HAL
open science

Outsourcing Network Services via the NBI of the SDN

Amin Aflatoonian

To cite this version:

Amin Aflatoonian. Outsourcing Network Services via the NBI of the SDN. Networking and Internet Architecture [cs.NI]. Ecole nationale supérieure Mines-Télécom Atlantique, 2017. English. NNT : 2017IMTA0032 . tel-01758280

HAL Id: tel-01758280

https://theses.hal.science/tel-01758280v1

Submitted on 4 Apr 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

UNIVERSITE BRETAGNE LOIRE

THÈSE / IMT Atlantique

sous le sceau de l'Université Bretagne Loire

pour obtenir le grade de

DOCTEUR D'IMT Atlantique

Spécialité : Informatique

École Doctorale Mathématiques et STIC

Présentée par

Amin Aflatoonian

Préparée dans le département Systèmes réseaux, cybersécurité & droit du numérique

Laboratoire Irisa

Outsourcing Network Services Via the NBI of SDN

Thèse soutenue le 19 septembre 2017

devant le jury composé de :

Christophe Chassot
Professeur, Insa / DGEI - Toulouse / président

Noël Crespi
Professeur, Télécom SudParis / rapporteur

Pascal Lorenz
Professeur, Université Haute Alsace - IUT de Colmar / rapporteur

Mathieu Bouet
Ingénieur R&D, Thales Communications & Security S.A.S - Gennevilliers / examinateur

Karine Guillouard
Ingénieur R&D, Orange Labs - Cesson Sévigné / examinateur

Vincent Catros
Ingénieur R&D, Orange Labs - Lannion / examinateur

Ahmed Bouabdallah
Maître de conférences, IMT Atlantique / examinateur

Jean-Marie Bonnin
Professeur, IMT Atlantique / directeur de thèse

Acknowledgements

This thesis took place in Orange Labs of Rennes within the MCA's team in the context of a CIFRE contract. I would like then to thank the MCA's former team manager M. Mathieu MORANGE and current manager M. Florent LE LAIN for giving me this opportunity.

I am most grateful to the people with whom I have most closely worked. First of all, my advisor Dr. Ahmed BOUABDALLAH, who followed this work from the early beginning to the manuscript writing. I am grateful for his availability and encouragement. I am also grateful to my thesis director Dr. Jean-Marie BONNIN for his opinions, advices and for guiding my work through these four years. I would like to express my gratitude to Dr. Karine GUILLOUARD and M. Vincent CATROS for their support, availability and involvement.

I would like to thank the members of my Ph.D. committee for reading my dissertation. I feel greatly honoured that they have accepted to evaluate my work. I am most grateful to my wife, Nasim, for her constant care and support throughout the long hours of work and for her listening when I was frustrated. I am especially grateful to my parents for having supported me during my studies.

Rennes, 19 September 2017 A. A.

Abstract

Over the past decades, Service Providers (SPs) have been crossed through several generations of technologies redefining networks and requiring new business models. The economy of an SP depends on its network which is evaluated by its reliability, availability and ability to deliver new services. The ongoing network transformation brings the opportunity for service innovation while reducing costs and mitigating the locking of suppliers. Digitalization and recent virtualization are changing the service management methods, traditional network services are shifting towards new on-demand network services. These ones allow customers to deploy and manage their services independently and optimally through a well-defined interface opened to the SP's platform. To offer this freedom to its customers and to provide on-demand network capabilities, the SP must be able to rely on a dynamic and programmable network control platform. We argue in this thesis that this platform can be provided by Software-Defined Networking (SDN) technology. Indeed, the SDN controller can be used to provide an interface to service customers where they could on-demand subscribe to new services and modify or retire existing ones.

To this end we first characterize the perimeter of this class of new services. We identify the weakest management constraints that such services should meet and we integrate them in an abstract model structuring their lifecycle. This one involves two loosely coupled views, one specific to the customer and the other one to the SP. This double-sided service lifecycle is finally refined with a data model completing each of its steps.

The SDN architecture does not support all stages of the previous lifecycle. We extend it through an original Framework allowing the management of all the steps identified in the lifecycle. This Framework is organized around a service orchestrator and a resource orchestrator communicating via an internal interface. Its implementation requires an encapsulation of the SDN controller. The example of the MPLS VPN serves as a guideline to illustrate our approach. A PoC based on the OpenDaylight controller targeting the main parts of the Framework is proposed.

Providing to the SP the mastering of SDN's openness on its northbound side should largely be profitable to both SP and customers. We therefore propose to value our Framework by introducing a new and original control model called BYOC (Bring Your Own Control) which formalizes, according to various modalities, the capability of outsourcing an on-demand service by the delegation of part of its control to an external third party. Opening a control interface and offering a granular access to the underlying infrastructure leads us to take into account some characteristics, such as multi-tenancy or security, at the Northbound Interface (NBI) level of the SDN controller.

An outsourced on-demand service is divided into a customer part and an SP one. The latter exposes to the former APIs which allow requesting the execution of the actions involved in the different steps of the lifecycle. We present an XMPP-based NBI allowing opening up a secured BYOC-enabled API. The asynchronous nature of this protocol together with its integrated security functions, eases the outsourcing of control into a multi-tenant SDN framework. Delegating the control of all or a part of a service introduces some potential value-added services. Security applications are one of these BYOC-based services that might be provided by an SP. We discuss their feasibility through a BYOC-based Intrusion Prevention System (IPS) service example.

Résumé

Au cours des dernières décennies, les fournisseurs de services (SP) ont eu à gérer plusieurs générations de technologies redéfinissant les réseaux et nécessitant de nouveaux modèles économiques. L'équilibre financier d'un SP dépend principalement des capacités de son réseau qui est valorisé par sa fiabilité, sa disponibilité et sa capacité à fournir de nouveaux services. À contrario l'évolution permanente du réseau offre au SP l'opportunité d'innover en matière de nouveaux services tout en réduisant les coûts et en limitant sa dépendance auprès des équipementiers. L'émergence récente du paradigme de la virtualisation modifie profondément les méthodes de gestion des services et conduit à une évolution des services réseau traditionnels vers de nouveaux services réseau à la demande. Ceux-ci permettent aux clients du SP de déployer et de gérer leurs services de manière autonome et optimale grâce à l'ouverture par le SP d'une interface bien définie sur sa plate-forme. Pour offrir cette souplesse de fonctionnement à ses clients en leurs fournissant des capacités réseau à la demande, le SP doit pouvoir s'appuyer sur une plate-forme de gestion permettant un contrôle dynamique et programmable du réseau. Nous montrons dans cette thèse qu'une telle plate-forme peut être fournie grâce à la technologie SDN (Software-Defined Networking). Nous proposons une caractérisation préalable de la classe de services réseau à la demande, qui en fixe le périmètre. Les contraintes de gestion les plus faibles que ces services doivent satisfaire sont identifiées et intégrées à un modèle abstrait de leur cycle de vie. Celui-ci détermine deux vues faiblement couplées, l'une spécifique au client et l'autre au SP. Ce cycle de vie est complété par un modèle de données qui précise chacune de ses étapes.

L'architecture SDN ne prend pas en charge toutes les étapes du cycle de vie précédent. Nous l'étendons à travers un Framework original permettant la gestion de toutes les étapes identifiées dans le cycle de vie. Ce Framework est organisé autour d'un orchestrateur de services et d'un orchestrateur de ressources communiquant via une interface interne. Sa mise en œuvre nécessite une encapsulation du contrôleur SDN. L'exemple du VPN MPLS sert de fil conducteur pour illustrer notre approche. Un PoC basé sur le contrôleur OpenDaylight ciblant les parties principales du Framework est proposé.

La maitrise par le SP de l'ouverture contrôlée de la face nord du SDN devrait être profitable tant au SP qu'à ses clients. Nous proposons de valoriser notre Framework en introduisant un modèle original de contrôle appelé BYOC (Bring Your Own Control) qui formalise, selon différentes modalités, la capacité d'externaliser un service à la demande par la délégation d'une partie de son contrôle à un tiers externe. L'ouverture d'une interface de contrôle offrant un accès de granularité variable à l'infrastructure sous-jacente, nous conduit à prendre

en compte certaines exigences incontournables telles que le multi-tenancy ou la sécurité, au niveau de l'interface Northbound (NBI) du contrôleur SDN.

Un service externalisé à la demande est structurée en une partie client et une partie SP. Cette dernière expose à la partie client des API qui permettent de demander l'exécution des actions induites par les différentes étapes du cycle de vie. Nous présentons un NBI basé sur XMPP permettant l'ouverture d'une API BYOC sécurisée. La nature asynchrone de ce protocole ainsi que ses fonctions de sécurité natives facilitent l'externalisation du contrôle dans un environnement SDN multi-tenant. La délégation du contrôle de tout ou partie d'un service permet d'enrichir certains services d'une valeur ajoutée supplémentaire. Les applications de sécurité font partie des services BYOC pouvant être fournis par un SP. Nous illustrons leur faisabilité par l'exemple du service IPS (système de prévention d'intrusion) décline en BYOC.

Personal Bibliography

Publications

(2016) H. Jiang, A. Bouabdallah, A. Aflatoonian, J. M. Bonnin, K. Guillouard, "A Secure Multi-Tenant Framework for SDN", *9th International Conference on Security of Information and Networks (SIN '16), ACM*, New York, 2016, pp. 40-44.

(2015) A. Aflatoonian, A. Bouabdallah, K. Guillouard, V. Catros and J. M. Bonnin, "BYOC: Bring Your Own Control a new concept to monetize SDN's openness," *1st IEEE Conference on Network Softwarization (NetSoft)*, London, 2015, pp. 1-5.

(2014) A. Aflatoonian, A. Bouabdallah, V. Catros, K. Guillouard, J. M. Bonnin, "An asynchronous push/pull communication solution for northbound interface of SDN based on XMPP", *10th edition of the Francophone conference on Management of Networks and Services GRES*, Paris, France. 2014.

(2014) A. Aflatoonian, A. Bouabdallah, V. Catros, K. Guillouard, J. M. Bonnin, "An Orchestrator-Based SDN Framework with Its Northbound Interface", *In: Advances in Communication Networking. EUNICE 2014.* Lecture Notes in Computer Science, vol 8846. Springer, Cham, pp. 1-13.

Patents

(2016) M. R. Sama, L. Suciu, A. Aflatoonian, K. Guillouard, "Device and methods for controlling an IP network core", EP Patent WO/2016/034798.

Contents

Acknowledgements **i**

Abstract **iii**

Résumé **v**

Personal Bibliography **vii**

List of Figures **xiii**

List of Tables **xv**

Resumé étendu **xxi**

1 Introduction **1**
1.1 Thesis context 1
1.2 Motivation and background 2
1.3 Problem statement 2
1.4 Contributions of this thesis 3
1.5 Document structure 4

2 Programming the network **7**
2.1 Technological context 7
2.2 Modeling programmable networks 8
2.3 Fundamentals of programmable networks 9
2.4 Software-Defined Networking (SDN) 11
2.4.1 Architecture 11
2.4.2 SDN Infrastructure 11
2.4.3 SDN Southbound Interface (SBI) 12
2.4.3.1 OpenFlow Protocol 12
2.4.3.2 OpenFlow switch 13
2.4.4 SDN Controller 14
2.4.5 SDN Northbound Interface (NBI) 15
2.5 SDN Applications Analysis 16
2.5.1 SDN Applications 16
2.5.1.1 Traffic engineering 16
2.5.1.2 Mobility and wireless 17

2.5.1.3 Measurement and monitoring 17
2.5.1.4 Security 17
2.5.2 Intuitive classification of SDN applications 18
2.5.3 Impact of SDN Applications on Controller design 19
2.6 Network Function Virtualization, an approach to service orchestration 20

3 SDN-based Outsourcing Of A Network Service 23
3.1 Introduction to MPLS networks 23
3.1.1 MPLS data plan 23
3.1.2 MPLS control plan 24
3.1.3 MPLS VPN Sample Configuration 26
3.1.4 MPLS VPN Service Management 27
3.2 SDN-based MPLS 28
3.2.1 OpenContrail Solution 28
3.2.2 OpenFlow-based MPLS Networks 29
3.2.2.1 MPLS Networks in OpenDaylight controller 30
3.2.2.2 OpenDaylight native MPLS API 31
3.2.2.3 OpenDaylight VPN Service project 31
3.3 Outsourcing problematics 34

4 Service lifecycle and Service Data Model 37
4.1 Service Lifecycle 38
4.1.1 Client side Service Lifecycle 38
4.1.1.1 Client side Service Lifecycle managed by Type-1 applications 39
4.1.1.2 Client side Service Lifecycle managed by Type-2 applications 39
4.1.1.3 Client side Service Lifecycle managed by Type-3 applications 40
4.1.1.4 Global Client-side Service Lifecycle 40
4.1.2 Operator Side Service Lifecycle 41
4.1.3 The global view 43
4.2 Service Data Model 44
4.2.1 A two-layered approach 44
4.2.2 Applying the two-layered model approach on Service Lifecycle 45
4.2.2.1 Applying two-layered model on client side service lifecycle . 45
4.2.2.2 Applying two-layered model on operator side service lifecycle 45
4.3 Conclusion 48

5 An SDN-based Framework For Service Provisioning 49
5.1 Illustrating Service Deployment Example 50
5.2 Orchestrator-based SDN Framework 52
5.2.1 Internal structure of the Service Orchestrator 52
5.2.1.1 Service Request Manager (SRM) 53
5.2.1.2 Service Decomposition and Compilation Manager (SDCM) . 53
5.2.1.3 Service Configuration Manager (SCM) 55

5.2.1.4 Service Compilation Manager (SCM) - SDN Controller (SDNC) Interface . . . 56
5.2.1.5 Service Monitoring Manager (SMM) . . . 57
5.2.2 Internal architecture of the Resource Orchestrator . . . 58
5.2.3 Framework interfaces . . . 59
5.3 Implementation . . . 59
5.3.1 Hardware architecture . . . 59
5.3.2 Network architecture . . . 60
5.3.3 Software architecture . . . 61
5.4 Conclusion . . . 65

6 Bring Your Own Control (BYOC) 67
6.1 Analysis of BYOC concept . . . 69
6.1.1 Outsourcing services . . . 69
6.1.2 Software-Defined Networking (SDN) Service Lifecycle and Northbound Interface (NBI) Application Programming Interface (API)s . . . 69
6.1.2.1 Applying the Bring Your Own Control (BYOC) concept to Type 1 services . . . 70
6.1.2.2 Applying the BYOC concept to Type 2 services . . . 70
6.1.2.3 Applying the BYOC concept to Type 3 services . . . 70
6.2 Northbound Interface permitting the deployment of a BYOC service . . . 72
6.2.1 Requirements for specification of the NBI . . . 72
6.2.2 SDN NBI Implementations . . . 73
6.2.2.1 Representational state transfer (REST) . . . 73
6.2.3 XMPP As An Alternative Solution . . . 74
6.2.3.1 XMPP-based NBI . . . 75
6.2.3.2 Global design . . . 75
6.2.3.3 NBI Data Model . . . 76
6.2.4 Simulation results . . . 76
6.3 Conclusion . . . 77

7 BYOC Use Case 79
7.1 IPS Control Plane as a Service . . . 79
7.1.1 Referenced architecture . . . 79
7.1.2 Proposed solution . . . 81
7.1.3 BYOC Use Case : A VPN service secured by a BYOC-based IPS . . . 82
7.1.3.1 Implementing a secured VPN . . . 82
7.1.3.2 Opening an IPS control interface . . . 84
7.1.3.3 Decision Engine (DE) . . . 85
7.1.3.4 Decision Base (DB) . . . 86
7.1.3.5 Service Dispatcher (SD) and NBI . . . 86
7.1.3.6 Detailed components of the Guest Controller (GC) . . . 86
7.1.3.7 Applying the GC decision on the infrastructure . . . 86

7.1.4 Distributed IPS control plane 86
7.2 Conclusion 87

8 Conclusions and Future Research **91**
8.1 Contributions 91
8.2 Future researches 93
8.2.0.1 A detailed study of the theoretical and technical approach of the BYOC 93
8.2.1 BYOC as a key enabler to flexible NFV service chaining 94
8.2.2 BYOC as a key concept leading to 5G dynamic network slicing 95

Bibliography **97**

List of Figures

2.1 Global model of programmable networks (source [9]) 8
2.2 Computation and communication capabilities of programmable networks (source [9]) 9
2.3 The IETF policy architecture (source [15]) 10
2.4 SDN Architecture 11
2.5 OpenFlow Protocol in practice 12
2.6 OpenFlow Switch Components (source [23]) 13
2.7 Flow Table (source [24]) 14
2.8 A general representation of the three types of services implemented by the Guest Controller 18
2.9 Additional control functions brought by SDN Applications 20
2.10 The MANO architecture proposed by ETSI (source [62]) 21

