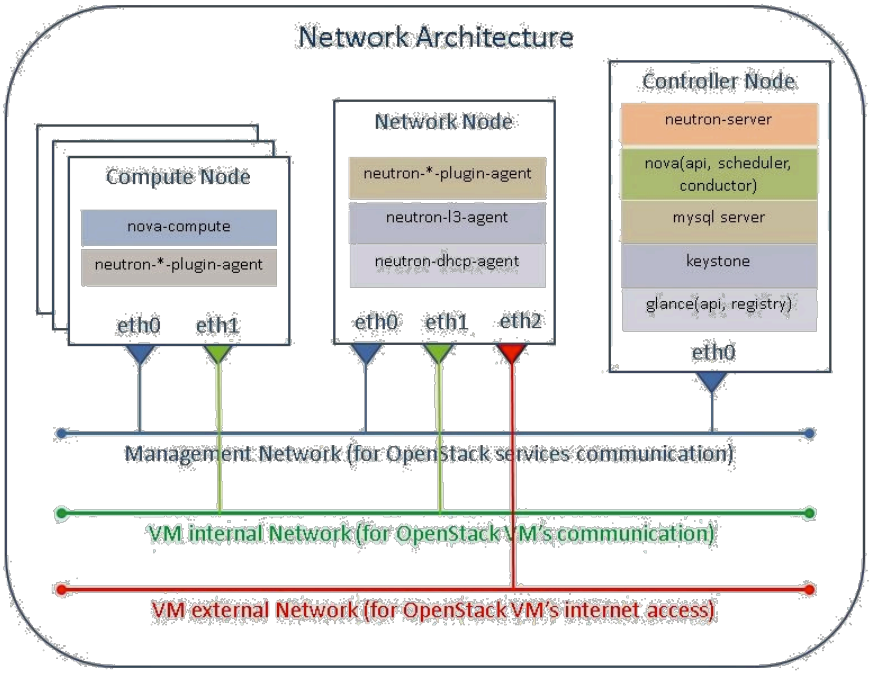


Figure E.2.2 – Aperçu de l'architecture OpenStack Neutron

Grace à la modularité de Neutron, il est constamment augmenté de nouvelles fonctions comme les drivers pour supporter des technologies L2. Les technologies suivantes sont déjà supportées : LAN, VLAN, VxLAN, GRE, Flat. Des technologies spécifiques aux constructeurs sont aussi supportées tel que : Arista, Cisco Nexus, Tail-f NCS. Cependant, Neutron souffre de quelque lacunes, prévues mais non encore implémentées. Parmi ces fonctionnalités, on peut citer :

- Découverte de l'infrastructure physique du réseau.
- Toute configuration L3 (le plugin routeur est en cours d'implémentation)
- Mécanismes de synchronisation avec d'autres systèmes de gestion

E.3 Exemples d'application du «Cloud Networking: Network as a Service»

- Migration de VMs au sein de cloud hybrides (privé, public, communauté) : Un client peut faire appel à différents fournisseurs NaaS afin de permettre un déplacement transparent de machines virtuelles entre différents Cloud, que ce soit pour des raisons de performances, de coût ou de sécurité.
- Plusieurs fournisseurs sur la même infrastructure (VNOs) : un fournisseur de réseaux d'accès peut adopter le modèle NaaS afin de « louer » une partie ou tout son réseau pour des opérateurs tiers.
- Bandwidth on Demand (BoD) : Dans un service conventionnel de BoD, chaque client doit spécifier des paramètres en termes de durée et bande passante avant l'instanciation du service, et le prix est en fonction de ces paramètres. Or, l'estimation de tels paramètres lors de l'initiation n'est pas toujours évidente. Grâce au NaaS, un fournisseur de service BoD pourra offrir cette flexibilité manquante grâce à la connaissance instantanée de l'état du réseau physique et à l'aide de l'instanciation dynamique des liens virtuels.
- Fourniture de VPNs : Ce service connait déjà un grand succès à l'instar des offres actuelles de fournisseurs de Cloud, comme Micorosft Azure [AZU 14]. Grâce à ce service, les clients d'un cloud peuvent configurer et établir des réseaux virtuels privés à la demande d'une manière dynamique. Le contrôle de ce service est poussé à la modification des paramètres du réseau en cours de fonctionnement.

F Data Distribution Service (DDS)

F.1 INTRODUCTION

DDS (Data Distribution Service) [DDS 2007, DDSi 2009] est un standard de l'OMG (Object Management Group)[8] qui cible les applications temps-réel distribuées avec des exigences de haute performance (distribution de gros volumes de données avec des latences faibles). Il présente une architecture producteur/consommateur qui permet aux applications de communiquer tout simplement, par publication, l'information dont elles disposent et de souscrire à l'information dont elles ont besoin. Le middleware DDS fournit les mécanismes nécessaires pour la prédictibilité et le contrôle des ressources tout en offrant la modularité, le passage à l'échelle, l'évolutivité, la fiabilité et la robustesse de la communication.

DDS définit un ensemble d'interfaces de niveau application permettant à des processus applicatifs d'échanger des informations selon le paradigme "producteur/consommateur" en respectant des exigences de qualité de service prédéfinies. Le standard DDS définit deux niveaux d'interface de service (voir
Figure F.1.1). Au niveau le plus bas, la couche DCPS (Data-Centric Publish Subscribe) organise les échanges de données entre les différents composants de l'application et offre aux processus applicatif un accès à un service de distribution de données efficace, résistant à l'échelle et respectant des exigences de QoS prédéfinies avec un contrôle intégré sur les ressources nécessaires. La couche optionnelle DLRL (Data Local Reconstruction Layer) située au-dessus n'offre qu'un niveau d'abstraction de la distribution de données plus élevé en donnant à l'application l'abstraction que toutes les données sont situées en local (DLRL prend en charge la gestion de la mise à jour des données). En plus des couche DCPS et DLRL, l'OMG a spécifié un protocole d'interopérabilité entre les middlewares DDS : Le protocole DDS Interoperability Wire Protocol (DDSI) qui permet d'interconnecter des applications DDS s'exécutant sur des middlewares DDS différents tels que NDDS[9], OpenSpliceDDS[10] et OpenDDS[11].

Nous nous focalisons dans la suite sur la couche DCPS.

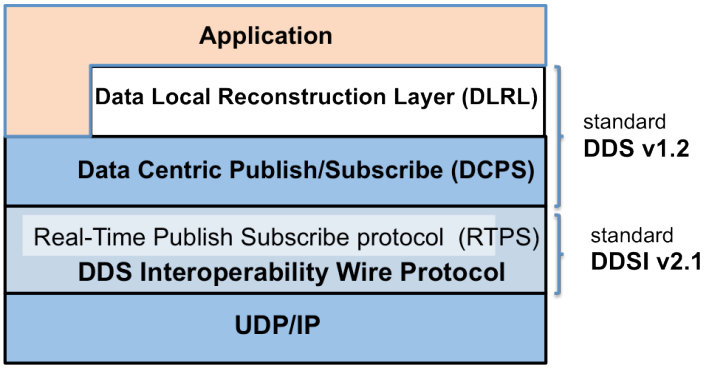


Figure F.1.1 Architecture de référence de DDS

F.2 MODELE DE COMMUNICATION DE DDS

Le modèle conceptuel de la distribution de données dans DDS repose sur une abstraction d'un espace global de données typées partagé entre des processus applicatifs producteurs (*publisher*)

[8] http://portals.omg.org/dds/
[9] http://www.rti.com/
[10] http://www.prismtech.com/opensplice/
[11] http://www.opendds.org/

qui produisent certaines de ces données et des processus consommateurs (*subscriber*) qui en consomment certaines (voir Figure F.2.1). Ces données sont spécifiées dans DDS par le biais de "*topic*" (détaillés ci-après). Ainsi, une application qui souhaite écrire une donnée dans l'espace partagé doit se déclarer comme producteur d'un *topic* et une application qui souhaite lire une donnée de l'espace partagé doit se déclarer comme consommateur du *topic* qui lui est associé.
A partir de ces déclarations d'intention (à produire ou à consommer un *topic*), le middleware DDS met automatiquement en relation les producteurs et consommateurs qui se partagent le même *topic*; DDS se charge alors de livrer les différentes valeurs (également appelés échantillons) produites aux consommateurs qui se sont déclarés intéressés par le *topic*. L'ensemble des producteurs et consommateurs d'un *topic* peut évoluer dans le temps puisque DDS supporte la détection automatique des nouvelles suscriptions/défections à un *topic*. Le service de distribution de données de DDS peut être précisément paramétré, par *topic*, par le biais d'un nombre conséquent de paramètres de Qualité de Service (désignés par DDS comme "*QoS Policy*"). Certains de ces paramètres sont dynamiques. Enfin, DDS fournit un modèle de propagation d'état qui assure une mise à jour des valeurs des données uniquement lorsque celles-ci changent.

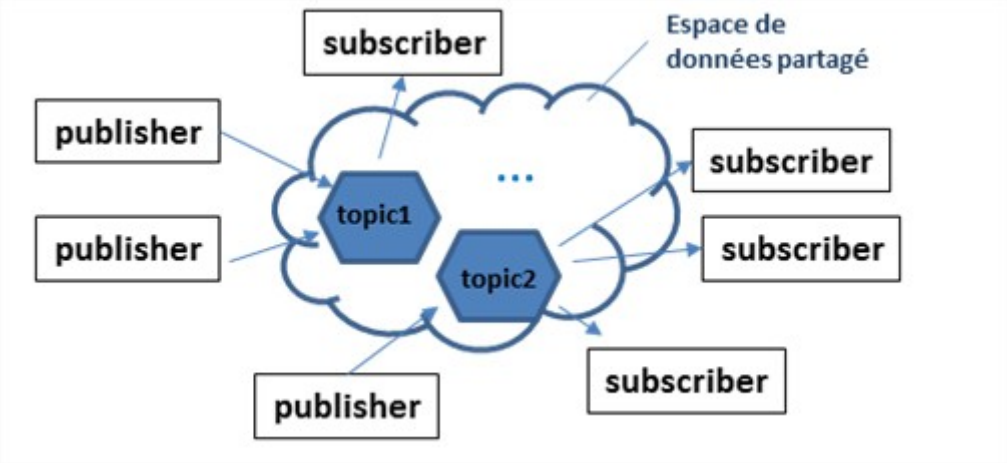


Figure F.2.1 Espace de données partagé DDS

Un topic correspond à un type de données qui peut être produit ou consommé dans l'espace global de données. Comme illustré à la Figure F.2.2 un topic associe un nom (identificateur unique), un type de données et une spécification de QoS sur la distribution des valeurs de la donnée. Il permet donc de spécifier un flux de données, échangé entre processus applicatifs, identifié par un nom, conforme à un type et qui est distribué selon une certaine QoS.
Un topic peut regrouper plusieurs instances de ce même topic qui peuvent être distinguées en utilisant une clef (désignée par "Key" dans DDS). Une instance est un sous flux du flux de données global d'un topic qui regroupe des échantillons de la donnée avec la même valeur de clef. Comme indiqué ci-après, un processus applicatif a la possibilité de souscrire à une instance de topic (un topic avec une valeur particulière de la clef).

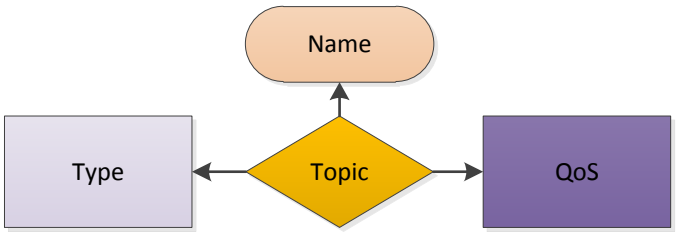


Figure F.2.2 Topic DDS (source : [DDS 2007])

Une application souhaitant produire un type de données (spécifié par un topic) doit attacher un *DataWriter* à un *publisher* (Figure F.2.3). Le *publisher* est en charge de la distribution effective des

données alors que le *DataWriter* est le moyen pour l'application d'indiquer au *publisher* la présence d'un nouvel échantillon. Un processus applicatif souhaitant transmettre un nouvel échantillon utilise donc un *DataWriter* pour activer au niveau du *publisher* la dissémination de la valeur de la donnée qui se fera en fonction de la QoS prévue. Contrairement à un *publisher*, un *DataWriter* est associé à un unique topic. Plusieurs *DataWriters* peuvent être attachés à un *publisher*. Des politiques de *QoS* y sont attachées pour paramétrer leur fonctionnement.

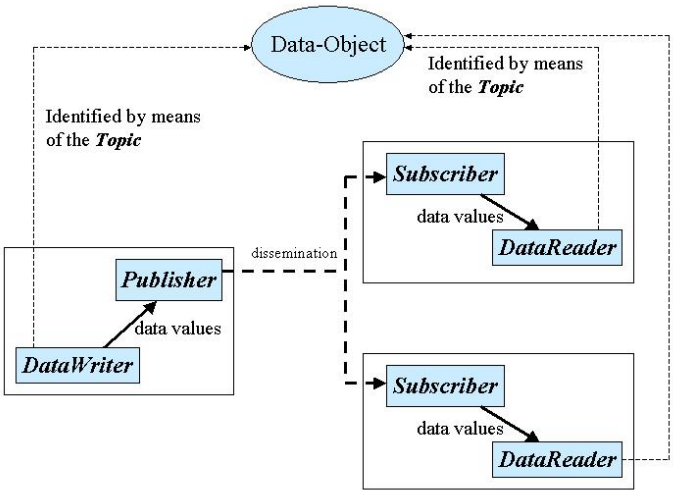


Figure F.2.3 Principales entités DDS (source : [DDS 2007])

Du côté consommateur, un *subscriber* réceptionne les échantillons publiés des Topics auxquels il a souscrit. L'application utilise de son côté un *DataReader* (un par type de données) pour récupérer les échantillons reçus. Une application souhaitant consommer un type de données (spécifié par un *topic*) doit donc attacher un *DataReader* à un *subscriber*. Ce dernier engage, pour chaque type de données consommées, une phase de souscription qui lui permet de s'associer aux *publishers* respectifs. Cette souscription peut concerner tous ou partie des échantillons d'un *topic* en jouant sur les caractéristiques d'un *topic*, à savoir le nom, la clef éventuelle (souscription à une instance) ou le contenu/valeur des échantillons. La souscription vérifie que les paramètres de QoS associées aux *publishers* et *subscribers* sont compatibles (la QoS offerte par le *publisher* répond à celle requise par le *subscriber*). En cas d'incompatibilité la souscription échoue.
Il est important de noter que ces souscriptions peuvent intervenir à tout moment (pas forcément à l'initialisation du système). En effet, DDS offre des mécanismes pour découvrir automatiquement les participants à l'espace global de données et une souscription (et un désabonnement) peut alors intervenir à tout moment sans effort particulier au niveau de l'application (ceci est possible car les *publishers* et *subscribers* sont seulement liés via les *Topics*). Enfin, DDS définit le concept de domaine qui permet de confiner la distribution de données au sein des participants à un domaine.

F.3 QoS DANS DDS

L'une des particularités de DDS est de fournir un nombre significatif de paramètres de QoS (QoS policies dans DDS) qui permettent principalement de paramétrer le service de distribution des échantillons de chaque "topic", mais aussi également de contrôler les ressources nécessaires et mécanismes de communication sous-jacents (au niveau du réseau et des nœuds terminaux) pour répondre aux exigences de QoS.
Les *QoS policies* prévues par DDS sont listées dans le Tableau F-1. Elles sont directement associées à un *topic* ou indirectement liées à un *topic* via les *DataWriter*, *DataReader*, *publisher*, ou *subscriber* de ce *topic* (voir Figure F.3.1).