3.1 Label Switch Routers (LSR)s 24
3.2 MPLS Architecture Elements 25
3.3 Routing protocols used by PE 25
3.4 MP-BGP update process 26
3.5 OpenContrail control plane architecture (source [69]) 28
3.6 OpenDaylight Architecture (source [32]) 30
3.7 OpenDaylight VPN Service project (source [32]) 32
3.8 VPN Configuration Using OpenDaylight VPN Service projetct 33

4.1 Client side Service Lifecycle of Type-1 applications 39
4.2 Client side Service Lifecycle of Type-2 applications 40
4.3 Client side Service Lifecycle of Type-3 applications 41
4.4 Global Client Side Service Lifecycle 42
4.5 Operator Side Service Lifecycle 43
4.6 Global Service lifecycle 44

5.1 Functional modules of the service and resource management platform. 50
5.2 Service Deployment Call Flow 51
5.3 Proposed orchestrator-based SDN Framework 53
5.4 Physical topology of the negotiated MPLS VPN service 54
5.5 Abstract architecture of the negotiated MPLS VPN service 54
5.6 Service Update Call Flow 55

5.7 REST call allowing to reserve a resource 59
5.8 Physical architecture of the implemented framework 60
5.9 Network architecture of the implemented service 60
5.10 Class diagram of the implemented framework 62
5.11 Negotiated MPLS VPN service model 63
5.12 Implemented MPLS VPN transformer simplified algorithm 63
5.13 A simplified device model template used for Ingress LSR 64

6.1 Positioning the BYOC in the SDN architecture 68
6.2 Main APIs structuring the NBI 69
6.3 Nominal call-flow for the three types of BYOC services applied to the "MPLS networks" use case 71
6.4 Communication between components 73
6.5 The XMPP-based NBI 75
6.6 NBI Overhead Charge 77

7.1 NM-DIPS architecture (source [97]) 80
7.2 ALADDIN architecture (source [98]) 81
7.3 Global architecture of the proposal 82
7.4 Physical topology of the negotiated Secured VPN service 83
7.5 Internal architecture of an IDS-end module deployed on each site 84
7.6 A model of Decision Base implemented within the SO 85
7.7 Decision Engine task flowchart 85
7.8 Internal architecture of the Security Manager 87
7.9 Distributed IPS control plane 88
7.10 A simplified abstract model of the Secured VPN service 89

List of Tables

2.1 SDN Controllers and their NBI 15

3.1 MPLS VPN configuration parameters 27
3.2 MPLS VPN configuration parameters accessible via OpenDaylight API 32
3.3 MPLS VPN configuration parameters accessible via OpenDaylight VPN Service project 34

4.1 Service lifecycle phases and their related data models and transformation methods 47

Acronyms

5G	5th generation
AJAX	Asynchronous JAvascript and Xml
ALTO	Application-Layer Traffic Optimization
API	Application Programming Interface
ARP	Address Resolution Protocol
ARPU	Average Revenue Per User
AS	Autonomous System
BGP	Border Gateway Protocol
BYOC	Bring Your Own Control
CAGR	Compound Annual Growth Rate
CapEx	Capital Expenditure
CDN	Content Distribution Network
CE	Customer Edge
CLI	Command Line Interface
CoS	Class of Service
COPS	Common Open Policy Service
COTS	Commercial Off-The-Shelf
CSV	Comma-Separated Values
DB	Decision Base
DC	Data Center
DDoS	Distributed Denial of Service
DE	Decision Engine
DHCP	Dynamic Host Configuration Protocol
DoS	Denial of Service
DPID	Datapath ID
EaYB	Earn as You Bring
eBGP	Exterior BGP
EMS	Elemental Management Systems
eNB	evolved Node B
EPC	Evolved Packet Core
EPS	Evolved Packet System
ETSI	European Telecommunications Standards Institute
FAWG	Forwarding Abstraction Working Group

FIB	Forwarding Information Base
ForCES	Forwarding and Control Element Separation
GC	Guest Controller
GRE	Generic Routing Encapsulation
GUTI	Globally Unique Temporary ID
HSS	Home Subscriber Server
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
ID	Identifier
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IGP	Interior Gateway Protocol
IM	Instant Messaging
IP	Internet Protocol
IPS	Intrusion Prevention System
ISMI	International Mobile Subscriber Identity
IT	Information Technology
JID	Jabber ID
JSON	JavaScript Object Notation
LAN	Local Area Network
LDP	Label Distribution Protocol
LSP	Label Switch Path
LSR	Label Switch Router
LTE	Long Term Evolution
MAC	Media Access Control
MANO	Management and Orchestration
MME	Mobility Management Entity
MNO	Mobile Network Operator
MP-BGP	Multiprotocol BGP
MPLS	MultiProtocol Label Switching
MPLS-TE	MPLS Traffic Engineering
MVNO	Mobile Virtual Network Operator
NaaS	Network as a Service
NBI	Northbound Interface
NETCONF	Network Configuration Protoco
NDPMs	Negotiable Data Path Models
NFV	Network Function Virtualization
ONF	Open Networking Foundation
OpEx	Operating Expenditure

OS	Operating System
OSPF	Open Shortest Path First
OSS	Operation Support System
OTT	Over The Top
OVS	Open vSwitch
OVSDB	Open vSwitch Database
P	Provider
PCEP	Path Computation Element Protocol
PDN	Packet Data Network
PDP	Policy Decision Point
PE	Provider Edge
PEP	Policy Enforcement Point
PGW	PDN Gateway
POC	Proof Of Concept
PoP	Point of Presence
POSIX	Portable Operating System Interface
QoE	Quality of Experience
QoS	Quality of Service
RAM	Random-Access Memory
RD	Route Distinguisher
REST	Representational State Transfer
RR	Route Reflector
RRC	Radio Resource Control
RRM	Resource Reservation Manager
RO	Resource Orchestrator
RSS	Rich Site Summary
RT	Route Target
RTD	Round-Trip Delay time
SAL	Service Abstraction Layer
SASL	Simple Authentication and Security Layer
SBI	Southbound Interface
SC	Service Customer
SCM	Service Compilation Manager
SD	Service Dispatcher
SDCM	Service Decomposition Compilation Manager
SDN	Software-Defined Networking
SDNC	SDN Controller
SGW	Serving Gateway
SM	Security Manager
SMDB	Service Model Database

SMM	Service Monitoring Manager
SDOs	Standard Development Organizations
SC	Service Chain
SF	Service Function
SFC	Service Function Chaining
SFF	Service Function Forwarder
SFP	Service Function Path
SLA	Service Level Agreement
SMM	Service Monitoring Manager
SNMP	Simple Network Management Protocol
SO	Service Orchestrator
SOAP	Simple Object Access Protocol
SP	Security Proxy
SRM	Service Request Manager
SSL	Secure Sockets Layer
STP	Spanning Tree Protocol
TDB	Topology Database
TCP	Transmission Control Protocol
TE	Traffic Engineering
TLS	Transport Layer Security
TTM	Time To Market
UE	User Equipement
URL	Uniform Resource Locator
VIM	Virtual Infrastructure Manager
VLAN	Virtual LAN
VM	Virtual Machine
VNF	Virtual Network Function
VNFM	Virtual Network Function Manager
vNIC	Virtual Network Interface Cart
VPN	Virtual Private Network
VRF	Virtual Routing and Forwarding
vRouter	Virtual Router
VXLAN	Virtual Extensible LAN
WAP	Wireless Access Point
XEP	XMPP Extension Protocols
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

Resumé étendu

Introduction

Au cours des dernières décennies, les fournisseurs de services (SP) ont eu à gérer plusieurs générations de technologies redéfinissant les réseaux et nécessitant de nouveaux modèles économiques. L'équilibre financier d'un SP dépend principalement des capacités de son réseau qui est valorisé par sa fiabilité, sa disponibilité et sa capacité à fournir de nouveaux services. La croissance des demandes d'accès au réseau conduisent les fournisseurs de services à rechercher des solutions rentables pour y répondre tout en réduisant la complexité et le cout du réseau et en accélérant l'innovation de service.

Le réseau d'un opérateur est conçu sur la base d'équipements soigneusement développés, testés et configurés. En raison des enjeux critiques liés à ce réseau, les opérateurs limitent autant que faire se peut, sa modifications. Les éléments matériels, les protocoles et les services nécessitent plusieurs années de standardisation avant d'être intégrés dans les équipements par les fournisseurs. Ce verrouillage matériel réduit la capacité des fournisseurs de services à innover, intégrer et développer de nouveaux services.

La transformation du réseau offre la possibilité d'innover en matière de service tout en réduisant les coûts et en atténuant les restrictions imposées par les équipementiers. Transformation signifie qu'il est possible d'optimiser l'exploitation des capacités du réseau grâce à la puissance des applications pour finalement donner au réseau du fournisseur de services une dimension de plate-forme de prestation de services numériques. L'émergence récente de la technologie Software Defined Networking (SDN) accompagné du modèle Network Function Virtualisation (NFV) permettent d'envisager l'accélération de la transformation du réseau. La promesse de ces approches se décline en terme de flexibilité et d'agilité du réseau tout en créant des solutions rentables.

Le concept SDN introduit la possibilité de découpler les fonctionnalités de contrôle et de réacheminement des équipements réseau en plaçant les premières sur une unité centrale appelée contrôleur. Cette séparation permet de contrôler le réseau à partir d'une couche applicative centralisé, ce qui simplifie les tâches de contrôle et de gestion du réseau. De plus la programmabilité du contrôleur accélère la transformation du réseau des fournisseurs de services.

Dans cette thèse nous nous intéressons à la gestion des services de télécommunication dans un environnement contrôlé. L'exemple de la gestion d'un service de connectivité (MPLS

VPN) enrichi d' un contrôle de la qualité de service (QoS) centralisé, nous sert de fil conducteur pour illustrer notre analyse. Au cours de la dernière décennie, les réseaux MPLS ont évolué et sont devenus critiques pour les fournisseurs de services. MPLS est utilisé à la fois pour une utilisation optimisée des ressources et pour l'établissement de connexions VPN.

À mesure que la transformation du réseau devient réalité et que la numérisation modifie les méthodes de gestion des services, les services de réseau traditionnels sont progressivement remplacés par les services de réseau à la demande. Les services à la demande permettent aux clients de déployer et de gérer leurs services de manière autonome grâce à l'ouverture par le fournisseur de service d'une interface bien définie sur sa plate-forme. Cette interface permet à différents clients de gérer leurs propres services possédant chacun des fonctionnalités particulières. Pour offrir cette souplesse de fonctionnement à ses clients en leurs fournissant des capacités réseau à la demande, le fournisseur de services doit pouvoir s'appuyer sur une plate-forme de gestion permettant un contrôle dynamique et programmable du réseau. Nous montrons dans cette thèse qu'une telle plate-forme peut être fournie grâce à la technologie SDN (Software-Defined Networking).

Un réseau de télécommunications fait appel à différentes technologies fournissant plusieurs types de services. Ces services sont utilisés par plusieurs clients et une mauvaise configuration d'un service client peut avoir des conséquences sur la qualité de service des autres. La position centrale du contrôleur SDN permet à l'opérateur de gérer tous les services et équipements. Cependant la fourniture d'une interface de gestion et de contrôle de service à granularité variable s'appuyant sur ce contrôleur requiert la mise en place d'une couche supplémentaire de gestion des services au-delà du contrôleur et permettant au fournisseur de services de gérer le cycle de vie du service tout en mettant à la disposition de ses clients une interface de gestion de service.

Nous présentons dans le cadre de cette thèse un framework basé sur SDN permettant à la fois de gérer le cycle de vie d'un service et d'ouvrir avec une granularité contrôlable l'interface de gestion de services. La granularité de cette interface permet de fournir différents niveaux d'abstraction au client, chacun permettant d'offrir une partie des capacités nécessaires pour un service à la demande.

Découpler le plan de contrôle et le plan de données des réseaux MPLS et localiser le premier dans un contrôleur apporte plusieurs avantages en termes de gestion de service, d'agilité de service et de contrôle de la QoS. La couche de contrôle centralisée offre une interface de gestion de service disponible pour le client. Néanmoins, cette localisation et cette ouverture peuvent créer plusieurs défis. Le backone MPLS est un environnement partagé entre les clients d'un opérateur. Pour déployer un réseau VPN l'opérateur configure un ensemble de périphériques, situés en cœur et en bordure de réseau. Ces équipements fournissent ainsi e, parallèle plusieurs services aux clients qui utilisent la connexion VPN comme moyen de transaction fiable de leurs données confidentielles.

L'externalisation du plan de contrôle vers un contrôleur SDN (SDNC) apporte beaucoup de visibilité sur le trafic échangé au sein du réseau. C'est grâce à l'interface nord (NBI) de ce

contrôleur qu'un client peut créer un service à la demande et gérer ce service dynamiquement. La granularité de cette information donne au client plus de liberté dans la création et la gestion de son service.

En plus de maintenir l'agilité de la gestion des services, nous proposons d'introduire un framework de gestion des services au-delà du SDNC. Ce framework fournit une vue d'ensemble de toutes les fonctions et commandes. Nous renforçons ce framework en ajoutant la question de l'accès du client aux ressources et services gérés. En effet, ce framework doit être capable de fournir une granularité variable, capable de gérer tous les types de services :

— **Applications de type 1 :** Le modèle abstrait de service apporté par NBI du framework permet à l'application côté client de configurer un service avec un minimum d'informations communiquées entre l'application et le framework. L'accès restreint fourni par le framework empêche les fuites de données involontaires ou intentionnelles et la mauvaise configuration du service.
— **Applications de type 2 :** Du côté sud, les blocs internes du framework reçoivent les événements de réseau venus directement à partir des ressources, ou indirectement via le SDNC. Du côté nord, ces blocs ouvrent une API aux applications leurs permettant de s'abonner à certaines métriques utilisées pour des raisons de surveillance. Sur la base d'événements remontés par les ressources, ces métriques sont calculées par des blocs internes du framework et sont envoyées à l'application appropriée.
— **Applications de type 3 :** L'accès contrôlé aux fonctions basées sur SDN assurées par le framework fournit non seulement une API de gestion de service, mais aussi une interface de contrôle de service ouvert à l'application client. L'API de contrôle avec une granularité fine permet aux clients d'avoir un accès de bas niveau aux ressources réseau via le framework. En utilisant cette API les clients reçoivent les événements réseau envoyés par les équipements, à partir desquels ils reconfigurent le service.

Afin de fournir un framework capable de mettre en œuvre les API mentionnées, nous devons analyser le cycle de vie du service en détail. Cette analyse conduit à l'identification de tous les blocs internes du framework et à leurs articulations internes pour permettre aussi bien la présentation d'une API de service et de contrôle que le déploiement l'allocation et la configuration de ressources.

Cycle de vie du service et modèle de données de service

Afin de réduire la complexité de la gestion du cycle de vie, nous divisons le cycle de vie du service global en deux points de vue complémentaires : la vue du client et celle de l'opérateur. Chacune des deux vues capture uniquement les informations utiles pour l'acteur associé. La vue globale peut cependant être obtenue en composant les deux vues partielles. Sur la base de la classification des applications abordées dans nos études, nous analysons le cycle de vie du service côté client pour les trois principaux types d'applications. Les applications de type 1 sont constituées d'applications créant un service réseau à l'aide de la NBI. Cette catégorie ne surveille ni ne modifie le service en fonction des événements réseau.

Le cycle de vie du service côté client géré par ce type d'applications contient deux étapes principales :

— Création de service : L'application spécifie les caractéristiques de service dont elle a besoin, elle négocie le SLA associé qui sera disponible pour une durée limitée et enfin elle demande une nouvelle création de service.
— Retrait du service : l'application retire le service à la fin de la durée négociée. Cette étape définit la fin de la durée de vie.

Les applications de type 2 tire parti des événements provenant de la NBI pour surveiller le service. Il est à noter que ce service peut être créé par la même application qui surveille le service.

Ce type d'application ajoute une étape supplémentaire au cycle de vie du service côté client. Ce cycle de vie contient trois étapes principales :

— Création de service.
— Surveillance de service : Une fois créé, le service peut être utilisé par le client pour une durée négociée. Pendant ce temps, certains paramètres réseau et de service seront surveillés grâce aux événements et aux notifications envoyées par le SDNC à l'application.
— Retrait de service.

Dans un cas plus complexe, c'est-à-dire les applications de type 3, une application peut créer le service via la NBI, elle surveille le service via cette interface et, en fonction des événements à venir, elle reconfigure le réseau via le SDNC. Ce type de contrôle ajoute une étape rétroactive au cycle de vie du service côté client. Celui-ci contient quatre étapes principales :

— Création de service.
— Surveillance de service.
— Modification de service : Les événements remontés par les notifications peuvent déclencher un algorithme implémenté dans l'application (implémenté au nord du SDNC), dont la sortie reconfigure les ressources réseau sous-jacentes via le SDNC.
— Retrait de service.