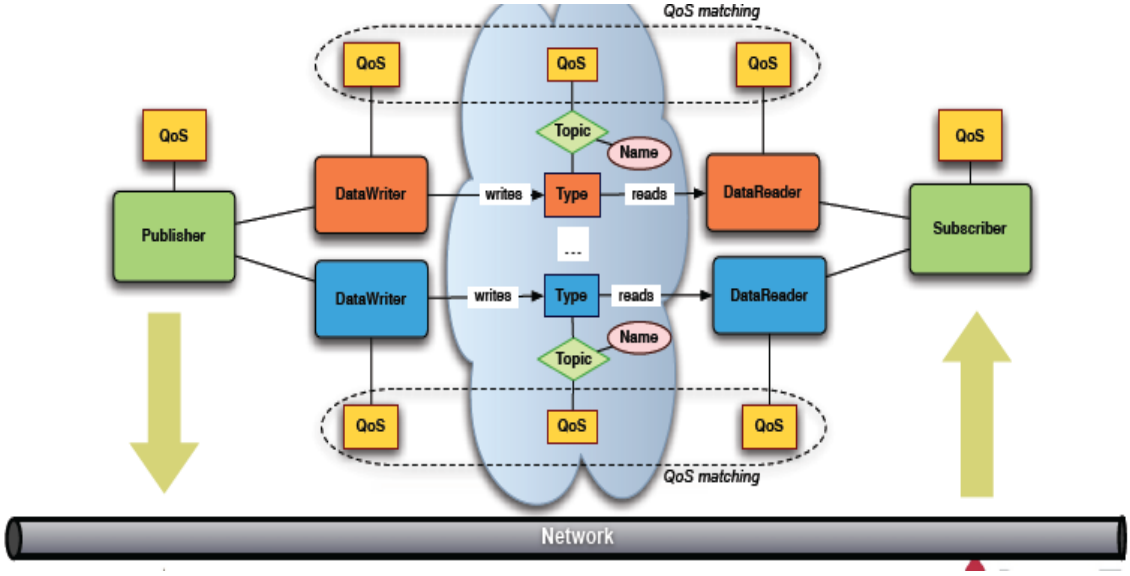


Figure F.3.1 Modèle à QoS de DDS (source : [Schmidth 2008])

Les *QoS policies* sont divisées en cinq catégories (dernière colonne du Tableau F-1) :

- "**Data availability**" qui contrôle la durée de vie (disponibilité) et la durée de validité d'un échantillon produit;

- "**Data Delivery**" qui contrôle l'entité qui a le droit d'écrire les échantillons d'un topic (en cas de présence de plusieurs producteurs possibles) et le niveau de service avec lequel ils sont distribués (exprimés en termes de fiabilité et ordre);

- "**Data Timeliness**" qui contrôle les aspects temporels associés à la distribution des échantillons d'un topic;

- "**Ressources**" qui permettent de contrôler et donc limiter les ressources nécessaires à la distribution des échantillons;

- "**Configurations**" qui correspondent à des données qui peuvent être rattachées aux entités DDS (*topic*, *DataReader*, *DataWriter*, *Publisher*, *Subscriber*) pour les besoins de l'application. Ces données sont typiquement utilisées par l'application lors de la phase de découverte des participants à l'espace partagé de données.

DDS permet aux différentes entités DDS suscités d'exprimer leur point de vue sur les caractéristiques du service de distribution. Les *DataWriter* et Publisher caractérisent le service de distribution offert côté production en décrivant certaines *QoS policies*. Ces *QoS policies* sont utilisées, côté réception, par le *DataReader* et *Subscriber* pour caractériser le service requis. C'est au moment de la demande de souscription à un topic que le middleware DDS vérifie que les spécifications de QoS des entités côté production sont consistantes avec les spécifications côté consommation (la qualité de l'offre répond aux exigences de la demande). La motivation principale de DDS derrière un tel choix est d'offrir un découplage fort dans la définition des entités (côté publication et consommation). Un échange de données entre entités ne pourra se faire que s'ils partagent le même intérêt (topic) et si la QoS de la distribution offerte par l'entité productrice répond à la QoS requise par l'entité consommatrice.). La colonne "*Applicability*" liste les entités qui peuvent être amenés à spécifier chaque *QoS policy*. La colonne RxO (qui désigne Request/Offer) liste les *QoS policies* qui seront concernées, lors de la souscription, par cette vérification de la conformité de la qualité de l'offre par rapport à la demande.

Une dernière possibilité intéressante de DDS est de permettre aux applications de modifier dans le temps certaines *QoS policies* afin d'adapter le service de distribution à de nouvelles exigences. La colonne "Modifiable" liste les *QoS policies* dont les valeurs peuvent évoluer dans le temps.

QoS Policy	Applicability	RxO	Modifiable	
DURABILITY	T, DR, DW	Y	N	Data Availability
DURABILITY SERVICE	T, DW	N	N	
LIFESPAN	T, DW	-	Y	
HISTORY	T, DR, DW	N	N	
PRESENTATION	P, S	Y	N	Data Delivery
RELIABILITY	T, DR, DW	Y	N	
PARTITION	P, S	N	Y	
DESTINATION ORDER	T, DR, DW	Y	N	
OWNERSHIP	T, DR, DW	Y	N	
OWNERSHIP STRENGTH	DW	-	Y	
DEADLINE	T, DR, DW	Y	Y	Data Timeliness
LATENCY BUDGET	T, DR, DW	Y	Y	
TRANSPORT PRIORITY	T, DW	-	Y	
TIME BASED FILTER	DR	-	Y	Resources
RESOURCE LIMITS	T, DR, DW	N	N	
USER_DATA	DP, DR, DW	N	Y	Configuration
TOPIC_DATA	T	N	Y	
GROUP_DATA	P, S	N	Y	

Tableau F-1 *QoS policies* de DDS (source : [Scmidth 2008])

Nous décrivons ci-après les *QoS policies* qui nous paraissent les plus importantes. Pour une revue complète de ces dernières, se référer à [DDS 2007]. Nous considérons implicitement que les *QoS policies* décrites ci-après concernent la distribution des échantillons d'un topic donné.

Pour la catégorie "Data Availability", nous distinguons les paramètres de QoS suivants:

- **Durability** : Elle permet à l'application de contrôler si les échantillons écrits dans l'espace global de données à un instant donné méritent d'être maintenus à l'attention d'éventuels futurs consommateurs. Ce paramètre peut prendre les valeurs suivantes: *Volatile*, *Transient* et *Persistent*. Si configurée à *volatile*, la distribution d'un échantillon touchera les *DataReaders* connus (qui y ont souscrits préalablement) au moment de sa production. Aucun échantillon n'est maintenu. Si fixée à *Transient*, le service de distribution cherchera à maintenir (au-delà du moment de production) certains échantillons pour le compte d'éventuels futurs *DataReaders*. Les échantillons maintenus sont déduits d'autres *QoS policies* (*History*, *Resource_limits*,..). Souvent, ils sont maintenus tant que le *DataWriter* existe. Enfin, si configuré à *Persistent*, ces échantillons sont sauvés de manière permanente sur des supports appropriés (mémoire non volatile).

- **LifeSpan**: Elle permet à l'application de contrôler la durée de validité d'un échantillon produit et d'éviter que des échantillons trop anciens (et donc non valides du point de vue de l'application) ne soient délivrés à l'application. Les échantillons périmés sont éliminés en les enlevant des mémoires cache des *DataReaders* et des éventuelles mémoires utilisées pour répondre à une *Durability* de *Transient* ou *persistent*.

- **History**: Cette *QoS policy* est importante dans la situation où la consommation (ou la distribution) des échantillons n'arrive pas à suivre le rythme de leur production. Dans cette situation, elle permet de contrôler si le service doit livrer tous les échantillons, ou les ***n*** derniers échantillons. La policy *History* prend alors respectivement les valeurs *KEEP_ALL*, *KEEP_LAST* avec un attribut supplémentaire *Depth* fixé à *n*. Il est à noter que cette *QoS policy* peut être fixée de manière indépendante du côté *DataReader* et *DataWriter* (pas de *RxO* à respecter). Du côté *DataWriter*, elle permet de contrôler le comportement de ce dernier si le rythme des écritures est plus grand que celui de la distribution. Du Côté du *DataReader*, elle contrôle son comportement lorsque le rythme des arrivées des échantillons dépasse le rythme de consommation de la part de l'application. Enfin, si la "*Durability*" est fixée à "*Transient*" ou "*persistent*", le paramètre *History* permet de délimiter les échantillons qui méritent d'être maintenus à l'attention de futurs *DataReaders*. La Figure F.3.2 montre un exemple d'utilisation du paramètre de QoS « History » : le producteur de gauche est configuré avec la valeur KEEP_ALL. Il crée alors une copie des échantillons qu'il souhaite produire et la garde dans son cache interne. Si un consommateur demande la retransmission d'un échantillon, suite par exemple à une perte ou la non réception, le producteur peut le récupérer directement de son cache et l'envoyer au consommateur. Le producteur suivant ne doit garder que les deux derniers échantillons produits (***KEEP_LAST 2***). Le consommateur à droite est configuré de manière à ne garder dans son cache interne que les 4 derniers échantillons (***KEEP_LAST 4***). Les autres échantillons ayant été supprimés.

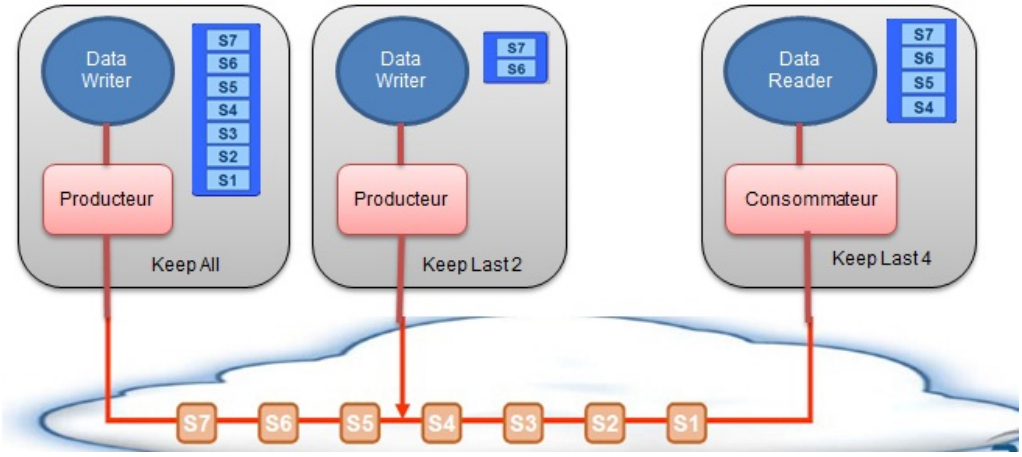


Figure F.3.2 Illustration du paramètre de QoS DDS « History » [Corsaro 2010]

Pour la catégorie "Data Delivery", nous distinguons les paramètres de QoS suivants:

- **Reliability**: Elle permet de contrôler le niveau de fiabilité du service de distribution de données. Il peut prendre deux valeurs, à savoir *Reliable* ou *Best-effort*. Dans le premier cas, le service doit délivrer tous les échantillons (délimités par la politique "*History*") à tous les *DataReaders* dans l'ordre et toute distribution d'échantillon qui échoue est retentée. Un attribut "*max_blocking_time*" est également prévu pour fixer la durée maximale de blocage de la primitive d'écriture d'un échantillon dans le cas où le *DataWriter* ne dispose pas d'espace mémoire pour le stocker (cela permet d'éviter pendant cette période de sursis la perte d'un échantillon). Dans le second cas, l'application indique qu'elle peut tolérer des échecs de distribution d'échantillons. La Figure F.3.3 illustre le comportement du

middleware DDS suite à une perte d'échantillon dans le cas d'un service de distribution en mode *Best-effort* (figure de gauche) et en mode *Reliable* (figure de droite). Dans le premier cas, aucune retransmission de l'échantillon 3 n'est mise en œuvre alors que dans le mode Reliable, l'échantillon 2 est retransmis à l'attention du consommateur.

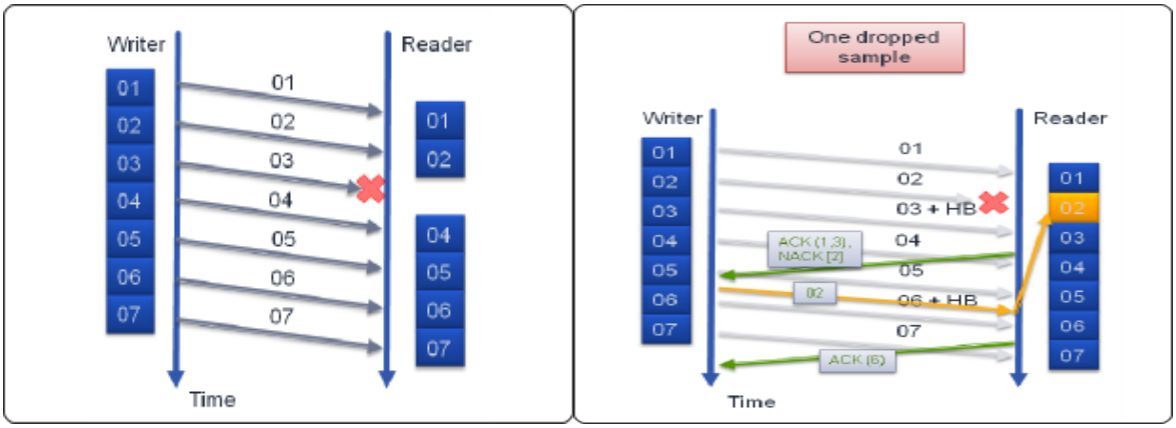


Figure F.3.3 Illustration du paramètre de QoS DDS « Reliability » [Corsaro 2010]

- **Ownership**: Il permet d'identifier le ou les producteurs qui ont le droit d'écrire des échantillons d'un topic (d'une instance de *topic*). Ce paramètre peut prendre deux valeurs: *Shared* ou *Exclusive*. Dans le premier cas, plusieurs *DataWriters* (éventuellement situés sur des nœuds différents) ont le droit d'écrire des échantillons et ces derniers seront distribués aux *DataReaders* concernés. Dans le second cas, un *topic* (ou instance de *topic*) ne peut être mis à jour à un instant donné que par un seul *DataWriter*. Seuls les échantillons produits par ce dernier seront distribués. Le choix du *DataWriter* qui a l'exclusivité de la mise à jour est déterminé par une autre *QoS Policy*, à savoir "*Ownership_Strength*". Cette dernière prend une valeur entière correspondant à un niveau de priorité qui permet de départager les différents *DataWriters*. La Figure F.3.4 présente un exemple d'utilisation de cette « *QoS Policy* » : le producteur associé au *Data Writer* B ayant l'exclusivité des productions aux dépens du *Data Writer* A qui ne pourra distribuer ses échantillons qu'en cas de retrait de B.

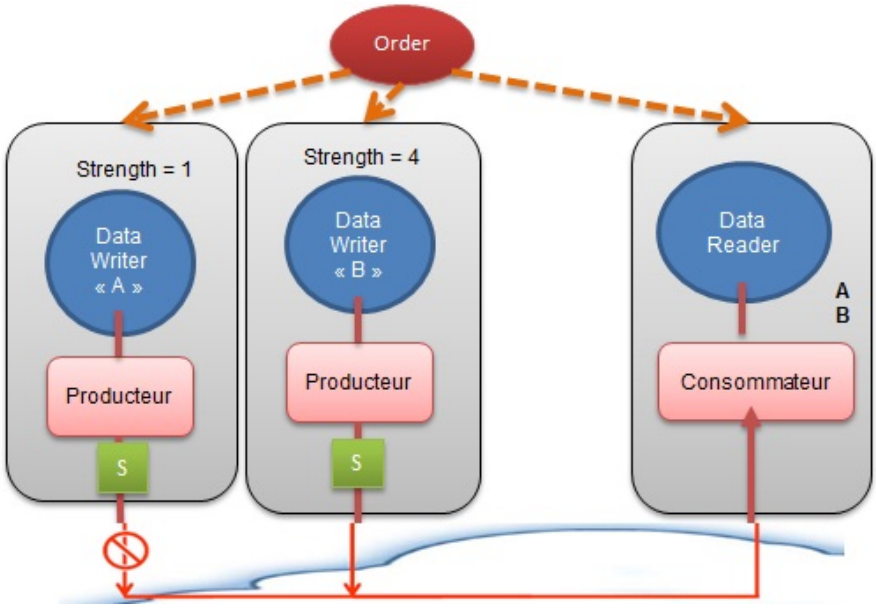


Figure F.3.4: Illustration du paramètre de QoS DDS « Ownership » [Corsaro 2010]

- **Destination_Order**: DDS permet à plusieurs *DataWriters* (situés sur des nœuds différents) d'écrire des échantillons d'un même *topic* (*Ownership=shared*). Cette *QoS policy* est utile dans ce contexte et permet de contrôler la manière avec laquelle les échantillons seront ordonnés pour être livrés aux consommateurs. Cette *QoS policy* peut prendre deux valeurs: *By_Reception_TimeStamp* ou *By_Source_TimeStamp* et permet respectivement d'ordonner les échantillons en fonction des instants d'arrivée ou d'estampilles temporelles placées côté producteurs par le service ou l'application.

Pour la catégorie "Data Timeliness", nous distinguons les paramètres de QoS suivants:

- **Deadline**: C'est une durée qui permet de spécifier, côté *DataReader*, la durée maximale entre deux arrivées successives d'échantillons. Elle permet d'identifier des exigences minimales sur le comportement du ou des *publishers*. Côté *DataWriter*, elle permet de spécifier un engagement de la part de l'application sur la durée maximale entre deux écritures successives (relatives à un topic). La Figure F.3.5 présente un diagramme temporel des arrivées des échantillons d'un topic au niveau du consommateur. A l'exception du dernier échantillon, toutes respectent le deadline côté « Data Reader ». Une violation du deadline est donc générée avant l'arrivée de ce dernier échantillon.