Un cycle de vie global de service côté client contient toutes les étapes préalables nécessaires pour gérer les trois types d'applications, discutées précédemment. Nous introduisons dans ce modèle une nouvelle étape déclenchée par les opérations côté opérateur :

— Création de service.
— Surveillance de service.
— Modification de service.
— Mis à jour de service : La gestion du réseau de l'opérateur peut entraîner la mise à jour du service. Cette mise à jour peut être émise en raison d'un problème survenant lors de l'utilisation du service ou d'une modification de l'infrastructure réseau. Cette mise à jour peut être minime, telle que la modification d'une règle dans l'un des équipements sous-jacents, ou peut avoir un impact sur les étapes précédentes, avec des conséquences sur la création du service et / ou sur la consommation du service.
— Retrait de service.

Le cycle de vie du service côté opérateur comprend en revanche six étapes principales :

— Demande de service : Une fois qu'une demande de création ou de modification de service arrive du portail de service des utilisateurs, le gestionnaire de demandes négocie le SLA et une spécification de service de haut niveau afin de l'implémenter. Il convient de noter qu'avant d'accepter le SLA, l'opérateur doit s'assurer que les ressources existantes peuvent gérer le service demandé au moment où il sera déployé. En cas d'indisponibilité, la demande sera mise en file d'attente.
— Décomposition de service, compilation : Le modèle de haut niveau du service demandé est décomposé en plusieurs modèles de service élémentaires qui sont envoyés au compilateur de service. Le compilateur génère un ensemble de configurations de ressources réseau qui composent ce service.
— Configuration de service : Sur la base du précédent ensemble de configurations de ressources réseau, plusieurs instances de ressources virtuelles correspondantes seront créées, initialisées et réservées. Le service demandé peut ensuite être implémenté sur ces ressources virtuelles créées en déployant des configurations de ressources réseau générées par le compilateur.
— Maintenance et surveillance de service : Une fois qu'un service est mis en œuvre, sa disponibilité, ses performances et sa capacité doivent être maintenues automatiquement. En parallèle, un gestionnaire de journaux de service surveillera tout le cycle de vie du service.
— Mise à jour de service : Lors de l'exploitation du service, l'infrastructure réseau peut nécessiter des modifications en raison de problèmes d'exécution ou d'évolution technique, etc. Elle entraîne une mise à jour susceptible d'avoir un impact différent sur le service. La mise à jour peut être transparente pour le service ou peut nécessiter de relancer une partie des premières étapes du cycle de vie du service.
— Retrait de service : la configuration du service sera retirée de l'infrastructure dès qu'une demande de retrait arrive au système. Le retrait du service émis par l'exploitant est hors du périmètre de ce travail.

Un framework d'approvisionnement de services SDN

Les processus de gestion des services peuvent être divisés en deux familles plus génériques : la première gère toutes les étapes exécutants les taches liées au service, depuis la négociation de service jusqu'à sa configuration et sa surveillance, et le second gère toutes les opérations basées sur les ressources. Ces deux familles gérant ensemble tout le cycle de vie du service côté opérateur. Ce framework est composé de deux couches d'orchestration principales :

— Orchestrateur de service (SO)
— Orchestrateur de ressource (RO)

L' "Orchestrateur de service" sera dédié aux opérations de la partie service et est conforme au cycle de vie du service côté opérateur :

— Demande de service
— Décomposition de service, compilation
— Configuration de service

— Maintenance et surveillance de service
— Mise à jour de service
— Retrait de service

Cet orchestrateur reçoit les ordres de service et initie le cycle de vie du service en décomposant les demandes de service complexes et de haut niveau en modèles de service élémentaires.

Ces modèles permettent de dériver le type et la taille des ressources nécessaires pour implémenter ce service. Le SO demande la réservation de ressources virtuelles à partir de la couche inférieure et déploie la configuration de service sur les ressources virtuelles via un SDNC.

L' "Orchestrateur de ressource" gère les opérations sur les ressources :
— Réservation de ressources
— Surveillance des ressources

Cet orchestrateur, qui gère les ressources physiques, réserve et lance les ressources virtuelles. Il maintient et surveille les états des ressources physiques en utilisant son interface sud.

L'architecture interne de SO est composée de cinq modules principaux :
— Gestionnaire de demande de service (SCM) : il traite les demandes de service des clients et négocie les spécifications du service.
— Gestionnaire de décomposition et compilation de service (SDCM) : il répartit toutes les demandes de service reçues en un ou plusieurs modèles de service élémentaires qui sont des modèles de configuration de ressources.
— Gestionnaire de configuration de service (SCM) : il configure les ressources physiques ou virtuelles via le SDNC.
— Contrôleur SDN (SDNC)
— Gestionnaire de surveillance de service, d'une part, il reçoit les alarmes et notifications à venir de l'orchestrateur inférieur, RO, et d'autre part il communique les notifications de service à l'application externe via la NBI.

Bring Your Own Control (BYOC)

NBI fait référence aux interfaces logicielles entre le contrôleur et ses applications. Celles-ci sont extraites à travers la couche application consistant en un ensemble d'applications et de systèmes de gestion agissant sur le comportement du réseau en haut de la pile SDN à travers la NBI.

La nature centralisée de cette architecture apporte de grands avantages au domaine de gestion de réseau. Les applications réseau, sur la couche supérieure de l'architecture, atteignent le comportement réseau souhaité sans connaître la configuration détaillée du réseau physique. L'implémentation de la NBI repose sur le niveau d''abstraction du réseau à fournir à l'application et sur le type de contrôle que l'application apporte au contrôleur, appelé SO dans notre travail.

NBI apparaît comme une frontière administrative naturelle entre l'orchestrateur SDN, géré par un opérateur, et ses clients potentiels résidant dans la couche application. Fournir à l'opérateur la capacité de maîtriser l'ouverture de SDN sur son côté nord devrait être largement profitable à l'opérateur et aux clients. Nous introduisons une telle fonctionnalité à travers le concept de BYOC : Bring Your Own Control qui consiste à déléguer tout ou partie du contrôle et/ou de la gestion de réseau à une application tierce appelée Guest Controller (GC) et appartenant à un client extérieur.

BYOC devrait clairement permettre de réduire la charge de traitement du contrôleur. En effet, les architectures et les propositions SDN existantes centralisent la plupart du contrôle de réseau et de la logique de décision dans une seule entité. Celle-ci doit supporter une charge importante en fournissant un grand nombre de services tous déployés dans la même entité. Une telle complexité est clairement un problème que BYOC peut aider à résoudre en externalisant une partie du contrôle à une application tierce. La préservation de la confidentialité de l'application client de service est un autre point important apporté par BYOC. En fait, centraliser le contrôle du réseau dans un système et passer toutes les données de ce contrôleur peut créer des problèmes de confidentialité qui peuvent empêcher l'utilisateur final, que nous appelons SC, d'utiliser le SDNC. Enfin et surtout, BYOC peut aider l'opérateur à affiner sensiblement son modèle économique basé sur SDN en déléguant un contrôle presque "à la carte" via des APIs dédiés. Une telle approche peut être exploitée intelligemment selon le nouveau paradigme de "Earn as you bring" (EaYB) que nous présentons et décrivons ci-dessous.

En effet, un client extérieur possédant un algorithme sophistiqué propriétaire peut vouloir commercialiser les traitements spécialisés associés, à d'autres clients via l'opérateur SDN qui pourrait apparaître comme un courtier de ce type de capacités. Il convient de souligner que ces avantages de BYOC peuvent en partie être compensés par la tâche non triviale de vérifier la validité des décisions prises par l'intelligence externalisée qui doivent être au moins conformes aux différentes politiques mises en œuvre par l'opérateur dans le contrôleur. Ce point qui mérite plus d'investigation pourrait faire l'objet de recherches futures.

L'externalisation d'une partie des tâches de gestion et de contrôle modifie le modèle de cycle de vie du service. Il s'agit en effet de traduire du côté client, des parties de certaines tâches appartenant initialement à l'opérateur. Une analyse minutieuse nous permet d'identifier les tâches de compilation et de surveillance, réalisées au niveau du cycle de vie du service côté opérateur, comme des candidats potentiellement intéressants, dont certaines parties peuvent être déléguées au GC. Le GC est connecté au SO à travers la NBI. C'est là que l'opérateur de service communique avec le client du service et parfois avec les applications côté client, les orchestrateurs et les GCs.

Afin de réaliser ces fonctionnalités, certaines librairies devraient être implémentées. Ces dernières prennent en charge deux catégories de tâches : 1) création, configuration et modification de service, et 2) surveillance de service et contrôle de service BYOC. La première utilise une interaction synchrone qui implémente une simple communication requête / réponse

et qui permet à l'application côté client d'envoyer des demandes de service et des modifications, tandis que la seconde utilise une interaction asynchrone où une notification sera envoyée à l'application de service abonnée. La nature asynchrone de cette librairie la rend utile pour envoyer des messages de contrôle au GC. La communication entre le GC et le SO est basée sur l'algorithme Push-and-Pull (PaP) essentiellement utilisé dans les applications web. Dans cette proposition, nous essayons d'adapter cet algorithme pour déterminer la méthode de communication de la NBI qui utilisera le paradigme de publication / soumission de messagerie.

L'importance d'un système de provisionnement de service est basée sur son NBI qui connecte le portail de service au système de provisionnement de service (SO) dans un environnement SDN. Cette interface fournit une abstraction de la couche service et des fonctions essentielles pour créer, modifier et détruire un service, et, comme décrit ci-dessus, elle prend en compte les unités de contrôle externalisées appelées GC. Cette interface est un point d'accès partagé entre différents clients, chacun contrôlant des services spécifiques avec un abonnement associé à certains événements et notifications. Il est donc important que cette interface partagée implémente un environnement isolé pour fournir un accès multi-tenant. Celui-ci devrait être contrôlé à l'aide d'un système intégré d'authentification et d'autorisation.

Dans notre travail, nous introduisons une NBI basée sur le protocole XMPP. Ce protocole est développé à l'origine comme un protocole de messagerie instantané (IM) par la communauté. Ce protocole utilise une technologie de streaming pour échanger des éléments XML, appelés stanza, entre deux entités du réseau, chacune identifiée par un unique identifiant JID. La raison principale de la sélection de ce protocole pour implémenter la NBI du système de provisionnement de services repose sur son modèle d'interaction asynchrone qui, à l'aide de son système push intégré, autorise l'implémentation d'un service BYOC.

Conclusion et perspectives

Afin de définir un framework d'approvisionnement de service basé sur SDN permettant de définir les couches de contrôle et d'application, une analyse du cycle de vie du service devait avoir lieu. Nous avons organisé l'analyse du cycle de vie du service selon deux points de vue : client et opérateur. La première vue concerne le cycle de vie du service client qui traite les différentes phases dans lesquelles un client de service (ou client) peut être pendant le cycle de vie du service. Cette analyse est basée sur la classification des applications et des services que nous avons précédemment faite. Selon cette classification, un client de service peut utiliser l'interface de gestion de service pour gérer trois types de services. Le premier est le cas où le client demande et configure un service. Le deuxième type est le client qui surveille son service, et le troisième est le client qui, en utilisant l'interface de gestion, reçoit certains paramètres de service sur la base desquels il reconfigure ou met à jour ce service. Sur la base de cette analyse, le cycle de vie du service côté client peut être modifié. Nous avons analysé toutes les phases que chaque type de service pourrait ajouter au cycle de vie du service. D'un autre côté, l'analyse du cycle de vie du service côté opérateur présente un

modèle de cycle de vie du service représentant toutes les phases qu'un opérateur doit traverser pour déployer, configurer et maintenir un service. Cette analyse recto-verso permet de déterminer les actions que chaque client et opérateur de service peut effectuer sur un service qui est l'objet commun entre un client et un opérateur.

Nous avons présenté pour la deuxième fois le modèle de données de chaque cycle de vie basé sur une approche de modèle de données à deux couches. Dans cette approche, un service peut être modélisé en deux modèles de données : service et dispositif, et un modèle élémentaire, appelé transformation, définit comment l'un de ces deux modèles peut être transformé en un autre. Le modèle de service est un modèle général et simplifié du service présenté au client du service. Et le modèle de périphérique est la définition technique de la configuration de périphérique générée sur la base du modèle de service négocié. L'objet de service partagé entre l'opérateur et le client est décrit dans le modèle de service. Par conséquent, le cycle de vie du service côté client utilise le modèle de service et toutes les phases du cycle de vie sont basées sur ce modèle. Le modèle de service traverse le cycle de vie côté opérateur et est transformé en un ou plusieurs modèles de ressource.

L'analyse du cycle de vie du service nous donne un outil pour déterminer toutes les activités qu'un opérateur doit effectuer pour gérer un service. Basé sur le cycle de vie du service côté opérateur, nous proposons un framework à travers lequel un modèle de service présenté au client est transformé en modèles de ressources déployés sur des ressources. L'architecture de ce framework repose sur un système à deux couches gérant le cycle de vie du service via deux orchestrateurs : orchestrateur de service et orchestrateur de ressource. Le premier regroupe toutes les fonctions permettant à l'opérateur de gérer un service verticalement, et le second gère les ressources nécessaires au premier pour déployer un service.

Le framework proposé donne lieu à un système de déploiement et de gestion de services. Il ouvre une interface du côté des clients. Nous présentons un nouveau modèle de contrôle de service, appelé Bring Your Own Control (BYOC) qui suit le modèle d'application de type 3. Nous introduisons BYOC comme un concept permettant de déléguer, à travers la NBI, le contrôle de tout ou partie d'un service à un contrôleur externe, appelé Guest Controller (GC). Ce dernier peut être géré par le même client demandant et consommant le service ou par un opérateur tiers. L'ouverture d'une interface de contrôle au nord de la plate-forme SDN nécessite certaines spécifications au niveau de NBI. Nous avons abordé dans la suite de notre travail les exigences de la NBI permettant d'ouvrir l'API de BYOC. Sur la base de ces exigences, nous avons proposé l'utilisation de XMPP comme le protocole permettant de déployer une telle API.

L'analyse des avantages du concept BYOC et les problèmes de complexité et de sécurité que BYOC peut apporter au processus de gestion des services peuvent faire l'objet d'un travail futur. Cette analyse nécessite une étude plus sophistiquée de ce concept, du modèle économique potentiel qu'il peut introduire (ex. Earn as You Bring EaYB), des méthodes et des protocoles utilisés pour implémenter l'interface nord et contrôler l'accès aux ressources exposées au GC, et l'impact réel de ce type de services sur la performance des services.

L'ouverture de l'interface de contrôle de type BYOC permet de créer des nouveaux modèles de service non seulement dans le domaine SDN mais aussi dans les domaines NFV et 5G.

Chapter 1

Introduction

In this chapter we introduce the context of this thesis followed by the motivation and background of this studies. Then we present our main contributions and we conclude by the structure of this document.

1.1 Thesis context

Over the past two decades, service providers have been crossed through several generations of technologies redefining networks and requiring new business models. The economy of a Service Provider depends on its network which is evaluated by its reliability, availability and ability to deliver services. Due to the introduction of new technologies requiring a pervasive network, new and innovative applications and services are increasing the demand for network access [1]. Service Providers, on the other hand, are looking for a cost-effective solution to meet this growing demand while reducing the network complexity [2] and costs (i.e. Capital Expenditure (CapEx) and Operating Expenditure (OpEx)), and accelerating service innovation.

The network of an Operator is designed on the basis of equipments that are carefully developed, tested and configured. Due to the importance of this network, the operators avoid the risks of modifications made to the network. Hardware elements, protocols and services require several years of standardization before being integrated into the equipment by suppliers. This hardware lock-in reduces the ability of Service Providers to innovate, integrate and develop new services.

The network transformation brings the opportunity for service innovation while reducing costs and mitigating the locking of suppliers. Transformation means making it possible to exploit network capabilities through application power. This transformation converts the Operator network from a simple utility to a digital service delivery platform. The latter not only increases the velocity of the service, but also creates new sources of revenue. Recently Software-Defined Networking (SDN) [3, 4] and Network Function Virtualization (NFV) [5, 6] technologies are proposed to accelerate the transformation of the network. The promise

of these technologies is to bring more flexibility and agility to the network while creating cost-effective solutions. This will allow Service Providers to become digital businesses.

The SDN concept is presented to decouple the control and forwarding functionalities of network devices by putting the first one on a central unit called controller [7]. This separation makes it possible to control the network from a central application layer simplifying network control and management tasks. And the programmability of the controller accelerates the Service Providers network transformation.

1.2 Motivation and background

In this Ph.D. thesis we study the telecom service management in a controlled environment. This work is based on the management of a connectivity service (MultiProtocol Label Switching (MPLS) Virtual Private Network (VPN)) with a centralized reliability and Quality of Service (QoS) control. Over the last decade, MPLS networks have evolved and become critical for Service Providers. MPLS is used both for Traffic Engineering (TE) allowing optimized resource usage, and for establishing VPNs. This one is an increasingly profitable service for Providers.

As the network transformation is becoming a reality and the digitalization is changing the service management methods, traditional network services are replacing with on-demand network services. On-demand services allow customers to deploy and manage their services independently through a well-defined interface opened to the Service Providers platform. This interface allows different customers to manage their own services each one possessing special features.

For example, to manage a VPN service, a customer might have several types of interactions with the Service Provider platform. For the first case, a customer might request a fully managed VPN interconnecting its sites. For this type of service, the customer owns abstract information about the service and provides a simple service request to the Service Provider. The second case is a customer, with a more professional profile, who monitors the service by retrieving some network metrics sent from the Providers platform. And the third type consists of a more dynamic and open service sold to customers wishing to control all or part of their services. For this type of services, based on the metrics retrieved from the Service Providers platform, the customer re-configures the service.

1.3 Problem statement

In order to offer this freedom to its customers and to provide on-demand network capability, the Service Provider must be able to rely on a dynamic and programmable network control platform. We argue that this platform can be provided by SDN technology. Indeed, the SDN

controller layer might provide an interface to service customers where they might subscribe to a new service, modify an existing service, and retire a service.

A telco network is built on different technologies providing several types of services. These services are used by several customers and a misconfiguration of a customer service can have unexpecteds on the quality of service of other ones. The central visibility provided by the SDN controller allows the operator to manage all services and equipment. Nevertheless, providing a granular management and service control interface on this controller requires presenting an additional Service Management layer beyond the controller through which the Service Provider can manage the service lifecycle and provide a service management interface to its customers.

1.4 Contributions of this thesis

As part of this thesis we present an SDN based framework allowing to both manage the lifecycle of a service and open the service management interface with a fine granularity. The granularity of this interface allows to provide different levels of abstraction to the customer, each one allowing to offer part of the capabilities needed by an on-demand service discussed in Section 1.2.

The following are the main research contributions of this thesis.

— **A double-sided service lifecycle and the associated data model**
We first characterise the applications that might be deployed upon the northbound side of an SDN controller, through their lifecycle. The characterisation rests on a classification of the complexity of the interactions between the outsourced applications and the controller. This leads us to a double-side service lifecycle presenting two articulated points of view: client and operator. The service lifecycle is refined with a data model completing each of its steps.

— **A service management framework based on SDN paradigm**
The service lifecycle analysis gives us a tool to determine all activities an operator should do to manage a service. Based on the operator-side service lifecycle, we propose a framework through which a service model presented to the customer, is transformed to device models deployed on resources. This framework is organized into two orchestrator systems called respectively Service Orchestrator and Resource Orchestrator interconnected by an internal interface. Our approach is illustrated through the analysis of the MPLS VPN service, and a Proof Of Concept (POC) of our framework based on the OpenDaylight controller is proposed.

— **Bring Your Own Control (BYOC) service model**
We exploit the proposed framework by introducing a new and original service control model called Bring Your Own Control (BYOC). It allows the customer or a third party operator to participate in the service lifecycle following various modalities. We analyse the characteristics of interfaces allowing to deploy a BYOC service and we

illustrate our approach through the outsourcing of an Intrusion Prevention System (IPS) service.

1.5 Document structure

In **Chapter 2** we present a state of the art on SDN and NFV technologies. We try to focus our study on SDN control and application layer. We present two classifications of SDN applications. For the first classification we are interested in the functionality of applications and their contribution in the deployment of the controller. And for the second one, we present different types of applications according to the model of the interaction between them and the controller. We discuss in this second classification three types of applications, each one requiring some characteristics at the Northbound Interface (NBI) level.

In **Chapter 3** we discuss the deployment of a network service in SDN environment. For the first part of this chapter, we present the MPLS networks with a rapid analysis of the control and forwarding planes of these networks in the legacy world. This analysis quickly shows which information is used to configure such a service. This information is, for confidential reasons, managed by the operator most of which is not manageable by the customer.

For the second part of this chapter, we analyze the deployment of the MPLS service on the SDN network through the OpenDaylight controller. For this analysis we consider two possibilities: (1) deployment of the service using the third-party applications developed on the controller (the VPN Services project), and (2) deployment of the service using the northern Application Programming Interface (API)s provided by the controller's native functions.

The results obtained during the second part together with the case study discussed in the first part, accentuate the lack of a service management system in the current controllers. This justifies the presentation of a service management framework providing the service management interfaces and managing the service lifecycle.

In order to refine the perimeters of this framework, we firstly discuss a service life cycle studies in **Chapter 4**. This analysis is carried out on two sides: customer and operator.

For the service lifecycle analysis from the client-side perspective, we rely on the classification of applications made in Chapter 2. During this analysis we study the additional steps that each application adds in the lifecycle of a service. And for the analysis of the lifecycle from the operator side view point we study all steps an operator takes during the deployment and management of a service.

At the end of this chapter, we discuss the data model allowing to implement each step of the service lifecycle. This data model is based on a two layered approach analyzing a service provisioning system on two layers: service and device. Based on this analysis, we study the data model of each service lifecycle step, helping to define the internal architecture of the service management framework.

Service lifecycle analysis leads us to present, in **Chapter 5**, the SDN-based service management framework. This framework cuts up all the tasks an operator performs to manage the lifecycle of a service. Through an MPLS VPN service deployment example we detail all of these steps. Part of tasks are carried on the service presented to the client, and part of them on the resources managed by the operator. We organize these two parts into two orchestration systems, called respectively Service Orchestrator and Resource Orchestrator.

In order to analyze the framework's capability in service lifecycle management, we take the example of MPLS VPN service update. With this example we show how the basic APIs provided by an SDN controller can be used by the framework to deploy and manage a requested service.

The presented framework allows us not only to manage the service life cycle but also to open an NBI to the client. This interface allows us to provide different levels of abstraction used by each of lastly discussed three types of applications.

In **Chapter 6**, we present for the first time the new service model: Bring Your Own Control (BYOC). This new service allows a customer or a third party operator to participate in the service lifecycle. This is the practical case of a type 3 application, where the client configures a service based on the events coming up from the controller.

We analyze characteristics of interface allowing to deploy such a BYOC-type service. We present in this chapter the XMPP protocol as a good candidate enabling us to implement this new service model.

In **Chapter 7**, we apply the BYOC model to a network service. For this use case we choose to externalize the control of an IPS. Outsourcing the IPS service control involves implementing the attack detection engine in an external controller, called Guest Controller (GC).

In **Chapter 8**, we point out the main contributions of this thesis and give the research perspectives in relation to BYOC services in SDN/NFV and 5G networks.

Chapter 2

Programming the network

In this chapter we present, firstly, a state of the art on programmable networks. Secondly, we study Software-Defined Networking (SDN) as a technology allowing to control and program network equipment to provide on-demand services. For this analysis we discuss the general architecture of SDN, its layers and its interfaces. Finally, we discuss SDN applications, their different types and the impact that all applications can have on the internal architecture of an SDN controller.

2.1 Technological context

Nowadays Internet whose number of users exceeds 3,7 billions [8], is massively used in all human activities from the professional part to the private ones via academical ones, administrative ones, etc. The infrastructure supporting the Internet services rests on various interconnected communication networks managed by network operators. This continuously growing infrastructure evolves very dynamically and becomes quite huge, complex, and sometimes locally ossified. To configure and maintain their communication networks and provide high-level services, network operators have therefore to deal with a large number of routers, firewalls, switches and various heterogeneous devices with a progressively reduced lifecycle due to the fast hardware and software changes. This growing complexity makes the introduction of a new service or a new protocol together with its configuration, an exceptionally difficult task, because network operators have to translate a high-level service specification to low-level distributed device configurations and next to configure these ones through their Command Line Interface (CLI). This introduction has non trivial side effects leading to frequent network state changes for which operators have to adapt manually the existing network configuration to integrate the new services or protocols. As a result, this manual configuration may lead to frequent misconfigurations [9]. Last but not least, all these may have an adverse effect on the management cost of the operator, Operating Expenditure (OpEx).

Such rigid configuration framework has been quickly perceived as hard limitation preventing operators to cope with their critical scalability and adaptability' requirements. It leaded

however to the development of important research trends the goals of which were to develop new and original capabilities making the network management more flexible and reactive. The mainstream orientation of these research works broadly fall under the banner of "Programming the network". We present below several determinant works in this area.

2.2 Modeling programmable networks

The traditional processing carried out in routers and switches is insufficient to make the network programmable. The agility which should be brought by software to network architectures would in fact require a new dedicated computational capacity. This one appears clearly in the model below [9] (Fig. 2.1).

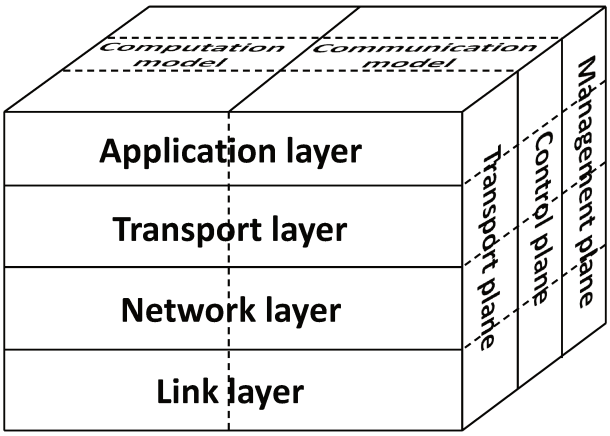

FIGURE 2.1: Global model of programmable networks (source [9])

It presents a communication dimension resting on the four layers (link, network, transport and application) of the internet model, articulated in the same time to the three fundamental planes (transport or data, control or signaling and management) specific to telecommunication operators. Each building block of this communication model resulting from the intersection of a layer and a plane is systematically declined with a computation capability belonging to the second main dimension of the model. This model can be seen as a kind of abstract horizon for programmable networks by clearly asserting the potential presence of the computation capability in each basic block of the classical architecture of the networks. It can however be refined by clarifying the organization of the communication and the computation capabilities, as it appears in the figure below [9] (Fig. 2.2).

This model is very instructive because it highlights the ideal link that should connect the two extremes of the vertical stack: on the lower side, the rigid hardware device belonging to the network infrastructure and on the upper side the flexible and versatile overlay network ideally resulting from the programmability of the network. Between these two extremes, it precisely identifies two fundamental Application Programming Interface (API)s which allow to break the gap between them and make it manageable. It finally states clearly that the computation capability is strictly at the service of the communication one. It technically

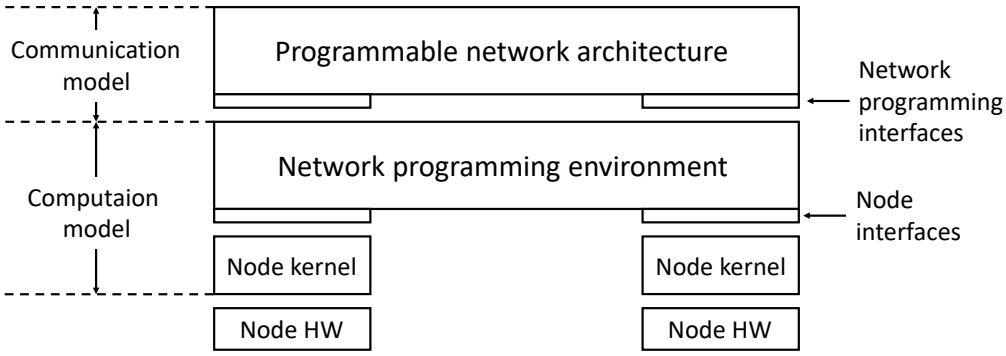

FIGURE 2.2: Computation and communication capabilities of programmable networks (source [9])

distinguishes two kinds of operating systems, a local one and a global one. The former is defined by the « node kernel » and provides primitives allowing to control and to exploit the rough resources offered by the hardware node. The latter called « network programming environment » allows to build the architecture of dedicated networks by resting on the cooperation between the local services offered by the node kernel through its interface.

2.3 Fundamentals of programmable networks

The high performance constraints required for routers in packet switched networks, limit the authorized processing to the sole modification of the packet headers. The strength of this approach is also its weakness because the counterpart of their high performance is their lack of flexibility. The evolution brought by the research on the programmability of the network, has led to the emergence of strong ideas whose relevance can be measured by their intellectual longevity.

The seed of the idea of having APIs allowing a flexible management of the network equipments at least goes back to the OpenSig initiative [9] which aimed to develop and promote standard programmable interfaces crafted on network devices [10]. It is one of the first fundamental steps towards the virtualization of networks the main objectives of which consisted in switching from a strongly coupled network, where the hardware and the software are intimately linked to a network where the hardware and the software are decorrelated. It concretely conducts in keeping the data forwarding capability inside the box while outsourcing the control.

In a general setting the control part of the processing carried out in routers, roughly consists in organizing in a smart and performant way the local forwarding of each received packet while ensuring a global soundness between all the boxes involved in its path. The outsourcing of the control has been designed according different philosophies. One aesthetically nice but extreme vision known as « Active Networks » recommends that each packet may carry in addition to its own data, the code of the process which will be executed at each crossed

network node, to achieve the treatment of the payload carried by the packet [11]. The challenge being that the processing achieved in the various nodes could thus be adapted very finely to the needs of the user or of the application, reaching by this way the desired agility of the network. Despite the great interest received from the academic community, the « active networks » concept has not received much attention from equipment manufacturers mainly because of its complexity in terms of implementation, deployment and management.

It is however interesting to point tight connections between active networks and policy-based management [12, 13]. This one [14] is derived from the Internet Engineering Task Force (IETF) model [15], shown in Fig. 2.3, which introduces two main entities namely the Policy Decision Point (PDP) and the Policy Enforcement Point (PEP). The first one which is the smart part of the architecture acts as a controller the goal of which consists in handling and interpreting policy events, and deciding in accordance with the policy currently applicable, what action should be taken. This one is transmitted to the PEP which has to concretely carry it out. The model may integrate a *Policy Repository* maintaining the set of policies of interest.

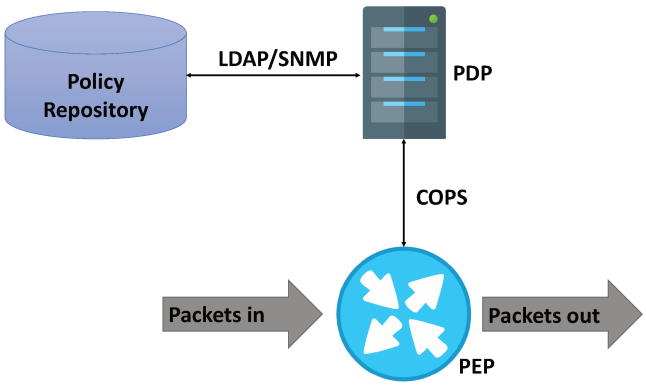

FIGURE 2.3: The IETF policy architecture (source [15])

IETF developed Common Open Policy Service (COPS) [16] as a dedicated protocol to support policy-based QoS control in the IETF framework [15]. COPS is client/server protocol using a query/answer schema from the clients implementing the PEP to the PDP acting as a server. Policy objects transported by COPS are self-identifying and may by the way, be adapted to dedicated proprietary needs. Such orthogonality between the protocol and the associated transported data defines the COPS extensibility capability which is quite useful. COPS is stateful and necessitates a reliable transport layer (TCP).

Outsourcing the control of networks following the spirit of the policy-based management approach can be seen as a reasonable trade-off to make the network programmable. We argue that it is exactly the path investigated by SDN around the SDN controller which may appear very roughly as a kind of network-specialized PDP.

2.4 Software-Defined Networking (SDN)

SDN is presented to change the way networks operate by giving hope to change the current network limitations. It enables simple network data-path programming, allows easier deployment of new protocols and innovative services, opens network virtualization and management by separating the control and data planes [4]. This paradigm is attracting attention by both academia and industry. SDN breaks the vertical integration of the traditional network devices by decoupling the control and data planes, where network devices become a simple forwarding device programmed by a logically centralized application called *controller* or *network operating system*.

2.4.1 Architecture

Figure 2.4 shows a simplified view of the SDN's architecture based on this separation.