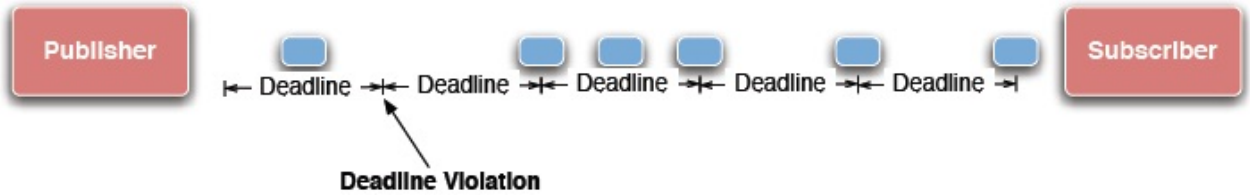


Figure F.3.5 : Illustration du paramètre de QoS DDS « *Deadline* »

- **Latency_Budget**: Il permet à l'application de spécifier une exigence sur la durée maximale entre le moment où un échantillon est écrit et le moment où il est placé dans la mémoire cache du *DataReader* à l'attention de l'application. En l'état actuel du standard DDS, ce paramètre est fourni à l'intergiciel DDS à titre informatif pour lui permettre d'évaluer l'urgence ou la criticité de la distribution. Le Middleware n'est pas obligé de garantir ce délai ni de vérifier s'il est respecté.

- **Transport_Priority**: Elle permet à l'application d'indiquer un niveau de priorité à prendre en compte lors de la distribution des échantillons. Cette indication de priorité est supposée être prise en compte par l'intergiciel et le réseau lors du traitement et la transmission des messages. "*Transport_Priority*" prend des valeurs entières. Plus la valeur est grande, plus la priorité est élevée.

- **Time_Based_Filter**: Il permet à une application consommatrice de spécifier une durée minimale à respecter entre deux arrivées successives d'échantillons. Par le biais de ce paramètre, l'application indique qu'elle ne souhaite pas forcément consommer tous les échantillons (d'un *topic* ou instance de *topic*) mais plutôt se limiter à des mises à jour séparées par cette durée minimale. Cette politique permet de mieux contrôler (et limiter) les ressources

réseau et machine nécessaires à la distribution des échantillons. La Figure F.3.6 montre un tel exemple où seuls les échantillons rouges seront délivrés car espacés de la durée minimale requise.

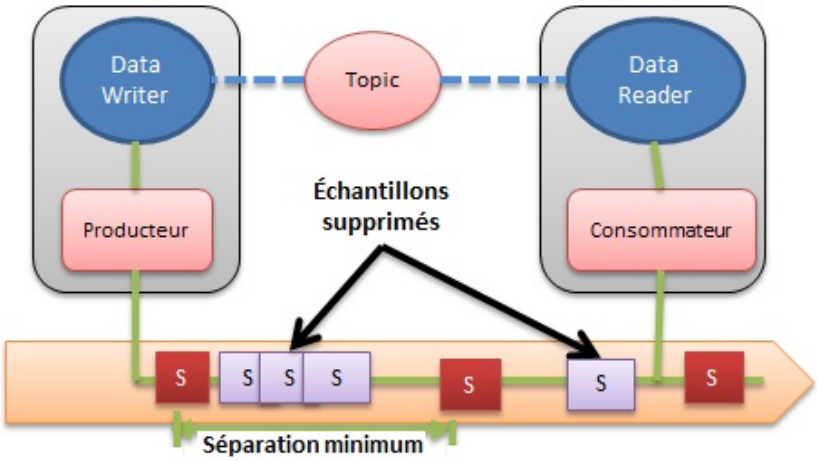


Figure F.3.6 : Illustration du paramètre de QoS DDS « Time_Based_Filter » [Corsaro 2010]

- **Resource_Limit**: Elle permet de contrôler les ressources nécessaires à la distribution des échantillons d'un *topic*. Trois paramètres sont définis

    - "*max_Samples*" qui spécifie le nombre maximal d'échantillons (toutes instances de topic confondues) qui peuvent, à un instant donnée, être gérés par un *DataReader* (ou *DataWriter*)

    - "*max_instances*" qui spécifie le nombre maximal d'instances de Topics qui peuvent être gérées à un instant donnée par un *DataReader* (ou *DataWriter*)

    - "*max_samples_per_instance*" qui spécifie le nombre maximal d'échantillons par instance qui peuvent être gérés à un instant donnée par un *DataReader* (ou *DataWriter*).

- **Partition**: c'est une *QoS Policy* associée aux entités *Subscriber* et *Publisher* qui prend comme valeurs des chaines de caractères (i.e. noms). Elle offre à l'application la possibilité de segmenter encore plus finement les flux de données qui sont échangés entre ses composants. En effet, elle contraint davantage (en plus des conditions classiques: même topic + exigences de QoS compatibles) la possibilité d'échange de données entre un DataWriter et un DataReader puisque le publisher et le subscriber à laquelle ils sont associés doivent se partager le même nom de Partition.

F.4 DDS INTEROPERABILITY WIRE PROTOCOL (DDSI)

La spécification de l'OMG du protocole DDSI décrit le protocole de découverte de nœuds d'une application distribuée DDS, à savoir le protocole SDP (Simple Discovery Protocol). SDP est composé de deux sous protocoles: SPDP (Simple Participant Discovery Protocol) et SEDP (Simple Endpoint Discovery Protocol). La spécification exige que chaque middleware DDS compatible avec le protocole d'interopérabilité DDSI (anciennement RTPS) implémente au moins SPDP. Il est possible d'implémenter son propre protocole SEDP et l'utiliser conjointement avec SPDP.

SDP utilise les publications DDS pour la découverte des autres participants. Des entités DDS spécifiques appelées « build-in entities » sont créés pour chaque Domain-participant avec des paramètres de QoS spécifiques. Le processus de découverte peut être réglé par des paramètres de QoS spécifiques qui s'appliquent à ces « build-in entities ». SDP utilise un ensemble spécifique de « Topics, DataReaders et DataWriters » pour la publication et la découverte de participants et des terminaux à travers le réseau. La figure ci-dessous illustre le scénario de découverte des entités basées sur le protocole SDP de DDS.
Ce scénario passe successivement par deux phases :

- La découverte du participant dans la phase « Participant Discovery Phase (PDP) »12 : elle consiste à publier les informations concernant le participant qui génère le message de découverte (étape « Advertises this participant » sur la figure) périodique en envoyant un échantillon (« participant DATA », sur la figure) d'un Topic spécifique « DCPSParticipant builtin Topic » et la souscription aux autres messages de découverte (périodiques aussi) envoyés par les autres participants (étape « Discovers other participants » sur la figure). Cette phase utilise un protocole de communication non fiable (best effort) pour l'envoi des messages. Le message « ParticipantData » contient des informations permettant d'établir la connexion entre deux participants (ou plus). Ces informations comportent la version du protocole, l'identification du vendeur, les informations d'adressage unicast/multicast (protocole de transport, les adresses IP la combinaison des ports) et des informations sur la façon de suivre la vivacité les participants (liveliness). Une fois que les participants sont conscients les uns des autres, ils passent à la phase de découverte « Endpoint Discovery » pour en apprendre davantage sur les DataWriters et DataReaders des autres participants, en établissant des communications fiables13.