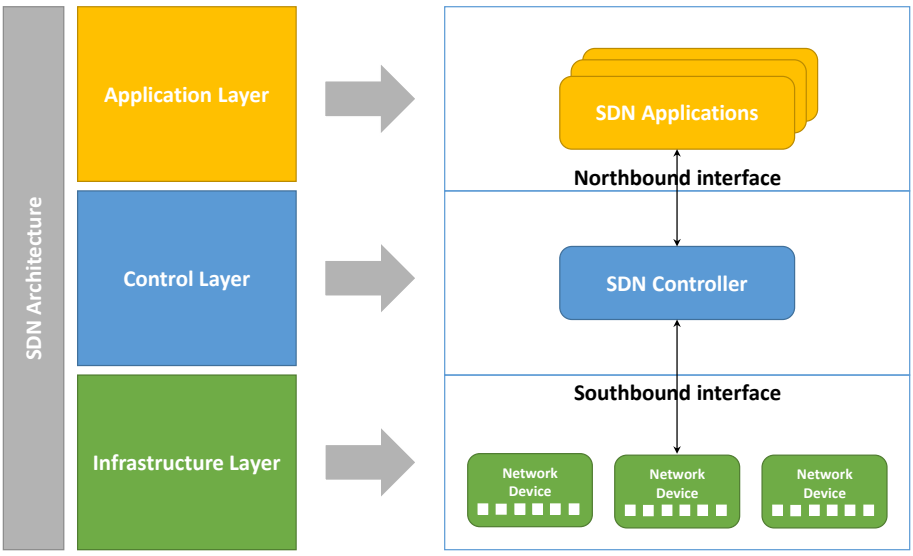

FIGURE 2.4: SDN Architecture

2.4.2 SDN Infrastructure

The lowest layer of the SDN architecture, called *Data* or *Infrastructure plane*, is composed of a series of data forwarding elements executing packet level operations (such as forwarding, dropping, modifying, etc.) based on their local flow tables. These tables contain a set of packet-handling rules, each one composed by three parts: a match rule, actions, and counters. Most of todays SDN datapath devices follow the OpenFlow-based design.

As the OpenFlow-based SDN community is growing up, a large variety of OpenFlow-enabled networking hardware and software switches are presented into the market. Hardware devices are produced for a long range purposes, from the small businesses [17, 18] to

high-class one [19] used for their high switching capacity. Software switches, on the other hand are mostly OpenFlow-enabled applications, and are used to provide the virtual access points in the data centers and to bring virtualized infrastructures.

2.4.3 SDN Southbound Interface (SBI)

The communication between the Infrastructure layer and the control one is assured through a well-defined API called Southbound Interface (SBI), that is the element separating the data and the control plane. This one provides for upper layer a common interface to manage physical or virtual devices by a mixture of different southbound APIs and control plug-ins. The most accepted and implemented of such southbound APIs is OpenFlow [20] standardized by Open Networking Foundation (ONF) [21].

2.4.3.1 OpenFlow Protocol

The SDN paradigm is started by the forwarding and control layer separation idea presented by OpenFlow protocol. This protocol enables flow-based programmability of a network device. Indeed, OpenFlow provides for SDN controller an interface to create, update and delete new entries reactively or proactively.

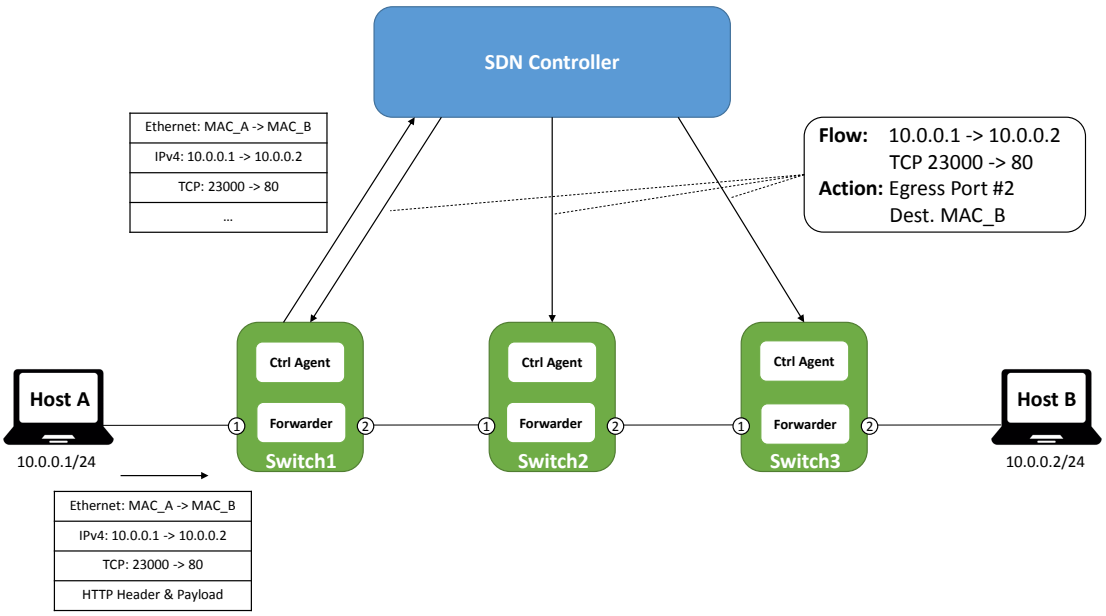

FIGURE 2.5: OpenFlow Protocol in practice

Fig. 2.5 shows OpenFlow protocol in practice, where a SDN controller programs the underlying switches based on new traffic. In this example Host A sends an Hypertext Transfer Protocol (HTTP) Get Web Request to Host B through several OpenFlow switches. By receiving the first packet, Switch 1 looks up in its flow table, if no match for the flow is found, the switch sends an OpenFlow PACKET_IN message to the SDN controller for instructions.

Based on this message, the controller creates a PACKET_OUT message and sends it to the switch. This message is used to add a new entry to the flow table of the switch.

Programming a network device using the OpenFlow can be done in three ways [22]:

— **Reactive flow instantiation**. When a new flow arrives to the switch, it looks up into the flow table and if the relevant action doesn't match with the flow, the switch sends a PACKET_IN message to the controller. In previous example, shown in Fig. 2.5, the SDN controller programs the Switch 1 in a reactive manner.

— **Proactive flow instantiation**. In contrast to the first case, a flow can be defined in advance. In this case when a new flow comes to the switch there is no lookup into the flow table and the action will be done based on a predefined entry. In our example (Fig. 2.5) the follow programing done for two Switches 2 and 3, is a proactive one. The proactive flow instantiation eliminates the latency introduced by controller interrogation.

— **Hybrid flow instantiation**. This one is a combination of two first modes. In our example (Fig. 2.5) for a specific traffic, sent by Host A to Host B, the controller programs the related switches using this method. The Switch 1 is programmed reactively and two other switches (Switch 2 and Switch 3) are programmed proactively. Using hybrid flow instantiation allows to benefit the flexibility of the reactive mode for granular traffics, while saving a low-latency traffic forwarding for the rest of traffic.

2.4.3.2 OpenFlow switch

The most recent OpenFlow Switch (1.5.0) has been defined by ONF [23]. Fig. 2.6 shows three main components of this switch:

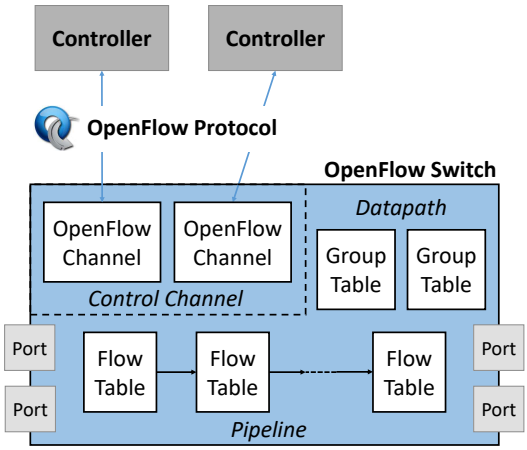

FIGURE 2.6: OpenFlow Switch Components (source [23])

— *OpenFlow Channel* creates a secured channel, over Secure Sockets Layer (SSL), between the switch and a controller. Using this channel, the controller manages the switch via OpenFlow protocol allowing commands and packet to be sent from the controller to the switch.

— *Flow Table* contains a set of flow entries dictating the switch how to process the flow. These entries include match fields, counters and a set of instructions.
— *Group Table* contains a set of group each one having a set of actions.

Fig. 2.7 shows an OpenFlow Switch flow table. Each flow table contains three columns: *rules*, *actions* and *counters* [24]. The rules column contains header fields used to define a flow. For an incoming packet, the switch looks up the flow table, if a rule matches the header of the packet, the related action of action table will be applied to the packet, and finally the counter value will be updated. There are several possible actions to be taken on a packet (Fig. 2.7). The packet can be forwarded to a switch port, it can be sent to the controller, it can be sent to a group table, it can be modified in some fashions, or it can be dropped.

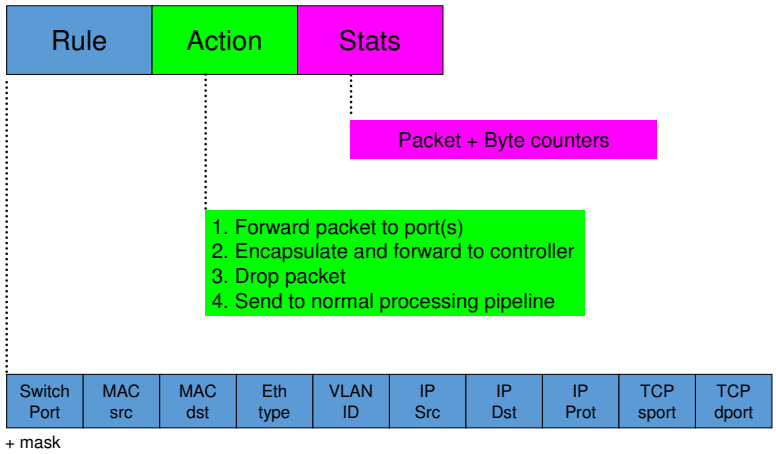

FIGURE 2.7: Flow Table (source [24])

2.4.4 SDN Controller

The *Control plane*, equivalent to the network operating system [25], is the intelligent part of this architecture. It controls the network thanks to its centralized perspective of networks state. On one hand, this logically centralized control simplifies the network configuration, management and evolution through the SBI. On the other hand, it gives an abstract and global view of the underlying infrastructure to the applications through the Northbound Interface (NBI).

While SDN's interest is quite extending in different environments, such as home networks [26], data center network [27], and enterprise networks [28], the number of proposed SDN controller architecture and the implemented functions is also growing up.

Despite this large number, most of existing proposals implement several core network functions. These functions are used by upper layers, such as network applications, to build their own logic. Among the various SDN controller implementations, these logical blocks can be classified into: Topology Manager, Device Manager, Stats Manager, Notification Manager and Shortest Path Forwarding. For instance, a controller should be able to provide a

network topology model to the upper layer applications. It also should be able to receive, process and forward events by creating alarm notifications or state changes.

As mentioned previously, nowadays, numerous commercial and non-commercial communities are developing SDN controllers proposing network applications on top of them. Controllers such as NOX [25], Ryu [29], Trema [30], Floodlight [31], OpenDayLight [32] and ONOS [33] are the top five today's controllers. These controllers implement basic network functions such as topology manager, switch manager, etc. and provide the network programmability to applications via NBI. In order to implement a complex network service on a SDN-based network, service providers face a large number of controllers each one implementing a large number of core services based on a dedicated work flow and specific properties. R. Khondoker et al. [34] tried to solve the problem of selecting the most suitable controller by proposing a decision making template. The decision however requires a deep analysis of each controller and totally depends on the service use case. It is worth to mention that in addition to this miscellaneous controller's world, the NBI abstraction level diversity also emphasizes the challenge.

2.4.5 SDN Northbound Interface (NBI)

In the SDN ecosystem the NBI is the key. This interface allows applications to be independent of a specific implementation. Unlike the southern interface, where we have some standard proposals (OpenFlow [20] and NETCONF [35]), the subject of a common and a standard NBI standard is remained open. Since use cases are still in development, it is still immature to define a standardized NBI. Contrary to its equivalent in the south (SBI), the NBI is a software ecosystem, it means that the standardization of this interface requires more maturity and a well standardized SDN framework. In application ecosystems, implementation is usually the leading engine, while standards emerge later [36].

Open and standard interfaces are essential to promote application portability and interoperability across different control platforms. As illustrated in Table 2.1, existing controllers such as Floodlight, Trema, NOX, ONOS, and OpenDaylight propose and define their own APIs in the north [37]. However, each of them has its own specific definitions.

SDN Controller	Northbound API	Programming language
Floodlight	REST	Java
Trema	ad-hoc API	C, Ruby
NOX	ad-hoc API	C++
ONOS	REST	Java
OpenDaylight	REST, RESTCONF	Java

TABLE 2.1: SDN Controllers and their NBI

This question has been raised several times and a common conclusion is that northbound APIs are indeed important, but it is too early to define a single standard at this time [38, 39,

40]. The experience gained in developing various controllers will certainly be the basis for a common application-level interface.

If we consider the SDN controller as a platform allowing to develop applications on a resource pool, a north API can be compared to the Portable Operating System Interface (POSIX) standard in operating systems [41]. This interface provides generic functions hiding the operational details of the computer hardware. These ordinary functions allow a software to manipulate this hardware by ignoring their technical details. Today, programming languages such as Procera [42] and Frenetic [43] are proposed to follow this logic by providing an abstraction layer on controller functions. The yanc project [44] also offers an abstraction layer simplifying the development of SDN applications. This layer allows programmers to interact with lower-level devices and subsystems through the traditional file system.

It may be concluded that it is unlikely that a single northern interface will emerge as a winner because the requirements for different network applications are quite different. For example, APIs for security applications may be different from routing ones. In parallel with its SDN development work, the ONF has begun a vertical solution in its North Bound Interface Working Group (NBI - WG) to present standardized northbound APIs [45]. This work is still ongoing.

2.5 SDN Applications Analysis

2.5.1 SDN Applications

At the toppest part of the SDN architecture, the *Application layer* programs the network behavior through the NBI offered by the SDN controller. Existing SDN applications implement a large variety of network functionalities from simple one, such as load balancing and routing, to more complex one, such as mobility management in wireless networks. This wide variety of applications is one of the major reasons to raise up the adoption of SDN into current networks. Regardless of this variety most SDN applications can be grouped mainly in five categories [46], including (I) traffic engineering, (II) mobility and wireless, (III) measurement and monitoring, (IV) security and dependability, and (V) data center networking.

2.5.1.1 Traffic engineering

The first group of SDN application consists of proposals that monitor the traffic through the SDN Controller (SDNC) and provide the load balancing and energy consumption optimization. Load balancing as one of the first proposed SDN applications [47] covers a big range of network management tasks, from redirecting clients requests traffic to simplifying the network services placement. For instance, the work [48] proposes the use of wilcard-based for aggregating a group of clients requests based on their Internet Protocol (IP) prefixes. In

the [49] also the network application is used to distribute the network traffic among the available servers based on the network load and computing capacity of servers.

The ability of network load monitoring through the SBI introduces applications such as energy consumption optimization and traffic optimization. The information received from the SBI can be used by specialized optimization algorithms to aim up to 50% of economization of network energy consumption [50] by dynamically scale in/out of the links and devices. This capacity can be leveraged to provision dynamic and scalable of services, such as Virtual Private Network (VPN) [51], and increase network efficiency by optimizing rules placement [52].

2.5.1.2 Mobility and wireless

The programmability of the stack layers of wireless networks [53], and decoupling the wireless protocol definition from the hardware, introduce new wireless features, such as creation of on-demand Wireless Access Point (WAP) [54], load balancing [55], seamless mobility [56] and Quality of Service (QoS) [57] management. These traditionally hard to implement features are implemented by the help of the well-defined logics presented from the SDN controller.

The decoupling of the wireless hardware from its protocol definition provides a software abstraction that allows sharing Media Access Control (MAC) layers in order to provide programmable wireless networks [53].

2.5.1.3 Measurement and monitoring

The detailed visibility provided by centralized logic of the SDN controller, permits to introduce the applications that supply network parameters and statistics for other networking services [58, 4]. These measurement methods can also be used to improve features of the SDN controller, such as overload reduction.

2.5.1.4 Security

The capability of SDN controller in collecting network data and statistics, and allowing applications to actively program the infrastructure layer, introduce works that propose to improve the network security using SDN. In this type of applications, the SDN controller is the network policy enforcement point [59] through which malicious traffic are blocked before entering a specific area of the network. In the same category of applications, the work [60] uses SDN to actively detect and prevent Distributed Denial of Service (DDoS) attacks.