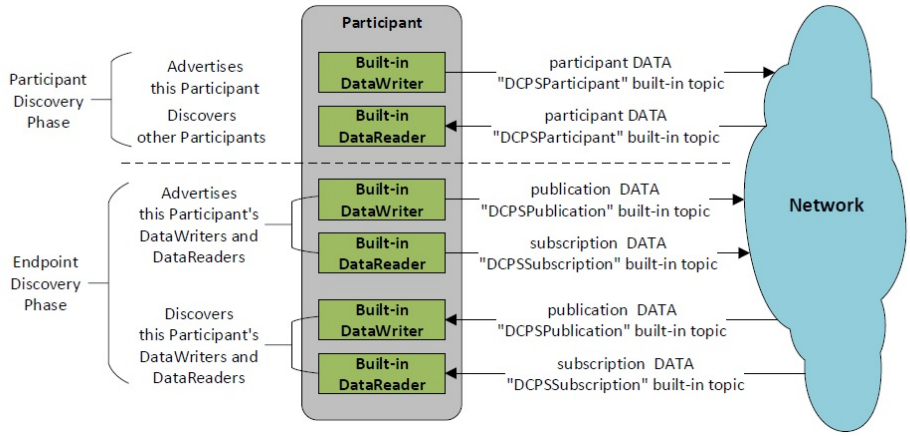


Figure F.4.1 Scénario de découverte des participants au lancement d'une application DDS

- SEDP fonctionne de manière similaire à SPDP et repose sur la publication/souscription à des « Built-in Topics » (« DCPSSubscription » et « DCPSPublication »). Plusieurs informations sont échangées entre « End-points » via ces topics : le nom du Topic du Endpoint, les types des données dans ces Topics, le data type code (la description des types de données contenus dans la structure du Topic), les paramètres de QoS de ce Topic, et d'autres informations spécifiques au standard DDSI. Le middleware doit s'assurer que les 3 premières informations (Topic name, data type, data type code) sont les mêmes et les paramètres de QoS sont consistants (respect du contrat Requested vs Offered (RxO) de QoS). Si ces informations sont correctes la communication entre le producteur du Topic et

[12] Le processus de découverte commence avec une liste connu à priori d'hôtes, une liste des adresses IP des hôtes distants est spécifiée. Si cette liste n'existe pas à priori, une adresse multicast est utilisée, il est possible aussi de spécifier une liste de groupes multicast ou unicast. Toute combinaison de ces possibilités est envisageable.

[13] Notons que même si des participants distants n'existent pas encore pendant la phase de découverte ce message est généré par le middleware (DDS) qui stocke ces informations dans sa base de données interne spécifique à la découverte des participants.

le consommateur de Topic peut s'établir correctement. Notons que les solutions commerciales du middleware DDS ajoutent d'autres paramètres de QoS DDS propriétaires qui ne figurent pas dans la liste de QoS décrite par le standard, et qui n'ont pas d'effet sur le maintien de la communication dans la phase SEDP.

F.5 ETAT DES LIEUX DES PRINCIPAUX MIDDLEWARES DDS

F.5.1 OPENSPLICE DDS

OpenSplice DDS [OpenSplice DDS 2014] est une implémentation du middleware DDS proposée par la compagnie PrismTech. Outre sa version entreprise, Opensplice DDS est disponible en version communautaire open source. Plusieurs plug-ins[14] sont proposés en complément aux fonctions classiques des autres middlewares DDS. Par exemple, DDS TouchStone est une application pour évaluer les performances du middleware dans le contexte d'un réseau local. SimD (Simple DDS) qui est une API du langage C++ ayant pour but de simplifier le développement et l'utilisation de DDS. Un autre plugin développé pour des applications DDS orientées service, est RESTful DDS. Le prototype RESTfull DDS permet aux applications pub/sub DDS d'accéder à des informations via un client web léger (i.e., écoute des connexions TCP dans un domaine DDS pour l'écoute des requêtes HTTP et exécuter des opérations de publications/souscription). Enfin, Escalier est un prototype (binding) pour le mapping des applications DDS vers des cas utilisation basée sur le langage SCALA.

F.5.2 RTI CONNEXT DDS

RTI Connext DDS [RTI DDS 2014] est une implémentation propriétaire de DDS proposée par la compagnie RTI15. RTI Connext DDS est présent dans plusieurs domaines tels que les systèmes militaires, les services financiers ou encore dans le domaine de l'imagerie médicale.

F.5.3 OPENDDS

OpenDDS [Open DDS 2010] est une version libre et open source du service de distribution de données DDS implémentée en C++. Elle est compatible avec la dernière version du standard et fourni des outils de développement ainsi qu'un EDI pour simplifier les tâches des développeurs. Plusieurs projets académiques et industriels utilisent cette version. OpenDDS s'appuie sur une couche sous-jacente l'Adaptive Communication Environment (ACE) pour fournir un environnement multiplateforme.

F.5.4 BEEDDS

BeeDDS [Bee DDS 2014] est une implémentation du middleware DDS en Java. Elle fournit les différentes fonctionnalités du standard OMG-DDS. Cette implémentation est intégrée à un environnement de développement « Sistemi Software Integrati », qui à son tour intègre une plateforme de communication de systèmes multi-agents, utilisé dans diverses applications militaires et civiles.

F.5.5 COREDX DDS

CoreDX DDS [CoreDX DDS 2014] est une implémentation de DDS qui a une très faible empreinte physique. Elle intègre les fonctions de base de DDS pour les applications temps-réel embarqués. CoreDX DDS ne propose pas d'environnement de développement intégré comme le font certaines versions évoluées du middleware DDS. CoreDX supporte plusieurs plateformes, comme Windows, Linux, Mac OS, QNX, et Android [Twin Oaks 2012]. Il n'est pas libre, c'est une implémentation commerciale, mais il est possible de se procurer d'une version de démonstration valable 30 jours.

[14] http://www.prismtech.com/opensplice/opensplice-dds-community/related-projects
[15] http://www.rti.com/

F.5.6 RTI CONNEXT MICRO

RTI Connext Micro [RTI Micro 2014] est une implémentation de RTI DDS pour les systèmes embarqués. La Figure F.5.1 montre ses principales caractéristiques. Il peut fonctionnement sur des plateformes matérielles variées et notamment avec des processeurs de faible puissance (ex. ARM) et des équipements avec une empreinte mémoire faible.

F.5.7 OPENSPLICE DDS VORTEX

Vortex [Vortex 2014] est une nouvelle implémentation de OpenSpliceDDS pour les systèmes embarqués et pour les communications machine à machine, de type Internet of Things (IoT) [Corsaro 2014]. Cette implémentation peut être portée sur plusieurs plateformes, des OS/CPU génériques jusqu'au cloud computing et la virtualisation, en passant par les systèmes d'exploitations embarqués. Elle peut être utilisée conjointement avec Android SDK pour développer des applications mobiles et M2M portables et extensibles. Les premières versions sont disponibles sous formes de licences limitées à des versions de démonstration.

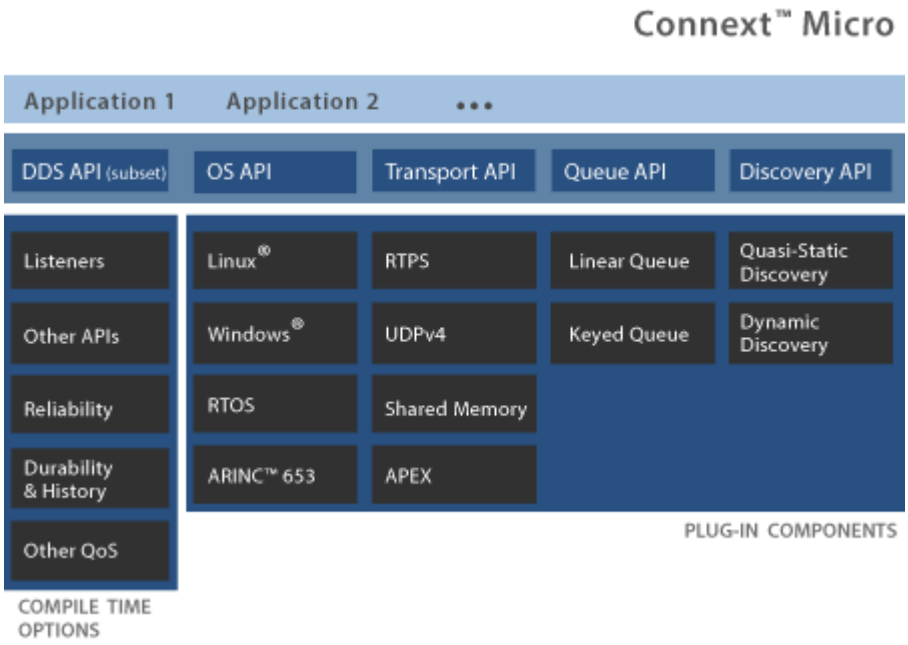


Figure F.5.1: caractéristiques de RTI Connext Micro [RTI DDS 2014]

G CONCLUSION

Ce document est le premier livrable de la tâche 1 (Système de communication – Architectures, algorithmes et mécanismes) du projet ADN dont l'objectif est de dresser un état de l'art sur les réseaux SDN, la virtualisation réseau et le middleware DDS.

La première partie de ce rapport a cherché à introduire et préciser le concept des réseaux SDN, de la virtualisation réseau, de la virtualisation des fonctions réseau et du « cloud networking » à travers les modèles de services de type « Network As A Service ». Pour chaque concept, les principales technologies ont été listées et des exemples d'application ont été présentés. Etant sujets à un fort intérêt aussi bien de la part du milieu académique et industriel avec un niveau de maturité qui est amené à se renforcer, ce travail est un état des lieux du moment qui va certainement évoluer le long du projet.

La deuxième partie de ce document a introduit le middleware DDS en se focalisant particulièrement sur ses possibilités à permettre aux applications d'exprimer leurs besoins statiques ou dynamiques de qualité de service. Cette partie a également introduit le protocole au cœur du middleware DDS, à savoir le protocole RTPS. Elle a également listé les principales implémentations DDS du marché avec leurs caractéristiques générales.

Dans la perspective du projet ADN, ce travail d'état de l'art va servir d'entrée à la tâche en charge de la définition de l'architecture de communication ADN. La programmabilité des réseaux (au sens SDN) et la virtualisation réseau seront au cœur de cette architecture. En revanche, la pertinence d'exploiter la virtualisation de fonctions réseau pourrait être étudiée. Similairement, la pertinence de reposer sur une architecture NaaS pourrait à son tour être considérée dans le cas d'un environnement maîtrisé. Enfin, l'état des lieux des différentes technologies qu'elles soient SDN, NFV ou DDS sera également un point d'entrée pour la définition des prototypes du projet ADN.

H REFERENCES

[AZU 14] Microsoft Azure, http://azure.microsoft.com

[Bas2013] "A Virtual SDN-Enabled LTE EPC Architecture: A Case Study for S-/P-Gateways Functions", Basta, A. ; Kellerer, W. ; Hoffmann, M. ; Hoffmann, K. ; Schmidt, E.-D. ; Future Networks and Services (SDN4FNS), 2013 IEEE SDN for Digital Object ; 2013 , Page(s): 1-7

[Bee DDS 2014] Bee DDS, http://www.beedds.com/en/index.html

[Bjorklund 2010] M. Bjorklund, RFC6020 YANG – A Data Modeling Language for the Network Configuration Protocol (NETCONF), (Standard), October 2010.

[Boucadair 2014] M.Boucadair, C.Jacquenet, RFC7149 Software-Defined Networking: A Perspective from within a Service Provider Environment (Informational), march 2014.

[Broad2014] "Broadband Forum work on Flexible Service Chaining (SD-326), C. Alter and al., February 2014.

[Casado 2007] M. Casado, M. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking control of the enterprise. ACM SIGCOMM Computer Communication Review, 37(4):1–12, 2007.

[Corsaro 2010] Angelo Corsaro, The DDS Tutorial - Part I, http://www.omgwiki.org/dds/sites/default/files/Tutorial-Part.II_.pdf, 2010

[Corsaro 2014] A. Corsaro, Building the Internet of Things with DDS, http://www.omg.org/news/meetings/tc/nj-13/special-events/iot-pdfs/corsaro.pdf, 2014

[Case 1990] J.Case, F.Fedor, M.Schoffstall, J.Davin, RFC1157 A Simple Network Management Protocol (SNMP), (Historic), May 1990.

[Chio2012] M. Chiosi and all. "Network Functions Virtualisation— Introductory White Paper" « SDN and Openflow World Congress » Oct. 2012, Darmstadt.

[CMT11] CURTIS, A. R., MOGUL, J. C., TOURRILHES, J., YALAGANDULA, P., SHARMA, P., AND BANERJEE, S. DevoFlow: Scaling Flow Management for High-Performance Networks. In SIGCOMM (2011).

[CoreDX DDS 2014] CoreDX DDS, http://www.twinoakscomputing.com/coredx

[CZM11] CHOWDHURY, M., ZAHARIA, M., MA, J., JORDAN, M. I., AND STOICA, I. Managing Data Transfers in Computer Clusters with Orchestra. In SIGCOMM (2011).

[DDS 2007] Data Distribution Service for Real-Time Systems Specification, DDSv1.2, http://www.omg.org/spec/DDS/1.2/, 2007

[DDSi 2009] OMG-RTPS, Data Distribution Service Interoperability Wire-Protocol Specification. DDSI v2.1, http://www.omg.org/spec/DDSI/2.1/, 2009

[DEA09] DOBRESCU, M., EGI, N., ARGYRAKI, K., CHUN, B.-G., FALL, K., IANNACCONE, G., KNIES, A., MANESH, M., AND RATNASAMY, S. RouteBricks: Exploiting Parallelism To Scale Software Routers. In SOSP (2009).

[Deme2013] « 5G on the horizon », P. Demestichas and al., IEEE Vehicular Technology Magazine, Sept. 2013.

[DG04] DEAN, J., AND GHEMAWAT, S. MapReduce: Simplified Data Processing on Large Clusters. In OSDI (2004).

[Doria 2010] A. Doria, J. H. Salim, R. Haas, H. Khosravi, W. Wang, L. Dong, R. Gopal, and J. Halpern. Forwarding and Control Element Separation (ForCES) Protocol Specification. RFC 5810 (Proposed Standard), March 2010.

[EFIA] European Future Internet Assemby, http://www.future-internet.eu/home/future-internet-assembly.html, en ligne, 2014

[Egilmez 2013] Egilmez, H.E.; Civanlar, S.; Tekalp, A.M., "An Optimization Framework for QoS-Enabled Adaptive Video Streaming Over OpenFlow Networks," Multimedia, IEEE Transactions on , vol.15, no.3, pp.710,715, April 2013

[Enns 2011] R. Enns, M. Bjorklund, J. Schoenwaelder, A. Bierman, RFC6241 Network Configuration Protocol (NETCONF), (Proposed Standard), June 2011.

[FIA] NSF Future Internet Architecture Project, http://www.nets-fia.net/, en ligne, 2014.

[FGR07] N. Feamster, L. Gao, J. Rexford. "How to lease the Internet in your spare time". Proc. Of ACM SIGCOMM , Sep 2007

[Foster 2013] N.Foster, A.Guha, M.Reitblatt, A.Stiry, M.J.Freedman, N.P.Katta, C.Monsanto, J.Reich, J.Rexfort, C.Schlesinger, D.Walker, R.Harisson, Languages for Software-Defined Networks, IEEE Communications Magazine, February 2013.

[GENI] NSF Global Environment for Network Innovations (GENI) Project, http://www.geni.net/, en ligne, 2014.

[GENI06] GENI: Global Environment for Network Innovations GDD-06-08 (2006), GENI Design Principles

[GS NFV 001] Network Functions Virtualisation (NFV); Use Cases ; ETSI SIG NFV 2013

[GS NFV 002] Network Functions Virtualisation (NFV); Architectural Framework; ETSI SIG NFV 2013

[GS NFV 003] Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV; ETSI SIG NFV 2013

[GS NFV 004] Network Functions Virtualisation (NFV); Virtualisation Requirements; ETSI SIG NFV 2013

[GS NFV-PER 002] Network Functions Virtualisation (NFV); Proofs of Concepts; Framework; ETSI SIG NFV 2013

[Haas 2010] R.Haas, RFC5813 Forwarding and Control Element Separation (ForCES) MIB (Proposed standard), march 2010.

[Hadi 2010] J.Hadi Salim, K.Ogawa, RFC5811 SCTP-Based Transport mapping Layer (TML) for the Forwarding and Control Element Separation (ForCES) Protocol (Standard proposed), march 2010.

[Halpern 2010] J.Halpern, J.Hadi Salim, RFC5812 Forwading and Control Element Separation (ForCES) Forwarding Element Model (Proposed standard), March 2010.

[HJP10] HAN, S., JANG, K., PARK, K., AND MOON, S. PacketShader: A GPU-Accelerated Software Router. In SIGCOMM (2010).

[Huang 2013] Danny Yuxing Huang, Kenneth Yocum, and Alex C. Snoeren. 2013. High-fidelity switch models for software-defined network emulation. In Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking (HotSDN '13). ACM, New York, NY, USA,

[HVR12] Router Virtualization in Service Providers white paper, http://www.cisco.com/en/US/solutions/collateral/ns341/ns524/ns562/ns573/white_paper_c11-512753_ns573_Networking_Solutions_White_Paper.html, en ligne, 2014.

[Kandoo 2012] S. Hassas Yeganeh and Y. Ganjali. Kandoo: a framework for efficient and scalable offloading of control applications. In Proceedings of the first workshop on Hot topics in software defined networks, HotSDN'12, pages 19–24, New York, NY, USA, 2012. ACM.