2.5.2 Intuitive classification of SDN applications

As described previously, SDN applications can be analyzed in different categories. In 2.5.1 we categorized the SDN applications based on the functionality they add to the SDN controller. In this section we analyze these applications based on their contribution on the network control life cycle.

SDN applications consist of modules implemented at the top of a SDNC which, thanks to the NBI, configure network resources through the SDNC. This configuration might control the network behavior to offer a network service. Applications which configure the network through a SDNC can be classified in three types. The Fig. 2.8 presents this classification. The first type concerns an application configuring a network service which once initialized and running will not be modified anymore. A "simple site interconnection" through MultiProtocol Label Switching (MPLS), can be a good example for this service. This type of services requires a one direction up-down NBI which can be implemented with a RESTful solution. The second one concerns an application which, firstly, configures a service and, secondly, monitors it during the service life. One example for this model is a network monitoring application which monitors the network via the SDNC in order to generate QoS reports. For example, for assuring the QoS of an MPLS network controlled by the SDNC, this application might calculate the traffic latency between two network endpoints thanks to metrics received from the SDNC. This model requires a bottom-up communication model in the NBI level so that the real-time events can be sent from the controller to the application. Finally, the third type of coordination concerns an application resting on, and usually including, the two previous types and adding specific control treatments executed in the application layer. In this case the application configures the service (type one), listens to network real-time events (type two), and calculates some specific network configurations in order to re-configure the underlying network accordingly (type one).

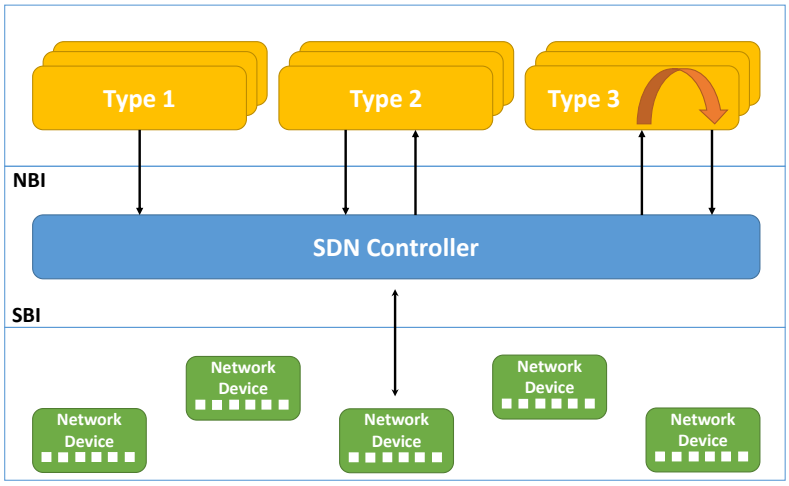

FIGURE 2.8: A general representation of the three types of services implemented by the Guest Controller

2.5.3 Impact of SDN Applications on Controller design

The variety of SDN applications developed at the top of the SDN controller may modify the internal architecture of the controller and its core functions, described in 2.4.4. In this section we analyze some of these applications and their contribution to a SDNcontroller core architecture.

The Aster*x [47] and Plug-n-Serve [49] projects propose HTTP load balancing applications that rely on three functional units implemented in the SDN controller: *"Flow Manager"*, *"Net Manager"* and *"Host Manager"*. The first one, Flow Manager, controls and routes flows based on a specific load-balancing algorithm implemented in this module. This one implements necessary controller core functions and Layer 2 protocols such as Dynamic Host Configuration Protocol (DHCP), Address Resolution Protocol (ARP) and Spanning Tree Protocol (STP). The second module, Net Manager, keeps track of the network topology, link usages and links packet latency. The third module, Host Manager, monitors the state of each HTTP servers and reports it to the Flow Manager. This load-balancing application adds two complementary modules inside the controller, within the core functions.

In work [48] authors implemented a series of load-balancing modules in a NOX controller that partition client traffics between multiple servers. The partitioning algorithm implemented in the controller receives client's Transmission Control Protocol (TCP) connection requests, arriving into the Load Balancer Switch, and balances the load over the servers by generating wildcard rules. The load-balancing application proposed in this work is implemented inside the controller, in addition with other controller's core modules.

Adjusting the set of active network devices in order to save the data center energy consumption is another type of SDN applications. ElasticTree [50], as one of these applications, proposes a "network-wide power manager" increasing the network performance and fault tolerance while minimizing its power consumption. This system implements three main modules: "Optimize", "Power control" and "Routing". The optimizer finds the minimum power network subset, it uses the topology, traffic matrix, and calculates a set of active components to both the power control and routing modules. Power control toggles the power states of elements. The routing chooses paths for all flows and pushes routes into the network. In ElasticTree these modules are implemented as a NOX application inside the controller. The application pulls network statistics (flow and port counters) sends them to the Optimizer module, and based on calculated subset it adjusts flow routes and port status by OpenFlow protocol. In order to toggle the elements, such as active ports, linecards, or entire switches different solutions, such as Simple Network Management Protocol (SNMP) or power over OpenFlow can be used.

In SDN architecture the network topology is one of the information provided to applications. The work [61] proposes the Application-Layer Traffic Optimization (ALTO) protocol as a topology manager component of this architectures. In this work authors propose this protocol to provide an abstract view of the network to the applications which, based on

this informations, can optimize their decision related to service *rendezvous*. ALTO protocol provides network topology by hiding its internal details or policies. The integration of ALTO protocol to the SDN architecture introduces an ALTO server inside the SDN controller through which the controller abstracts the information concerning the routing costs between network nodes. This information will be sent to SDN applications in the form of ALTO maps. These maps are used in different types of applications, such as: data centers, Content Distribution Network (CDN)s, and peer-to-peer applications.

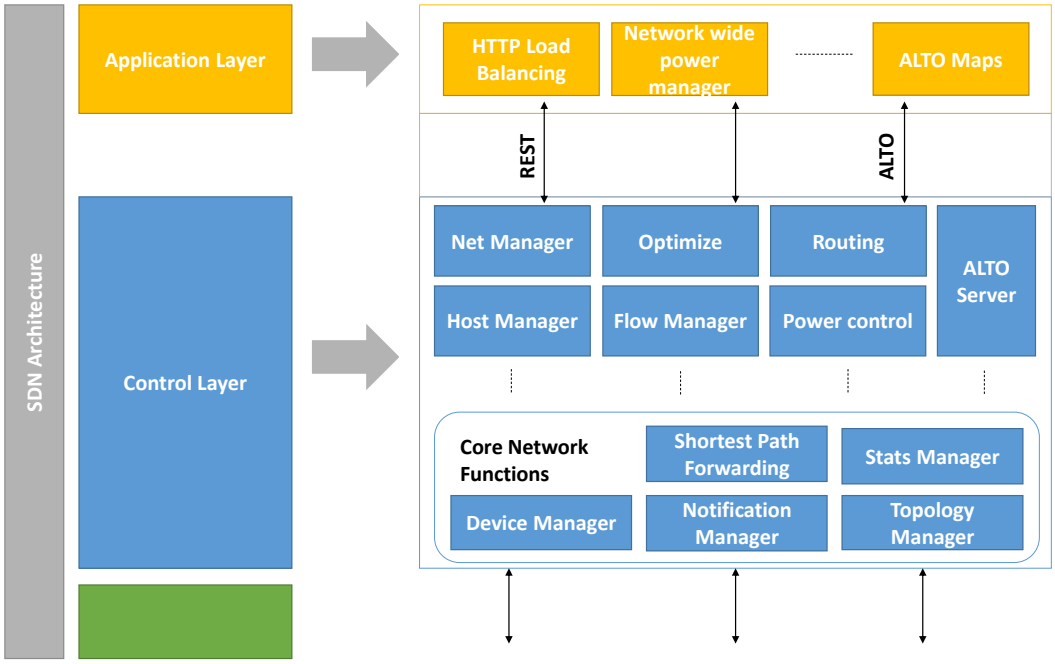

FIGURE 2.9: Additional control functions brought by SDN Applications

Fig. 2.9 illustrates our first analysis of different controllers, their core modules, NBI and applications. Proposing all control function required for implementing a service, may rely on the use of several SDNC. Managing the lifeceycle of a service also requires the use of several APIs proposed through the NBI.

2.6 Network Function Virtualization, an approach to service orchestration

Network Function Virtualization (NFV) is an approach to virtualize and orchestrate network functions, traditionally carried out on dedicated hardware, on Commercial Off-The-Shelf (COTS) hardware platform. This is an important aspect of the SDN particularly studied by service providers, who see here a solution to better adjust the investment according to the needs of their customers.

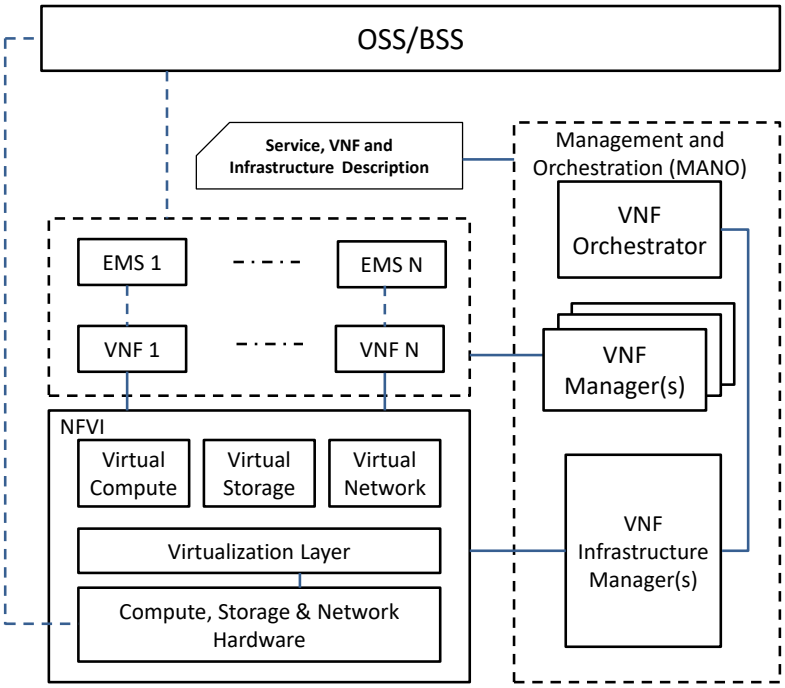

FIGURE 2.10: The MANO architecture proposed by ETSI (source [62])

The reference architecture of NFV, called Management and Orchestration (MANO), is presented by European Telecommunications Standards Institute (ETSI) [62]. Fig. 2.10 illustrates the MANO architecture. This architecture consists of three layers:

— A layer comprising the Virtual Network Function (VNF)s
— An infrastructure layer (computing, storage and network) in which the functions will be executed
— A transverse layer for management and orchestration

The main role of MANO is to manage the lifecycle of VNFs. This architecture includes Elemental Management Systems (EMS) managing each one an individual VNF.

The main advantage of using NFV to deploy and manage VNFs is that the Time To Market (TTM) of NFV-based service is less than a legacy service, thanks to the standard hardware platform used in this technology. The second advantage of NFV is lower Capital Expenditure (CapEx) while standard hardware platforms are usually cheaper than wholesale hardware used on legacy services.

This approach, however, has certain issues. Firstly, in a service operator network, there is no more a single central (data center type) network to manage, but also several networks deployed by different technologies, both physical or virtual. At first glance this seems to be contrary to one of the primary objectives of the SDN: the simplification of network operations. The second problem is the complexity that the diversity of NFV architecture elements brings to the service management system. In order to create and manage a service, several VNFs should be created. These VNFs are configured, each one, by an EMS, the life cycle of which is managed though the Virtual Network Function Manager (VNFM). All VNFs are deployed within an infrastructure managed by the Virtual Infrastructure Manager (VIM).

For the sake of simplicity, we don't mention the license management systems proposed by VNF editors to manage the licensing of their products. In order to manage a service all mentioned systems should be managed by the Orchestrator.

Chapter 3

SDN-based Outsourcing Of A Network Service

In this chapter we present the MPLS networks, its control plan and its data plan. Then, we study the processes and the necessary parameters in order to configure a VPN network. In the second part, we study the deployment of this type of network using SDN. For this analysis, we firstly analyze the management of the VPN network with non-openflow controllers, such as OpenContrail. Then, we analyze the deployment of the VPN network with one of the most developed OpenFlow enabled controller: OpenDaylight.

3.1 Introduction to MPLS networks

MPLS [63] technology supports the separation of traffic flows to create VPNs. It allows the majority of packets to be transferred over Layer 2 rather than Layer 3 of the service provider network. In an MPLS network, the label determines the route that a packet will follow. The label is injected between Layer 2 and Layer 3 headers of the packet. A label is a 32 bits word containing several information:

— Label: 20 bits
— Time-To-Live (TTL): 8 bits
— CoS/EXP: specifies the Class of Service used for the QoS, 3 bits
— BoS: determines if the label is the last one in the label stack (if BoS = 1), 1 bit

3.1.1 MPLS data plan

The path taken by the MPLS packet is called Label Switch Path (LSP). MPLS technology is used by providers to improve their QoS by defining LSPs capable of satisfying Service Level Agreement (SLA) in terms of traffic latency, jitter, packet loss. In general, the MPLS network router is called a Label Switch Router (LSR). Fig. 3.1 shows three main actions done on an MPLS packet by LSRs: *push*, *swap* and *pop*. (A) When a packet enters the MPLS network, it is labeled by an ingress router, called Provider Edge (PE) or Ingress-LSR. (B) This tag is used for all subsequent switching done by the core routers, Provider (P) routers or Transit LSRs,

without ever consulting the IP header. (C) Finally, the network egress PE router, or Egress LSR, removes the tag and transmits the original IP packet to its final destination.

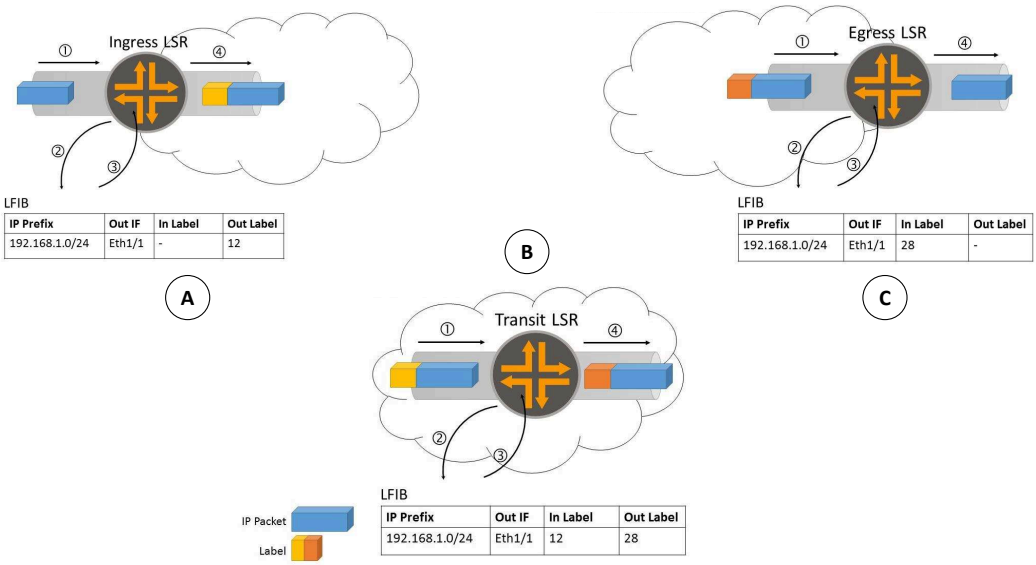

FIGURE 3.1: Label Switch Routers (LSR)s

3.1.2 MPLS control plan

Fig. 3.2 shows the topology of a simple MPLS network. The network is designed with three types of equipment:

— Customer Equipment (CE) is the LAN's gateway from the customer to the core network of the service provider

— Provider Equipment (PE) is the entry point to the core network. The PE labels packets, classifies them and sends them to a LSP. Each PE can be an Ingress or an Egress LSR. We discussed earlier the way this device injects or removes the label of the packet.

— P routers are the core routers of an MPLS network that switch MPLS packets. These devices are Transit LSRs, the operation of whom is discussed earlier.

Each PE can be connected to one or several client sites (Customer Edge (CE)s), Cf. Fig. 3.3. In order to isolate PE-CE traffics and to separate routing tables within PE, an instance of Virtual Routing and Forwarding (VRF) is instantiated for each site, this instance is associated with the interface of the router connected to the CE. The routes that PE receives from the CE are recorded in the appropriate VRF Routing Table. These routes can be propagated by Exterior BGP (eBGP) [64] or Open Shortest Path First (OSPF) [65] protocols. The PE distributes the VPN information via Multiprotocol BGP (MP-BGP) [66] to the other PE within the MPLS network. It also installs the Interior Gateway Protocol (IGP) routes learned from the MPLS backbone in its Global Routing Table.

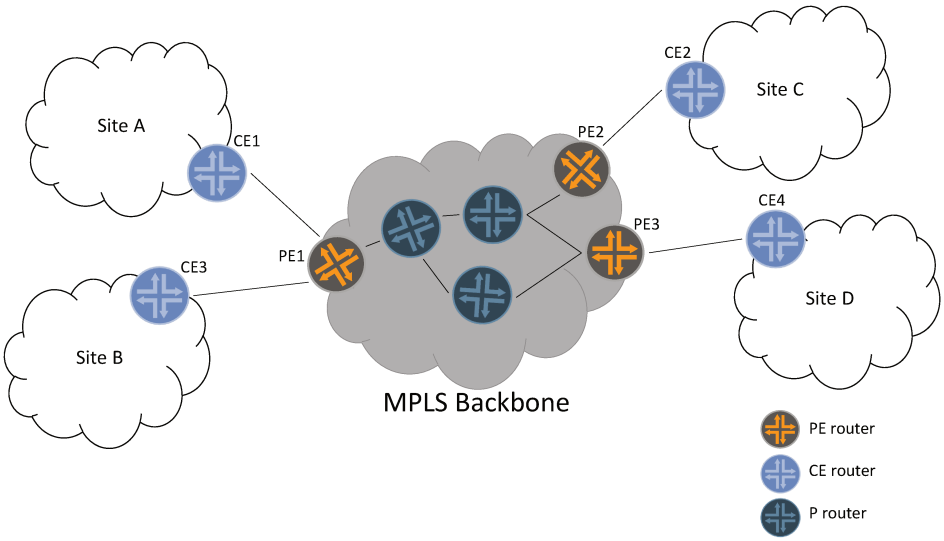

FIGURE 3.2: MPLS Architecture Elements

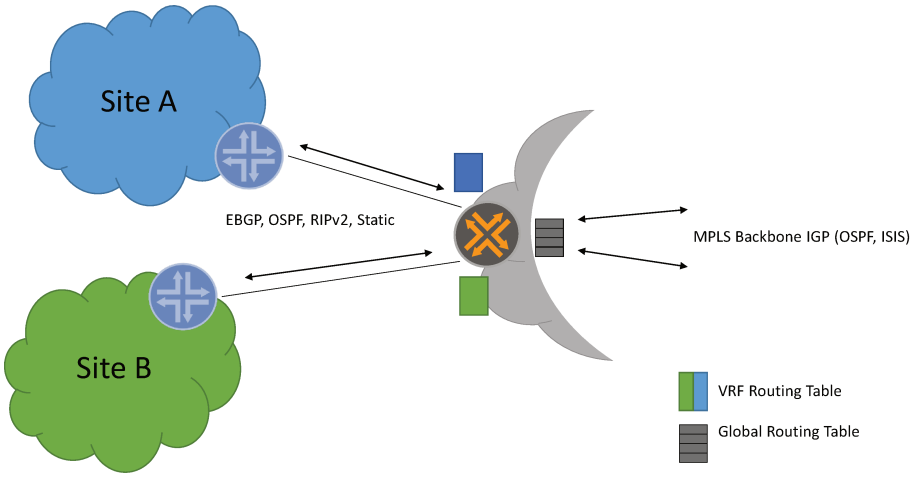

FIGURE 3.3: Routing protocols used by PE

VRF creates an isolation among routing domains while each customer site is assigned to its own VRF. Assuming that two customers are using the same IP address range (ex. 192.168.1.0/24) for their sites, the PE needs a way to keep track of which route belongs to which customer. The Route Distinguisher (RD) is used to isolate sets of routes sent from CEs. A single RD is prepended to each route within a VRF. This value is a unique 64-bits identifier appended to the client IP address (32-bits) propagated by the CE. This combination creates a unique 96-bits address, called VPNv4 address, which is exchanged with other PEs via MP-BGP.

In the MPLS network, we need a parameter allowing to share routes among VRFs. This parameter is Route Target (RT). This is a 64-bits identifier, applied to a VRF allowing to control the export and import of routes among VRFs. Each VRF is associated with one or more RTs and when a VPNv4 route is created, this route is associated with one or more RTs. All routes associated to a given RT must be distributed to all routers that have a VRF associated with the same RT. In other words, a RT is the identifier of a set of VRFs belonging to a VPN. This can be considered as the exchanging route key: everyone that has the same

RT has the right to exchange the route.

These parameters, RD and RT, are used to create the MP-BGP update message exchanged among PE. This 23-Bytes update message contains VPNv4 address, RT and the Label.

3.1.3 MPLS VPN Sample Configuration

We explain the MP-BGP update process through an example. Assume that sites A and C (Fig. 3.2) belong to the same customer. Site A is connected to PE1 and site C is connected to PE2. The VRF customer_1 instantiated on PE1 for the site A has RT = 65000:100 and RD = 65000:100. Given that the Site C is belonging to the same VPN, the VRF customer_1 created on the PE2 also has the RT = 65000:100. Fig. 3.4 shows the processes that each PE (1 and 2) does by receiving a new route announced by the CE1 (site A).

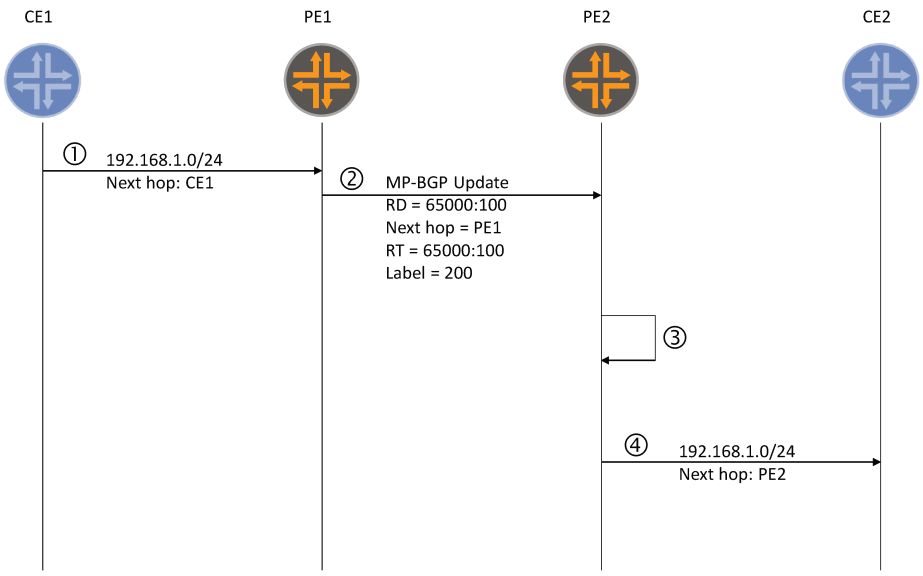

FIGURE 3.4: MP-BGP update process

(1) PE1 receives an IPv4 prefix from CE1. This route is announced by a routing protocol, such as eBGP or OSPF. (2) PE1 translates the route into a VPNv4 address by adding the RD, next hop, RT and label values. It sends the MP-BGP update to other PEs. (3) By receiving the update message, PE2 checks whether the RT (65000:100) is locally configured within any VRF. It translates the VPNv4 prefix into IPv4, adds the prefix into that VRF routing table and updates the CEF table with label 200 for the 192.168.1.0/24 network. (4) PE2 sends the IPv4 prefix to the CE2 thanks to the routing protocol between PE2 and CE2 (eBGP, OSPF, etc.).

For any packet going from CE2 to CE1, the PE2 pushes two labels. The top label is the one learnt by Label Distribution Protocol (LDP) [67] derived from an IGP route, and the lowest one is the label learnt via MP-BGP (label = 200). The former is used across the LSP by LSRs, and the latter is used by the destination PE (PE1). This label allows PE1 to be informed that

the destination CE is connected directly to him. It then pops the label and forwards the IPv4 packet to the CE1.

We drive this configuration example by joining one of customer_1 sites (Site D of Figure 2) to his VPN. Assuming that the MP-BGP of PE4 is already configured and the MPLS backbone IGP is already running on this router. To start the configuration, the service provider creates a dedicated VRF, called customer_1. He adds the RD value on this VRF, we use for this example the RD = 65000:100. For allowing that VRF to distribute and learn routes of this VPN, the RT specified to this customer (65000:100) is configured on the VRF. He then associates the physical interface connected to the CE4 with the initiated VRF. A routing protocol (eBGP, OSPF, etc.) is configured between the VRF and the CE4. This protocol allows to learn Site D network prefix, the information that will be used by PE4 to send the MP-BGP update to other PEs. We discussed earlier this process. By receiving this update, all sites belonging to this customer are able to communicate with Site D.

3.1.4 MPLS VPN Service Management

In the network of a service provider, the parameters used to configure the MPLS network of a client, are not managed neither configured by this client. In other words, for the sake of security, the customer doesn't have any right to configure the PEs connected to these sites or to modify the parameters of his service. For example, if a client A modify the configuration of its VRF by supplying the RTs used for the other VPN (of client B), it can overlap its VPN with that of the client B and put itself in the network of this client. On the other hand, a client can parameter the elements of its sites, for example the addressing plan of its Local Area Network (LAN), and exchange the parameters of its service, ex: service classes (Class of Service (CoS)). Table 1 summarizes parameters of an MPLS VPN service that can be modified by the service provider and its client.

MPLS VPN Parameters	**Service Provider**	**Service Client**
LAN IP address	✗	✓
RT	✓	✗
RD	✓	✗
Autonomous System (AS)	✓	✗
VRF name	✓	✗
Routing protocols	✓	✗
VPN Identifier (ID)	✓	✗

TABLE 3.1: MPLS VPN configuration parameters

3.2 SDN-based MPLS

Decoupling control from the forwarding plane of an OpenFlow-based MPLS network permits to centralize all routing and label distribution protocols (i.e. Border Gateway Protocol (BGP), LDP, etc.) in a logically centralized SDNC. In this architecture forwarding elements deploy uniquely three MPLS actions needed to establish an LSP. However, this architecture is not the only one proposed to deploy SDN-based MPLS. MPLS naturaly decouples the service (i.e. IP unicast) from the transport by LSPs [68]. This decoupling is achieved by encoding instructions (i.e. MPLS lables) in packet headers. In [68] the authors propose to use MPLS as a "key enabler" to deploy SDN. In this work to achieve data centers connectivity authors propose to use the OpenContrail controller that allows to establish overlay network between Virtual Machine (VM)s based on BGP MPLS protocols.

3.2.1 OpenContrail Solution

OpenContrail [69] is an open source controller developed based on BGP and MPLS service architecture. It decouples overlay network from underlay, and control plane from forwarding one by centralizing network policy management.

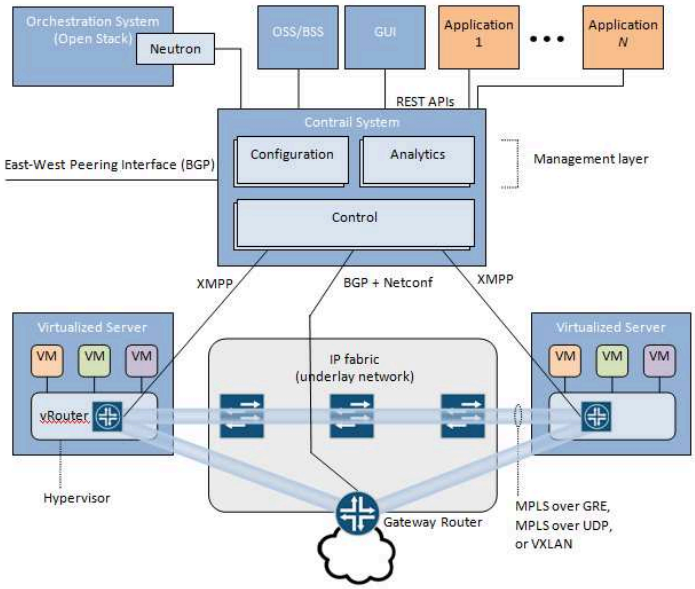

FIGURE 3.5: OpenContrail control plane architecture (source [69])

Fig. 3.5 illustrates the physical architecture of the OpenContrail in production. The architecture of this controller is based on three main nodes:

— **Configuration nodes**, that translate high-level service data-model into a low-level form used to configure network elements.
— **Control nodes**, that propagate low-level model to and from network elements.
— **Analytics nodes**, that capture real-time data from network elements, abstract it, and present it in a form suitable for applications to consume.

In OpenContrail architecture, the forwarding plane is implemented using vRouters. To configure vRouters, control node uses the Extensible Messaging and Presence Protocol (XMPP) based interface. These nodes communicate with other control nodes using their east-west interfaces implemented in BGP.

OpenContrail is a suitable solution used to interconnect VMs within one or multiple data centers. VMs are initiated inside a compute node that are general-purpose virtualized servers. Each compute node contains a vRouter implementing the forwarding plane of the OpenContrail architecture. Each VM contains one or several Virtual Network Interface Cart (vNIC)s, and each vNIC is connected to a vRouter's tap interface. In this architecture, the link connecting the VM to the tap interface is equivalent to the CE-PE link of VPN service. This interface is dynamically created as soon as the VM is spawned.

In OpenContrail proposed architecture, XMPP performs the same function as MP-BGP in signaling overlay networks. After joining spawning a VM the vRouter assigns an MPLS label to the related tap interface connected to the VM. Next, it advertises the network prefix and the label to the control node, using a XMPP Publish Request message. This message, going from the vRouter to the Control node is equivalent to a BGP update from both semantic and structural point of view. The Control node, acts like a Route Reflector (RR) that centralizes route signaling and sends routes from one vRouter to another one by an XMPP Update Notification.

Proposed OpenContrail architecture and its complementary blocs provide a turnkey solution suitable for public and private clouds. However, this solution covers mostly data center oriented use cases based on specific forwarding devices, called vRouters. The XMPP-based interface used by the latter creates "technological dependency" and reduces the openness of the solution, while the XMPP is not a commune interface usable by other existing SDN controllers.

3.2.2 OpenFlow-based MPLS Networks

Configuring and controlling MPLS networks via SDN controllers is one of challenges. Nowadays, SDN controllers propose to externalize MPLS control plane inside modules some of which are implemented within the controller or application layer. The work [70] proposes the implementation of MPLS Traffic Engineering (MPLS-TE) and MPLS-based VPN using OpenFlow and NOX. In this work authors discuss how the implementation of MPLS control plane becomes simple thanks to the consistent and up to date topology map of the controller. This externalization is done though the network applications implemented at the top of the SDN controllers, applications like Traffic Engineering (TE), Routing, VPN, Discovery, and Label Distribution. This work is an initiative to SDN-based MPLS networks, and the internal architecture of SDN controller and APIs allowing to configure an MPLS network is not explained in details.

In order to analyze the internal architecture of controllers proposing the deployment of MPLS networks, and also to study SDN APIs allowing to configure the underlying network, we try to orient our studies on one of the most developed open source SDN controllers, OpenDaylight.

3.2.2.1 MPLS Networks in OpenDaylight controller

With its large development community and it various projects the OpenDaylight controller is one of the most popular controllers in the SDN academic and open source world. OpenDaylight is an open source project under the Linux Foundation based on the microservices architecture. In this architecture, each core function of the controller is a microservice which can be activated or deactivated dynamically. OpenDaylight supports a large number of network protocols beyond OpenFlow, such as NETCONF, OVSDB, BGP, and SNMP.