[Khosvari 2003] H. Khosvari, T. Anderson, RFC3654 Requirements for Separation of IP Control and Forwarding (Informational), November 2003.

[Koponen 2010] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, et al. Onix: A distributed control platform for large-scale production networks. OSDI, Oct, 2010.

[Lantz 2010] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets-IX). ACM, New York, NY, USA.

[Levin 2012] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann. Logically centralized?: state distribution trade-offs in software defined networks. In Proceedings of the first workshop on Hot topics in software defined networks, HotSDN '12, pages 1–6, New York, NY, USA, 2012. ACM.

[MCB10] N.M. Mosharaf Kabir Chowdhury and Raouf Boutaba. « A survey of network virtualization », Comput. Netw. 54, 5 (April 2010)

[McKeown 2008] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 38(2):69–74, 2008.

[Mohan 2013] Mohan, P.M.; Divakaran, D.M.; Gurusamy, M., "Performance study of TCP flows with QoS-supported OpenFlow in data center networks," Networks (ICON), 2013 19th IEEE International Conference on , vol., no., pp.1,6, 11-13 Dec. 2013

[Nascimento 2011] M. Ribeiro Nascimento, C. Esteve Rothenberg, M. R. Salvador, C. Corrêa, S. Lucena and M. F. Magalhães. "Virtual Routers as a Service: The RouteFlow Approach Leveraging Software-Defined Networks". In 6th International Conference on Future Internet Technologies 2011 (CFI 11), Seoul, Korea, June 2011.

[NEC 2013] NEC Corporation of America, SDN RFI response, November 2013.

[NGN] New-Generation Network, http://www.nict.go.jp/en/nrh/nwgn/, en ligne, 2014

[Nunes 2014] B. Astuto A. Nunes et al,, A Survey of Sofware-Defined Networking: Past, Present, and Future of Programmable Networks, IEEE Communications Surveys & Tutorials 2014.

[Ogawa 2014] K.Ogawa, W.Wang, E.Haleplidis, J. Hadi Salim, RFC7121 High Availability within a Forwarding and Control Element Separation (ForCES) Network Element, February 2014.

[ONF 2013a] Open Networking Foundation, Software-Defined Networking: The New Norm for Networks, ONF white paper, avril 2013.

[ONF 2013b] Open Networking Foundation, SDN Architecture Overview, December 2013.

[ONF 2013c] Open Networking Foundation, OpenFlow Switch Specification – version 1.4.0 (Wire Protocol 0x05), October 2013.

[ONF 2013d] Open Networking Foundation, OpenFlow-Enabled Mobile and Wireless Networks, September 2013.

[ONF 2013e] Open Networking Foundation, Use Cases and Migration Methods, 2013.

[ONF 2013f] Open Networking Foundation, Use Cases and Migration Methods, Migration Working Group, ONF white paper, 2013.

[ONF 2013g] Open Networking Foundation, North Bound Interfaces Working Group Charter, june 2013.

[ONF 2014] Open Networking Foundation, OF-CONFIG 1.2 – OpenFlow Management and Configuration Protocol, 2014.

[ONF 2014b] Open Networking Foundation, OpenFlow-enabled SDN and Network Functions Virtualization, February 2014.

[Open DDS 2014] Open DDS, http://www.opendds.org/, 2014

[OpenSplice DDS 2014] OpenSplice DDS, http://www.prismtech.com/opensplice, 2014

[Pfaff 2013] B. Pfaff, B. Davie, RFC 7047 The Open vSwitch Database Management Protocol, (Informational), December 2013.

[Phemius 2013] K. Phemius and M. Bouet. OpenFlow: Why latency does matter. In Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on, pages 680–683, 2013.

[RouterOS11] http://wiki.mikrotik.com/wiki/Manual:Virtualization, en ligne 2104

[RTI-Cisco 2013] Gerardo Pardo-Castellote and Gary Berger, OMG SDN RFI Response from RTI and Cisco, November 2013.

[RTI DDS 2014] RTI Connext DDS, http://www.rti.com/products/dds/, 2014

[RTI Micro 2014] RTI DDS, http://www.rti.com/products/micro.html

[Schmidt 2008] Wang, N.; Schmidt, D.C.; van't Hag, H.; Corsaro, A, "Toward an adaptive data distribution service for dynamic large-scale network-centric operation and warfare (NCOW) systems," Military Communications Conference, 2008. MILCOM 2008. IEEE, vol., no., pp.1, 7, 16-19 Nov. 2008

[SDN12] ONF Market Education Committee, « Software-Defined Networking: The New Norm for Networks », Open Networking Foundation, 2012

[Sherwood 2009] R.Sherwood, G.Gibb, K.Yap, G.Appenzeller, M.Casado, N.McKeown, G.Parulkar, FlowVisor: A Network Virtualization Layer, FlowVisor Technical Report, October 2009.

[Sherwood 2010] R.Sherwood, G.Gibb, K.Yap, G.Appenzeller, M.Casado, N.McKeown, G.Parulkar, Can the Production Network Be the Testbed?, 9th USENIX symposium on Operating Systems Design and Implementation OSDI'2010, Vancouver, Canada, October 2010.

[SLF12] Ryan Shea and Jiangchuan Liu, "Network Interface Virtualization: Challenges and Solutions", EEE Network 26(5): 28-34, nov 2012

[Sonkoly 2012] Sonkoly, B.; Gulyas, A.; Nemeth, F.; Czentye, J.; Kurucz, K.; Novak, B.; Vaszkun, G., "On QoS Support to Ofelia and OpenFlow," Software Defined Networking (EWSDN), 2012 European Workshop on , vol., no., pp.109,113, 25-26 Oct. 2012

[SRIOV11] Intel LAN Access Division, An Introduction to SR-IOV Technology, intel, 2011

[Stewart 2007] R. Stewart, "RFC4960 : Stream Control Transmission Protocol," Sep. 2007.

[SWPF09] Gregor Schaffrath, Christoph Werle, Panagiotis Papadimitriou, Anja Feldmann, Roland Bless, Adam Greenhalgh, Andreas Wundsam, Mario Kind, Olaf Maennel, Laurent Mathy, "Network Virtualization Architecture: Proposal and Initial Prototype", in ACM SIGCOMM Workshop on Virtualized Infastructure Systems and Architectures - Barcelona, Spain, August 2009.

[Tootoonchian 2010] A. Tootoonchian and Y. Ganjali. Hyperflow: A distributed control plane for OpenFlow. In Proceedings of the 2010 internet network management conference on Research on enterprise networking, pages 3–3. USENIX Association, 2010.

[Tootoonchian 2012] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood. On controller performance in software-defined networks. In USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE), 2012.

[Twin Oaks 2012] Twin Oaks Computing, What Can DDS Do For Android, http://www.omg.org/hot-topics/documents/dds/Android_and_DDS1.pdf , 2012

[UIT-T 2011] UIT-T, Recommendation UIT-T Y.3001 Réseaux futurs: Objectifs et but de conception, mai 2011.

[Untangle12] http://wiki.untangle.com/index.php/Virtualizing_Untangle, en ligne 2104

[VMDQ09] Advanced Virtualization I/O Queuing Technologies An Intel-Microsoft Perspective, intel 2009

[Vortex 2014] DDS Opensplce Vortex, http://www.prismtech.com/vortex/software-downloads, 2014

[Vrouter14] http://www.brocade.com/products/all/network-functions-virtualization/product-details/5400-vrouter/index.page, en ligne, 2014

[Wang 2013] Shie-Yuan Wang; Chih-Liang Chou; Chun-Ming Yang, "EstiNet openflow network simulator and emulator," Communications Magazine, IEEE , vol.51, no.9, pp.110,117, September 2013

[Wang 2013a] W.Wang, E.Haleplidis, K.Ogawa, C.Li, J.Halpern, RFC6956 Forwarding and Control Element Separation (ForCES) Logical Function Block (LFB) Library (Proposed standard), June 2013.

[Wang 2013b] W.Wang, K.Ogawa, E.Haleplidis, M.Gao, J.Hadi Salim, RFC6984 Interoperability Report for Forwarding and Control Element Separation (ForCES) (Informational), august 2013.

[Yang 2004] L. Yang, R. Dantu, T.Anderson, R. Gopal, RFC3746 Forwarding and Control Separation (ForCES) Framework (Informational), April 2004.

[Yu 2010] M. Yu, J. Rexford, M. Freedman, and J. Wang. Scalable flow-based networking with difane. In Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM, pages 351–362. ACM, 2010.