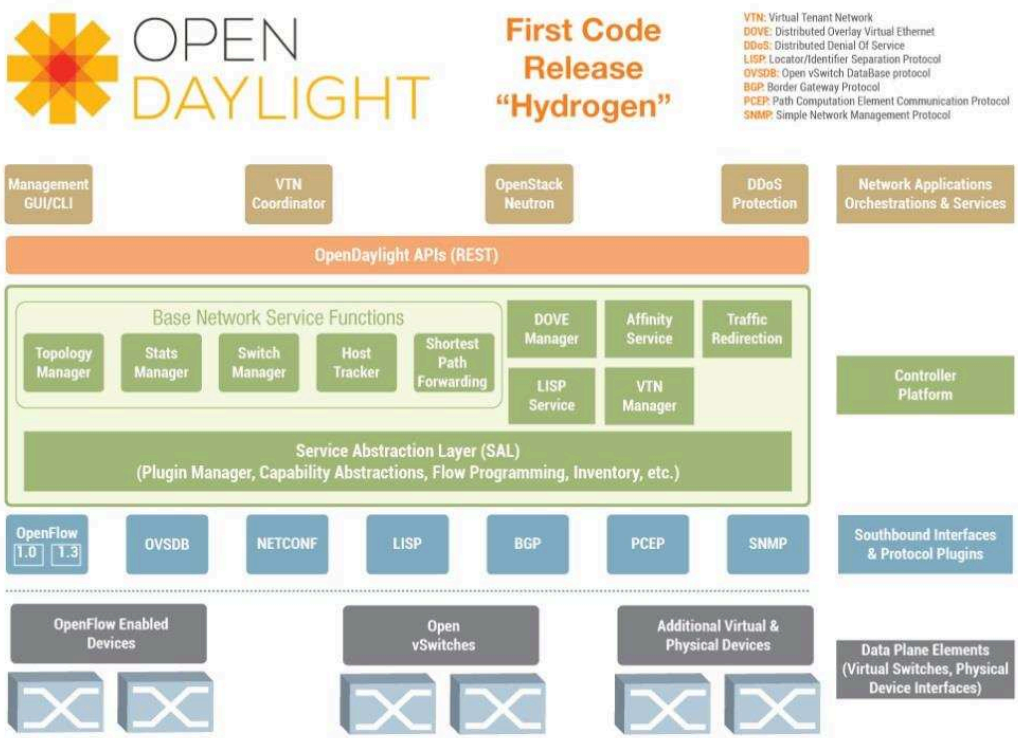

FIGURE 3.6: OpenDaylight Architecture (source [32])

The architecture of OpenDaylight is illustrated in Fig. 3.6. The central part of OpenDaylight is the Service Abstraction Layer (SAL) that connects the protocols of the SBI to the Base Service Network Functions which contains the following components:

— Topology Manager: handles information about the network topology. At the boot time, it builds the topology of the network based on the notifications coming from the switches. This topology can be updated according to notifications coming from other modules like Device Manager and Switch Manager.

— Statistics Manager: sends statistics request to resources (switches), collects statistics and stores them in a data base. This component implements an API to retrieve information like meter, table, flow, etc.
— Forwarding Rules Manager: manages forwarding rules, resolves conflict and validates rules. This module communicates via the SBI with the equipment. It deploys the new rules in switches.
— Switch Manager: provides information for nodes (network equipment) and connectors (ports). When the controller discovers the new device, it stores the parameters in this module. The latter provides an API for retrieving information about nodes and discovered links.
— Host Tracker: provides information on end devices. This information can be switch type, port type, network address, etc. To retrieve this information, the Host Tracker uses ARP. The database of this module can also be manually enriched via the north API.
— Inventory Manager: retrieves the information about the switches and its ports for keeping its database up to date.

These modules provide some APIs at the NBI level allowing to program the controller to install flows. Using this API, the modules implemented in application layer are able to control the behavior of each equipment separately. Programming an OpenFlow switch consists of a tuple of two rules: match and action. Using this tuple, for each incoming packet, the controller can decide if the packet should be treated, if so which action should be applied on this packet. This programming capacity allows the appearance of a large API allowing to manipulate almost every packet types, including MPLS packets.

3.2.2.2 OpenDaylight native MPLS API

OpenDaylight proposes native APIs to make three MPLS actions, *PUSH*, *POP*, and *SWAP*, each LSR might apply on the packet. Using these APIs, the NBI application may install flows on the Ingress LSR pushing tag on a packet entering MPLS network. It may install flows on Transit LSRs allowing to swap tags along and routing the packet along the LSP. This application may install a flow on Egress LSR to send the packet to its final destination by popping the tag.

In order to program the underlying networks behavior via this native API, the application needs to have a detailed perspective of the network and its topology, and a control on specified MPLS labels. Table 3.2 summarizes parameters that an application may control using the OpenDaylight native API.

3.2.2.3 OpenDaylight VPN Service project

Apart from OpenDaylight core functions, additional modules can be developed in this controller. In order to deploy a specific service, these modules benefit the information provided

MPLS VPN Parameters	OpenDaylight Native MPLS API
LAN IP address	✓
RT	N/A
RD	N/A
AS	N/A
VRF name	N/A
Routing protocols	N/A
VPN ID	✓
MPLS labels	✓

TABLE 3.2: MPLS VPN configuration parameters accessible via OpenDaylight API

by Base Service Network Functions.

Among these add-ons, the VPN Service project proposes to integrate the new L3 Service Functions into the this controller to deploy the L2 / L3 VPN service for data centers using BGP and MPLS technologies.

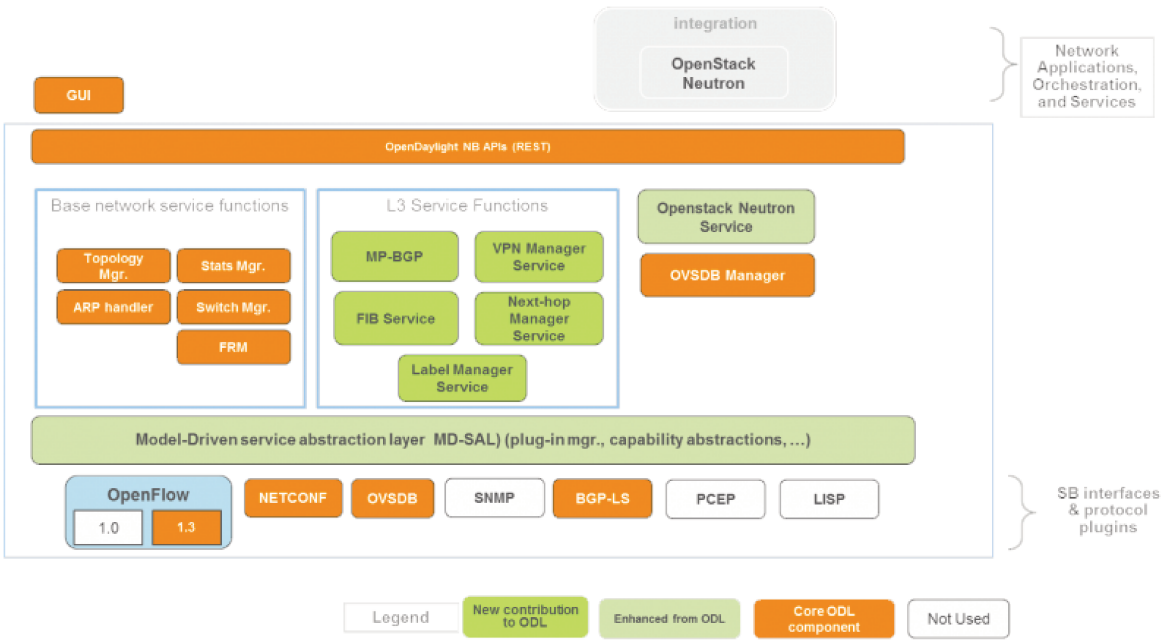

FIGURE 3.7: OpenDaylight VPN Service project (source [32])

The architecture of this proposal is illustrated in Fig. 3.7 The complementary modules proposed by this project are:

— MP-BGP Routing Stack: executes an MP-BGP instant inside the controller. It peers with legacies routers and exchanges routing information with them.

— MPLS Label Manager: manages the allocation of VPN labels. These labels are distributed via BGP.

— L3 VPN Manager: Implements VRFs inside the controller.

— NextHop Manager: Maintains information about the NextHop used to transmit packets. This information is also used by the Open vSwitch (OVS) to encapsulate outgoing

packets.

— FIB Service: Maintains the state of the Forwarding Information Base (FIB) that associates routes and NextHop for each VRF. This information is sent to the OVS by OpenFlow.

The VPN Service project provides the NBI APIs for deploying an L3 VPN for the Data Center (DC) Cloud environment.

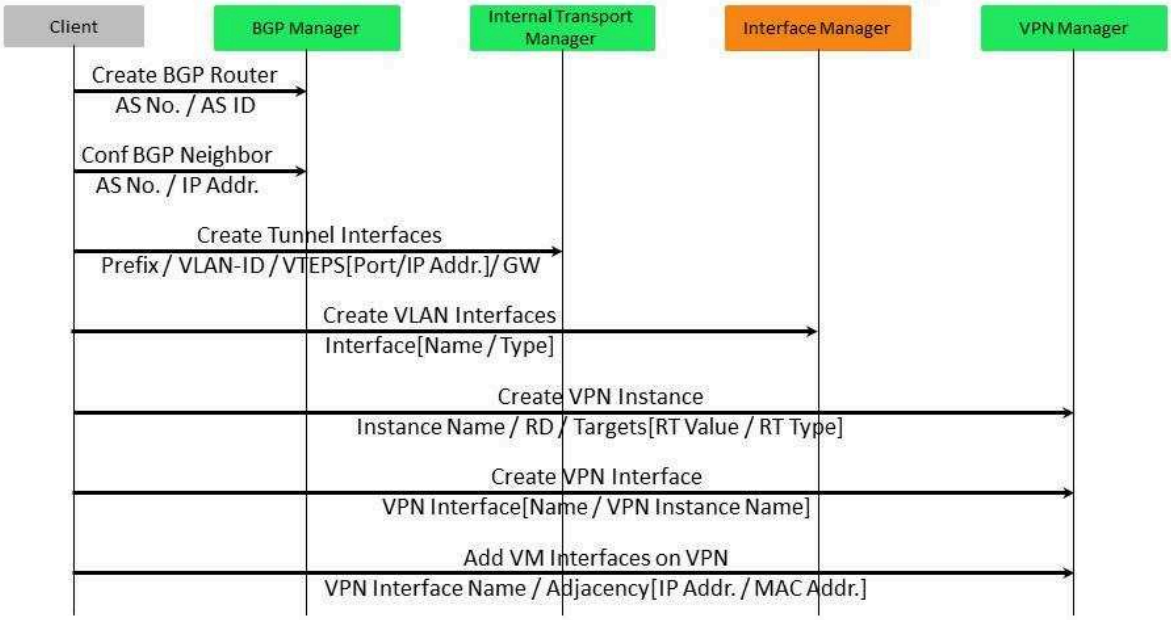

FIGURE 3.8: VPN Configuration Using OpenDaylight VPN Service projetct

Fig. 3.8 shows the VPN service configuration steps through OpenDaylight VPN Service. For this, the user must have a detailed knowledge of the service he wishes to deploy. The deployment of this service takes place via several internal modules of the controller through several steps:

1. First a BGP router must be created via the BGP Manager, and the user must configure the BGP Neighbors.
2. In the second step, via the Internal Transport Manager module, the user creates the tunnels between the switches. The ITM creates and maintains the Generic Routing Encapsulation (GRE) and Virtual Extensible LAN (VXLAN) tunnels between OpenFlow switches.
3. The third step is creating Virtual LAN (VLAN) interfaces for each VM connected to the VPN. The creation of the VLAN interface is implemented by the Interface Manager module of OpenDaylight.
4. Once VLAN interfaces are created, the user creates VPN instance containing all information about the VPN, such as RD, acRT, etc. The VPN instance creation is done through the VPN Manager module.

5. Then, via the API of the VPN Manager a VPN interface is created. In this step, the VPN ID is registered in the database and the VRF is created.
6. Finally, VMs interfaces are added on VPN interface.

As discussed in this example, the VPN Service project and its interfaces are rich enough to deploy a VPN service via OpenDaylight. Nevertheless, in order to create a "sufficient" complex VPN service, the user must manage the information concerning the service, its sites and its equipments. Table 3.3 summarizes the information that a user should manage using this project.

MPLS VPN Parameters	**OpenDaylight Native MPLS API**
LAN IP address	✓
RT	✓
RD	✓
AS	✓
VRF name	✓
Routing protocols	✓
VPN ID	✓
MPLS labels	✓

TABLE 3.3: MPLS VPN configuration parameters accessible via OpenDaylight VPN Service project

As it is shown in this table, the amount of manageable data, information about its BGP routers (local AS number and identifier) information about its BGP neighbor (AS number and IP address) information on VPN (VPN ID, RD and RT) and etc. can quickly increase exponentially. This large amount of information can make the service management more complex and reduce the QoS. It is important to note that the SDN controller of an operator manages a set of services on different network equipment shared between several clients. That means that, for the sake of security, most of the listed information will not be made available to the customer.

3.3 Outsourcing problematics

Decoupling control plane and data plane of MPLS networks and outsourcing the second one into a controller brings several benefits in terms of service management, service agility and QoS [70, 71]. Centralized control layer offers a service management interface available to the customer. Nevertheless, this outsourcing and openness can create several challenges.

The MPLS backbone is a shared environment among the customers of an operator. To deploy a VPN network, as discussed recently, the operator configures a set of devices, situated in the core and the edge of the network. This equipment, mostly provide several services to customers in parallel. The latter ones use the VPN connection as a reliable means of transaction of their confidential data.

Outsourcing the control plane to an SDNC brings a lot of visibility on the traffic exchanged within the network. It is through this controller that a customer can create an on-demand service and manage this service dynamically. Tables 3.2 and 3.3 present in detail the information sent from the NBI to deploy a VPN service. These NBIs are proposed by two solutions 3.2.2.2 and 3.2.2.3. The granularity of this information gives to customer more freedom in the creation and management of his service. Moreover, beyond this freedom a customer having access to the NBI not only can modify the parameters of his own service (i.e. VPN) but also it can modify the parameters concerning the services of other customers.

In order to control the customers access to the services managed by the controller, while maintaining service management agility, we propose to introduce a service management framework beyond the SDNC. From bottom-up perspective, this framework provides an NBI abstracting all rich SDNC functions and control complexities, discussed in Section 2.5.3. We strengthen this framework by adding the question of the access of the client to managed resources and services. Indeed, this framework must be able to provide a NBI of variable granularity, through which the customer is able to manage all three types of services discussed in Section 2.5.2:

— **Type-1 applications:** The service abstraction model brought by the framework's NBI allows the customers side application to configure a service with minimum of information communicated between the application and the framework. The restricted access provided by the framework prevent unintentional or intentional data leaking and service misconfiguration.

— **Type-2 applications:** On the southern side, internal blocks of the framework receive upcoming network events directly from the resources, or indirectly through the SDNC. On the northern side, these blocks open up an API to applications allowing them to subscribe to some metrics used for monitoring reasons. Based on receiving network events, these metrics are calculated by framework internal blocks and are sent to the appropriate application.

— **Type-3 applications:** The controlled access to SDN based functions assured by the framework provides not only a service management API, but also a service control one, opened to the customers application. The thin granularity control API allows customers to have a low-level access to network resources via the framework. Using this API customers receive upcoming network events sent by devices, based of which they reconfigure the service.

In order to provide a framework able to implement mentioned APIs, we need to analyze the service lifecycle in details. This analyze gives rise to all internal blocks of the framework and all steps they may take, from presenting a high-level service and control API to deploying a low-level resource allocation and configuration.

Chapter 4

Service lifecycle and Service Data Model

Contents

1.1 Thesis context 1
1.2 Motivation and background 2
1.3 Problem statement 2
1.4 Contributions of this thesis 3
1.5 Document structure 4

In order to propose different level of abstractions on the top of the service providers platform a service orchestrator should be integrated at the top of the SDNC. This system allows third party actor, called user or customer, to participate to all or part of his network service lifecyle.

Nowadays, orchestrating an infrastructure based on SDN technology is one of the SDN challenges. This problematic has at our knowledge been once addressed by Tail-F which proposes a partial proprietary solution [72]. In order to reduce the Operation Support System (OSS) cost and also the TTM of services, Tail-F Network Control System (NCS) [73] introduces an abstraction layer on the top of the NBI in order to implement different services, including layer 2 or layer 3 VPN. It addresses an automated chain from the service request, on the one hand, to the device configuration deployment in the network, on the other hand. To transform the informal service model to a formal one this solution uses the YANG data model [74]. The service model is mapped into device configurations as a data model transformation. The proposed work doesn't however cover all management phases of the service lifecycle, specially service monitoring, maintenance, etc. , and also it doesn't study the possibility of opening up a control interface to a third party actor. Due to the proprietary nature of this product it is not possible to precisely analyze its internal structure.

We present in this chapter a comprehensive solution to this problematic by identifying a reasonable set of capabilities of the NBI of the SDN together with the associated API. Our first contribution rests on a global analysis of an abstract model of the operator platform articulated to a generic but simple service lifecycle, described in Section 4.1, which takes into account the view of the user together with that of the operator. Tackling the service lifecycle

following these two views simplifies the service abstraction design. The first viewpoint allows us to identify the APIs structuring the NBI and shared by both actors (operator and service consumer).

The second part of this chapter, Section 4.2, is dedicated to service data model analysis, where we describe data model(s) used on each service lifecycle phases, both for client side and operator side.

4.1 Service Lifecycle

The ability of managing the lifecycle of a service is essential to implement it in an operator platform. Existing service lifecycle frameworks are oriented on human-driven services. For example, if a client needs to introduce or change an existing service, the operator has to configure the service manually. This manual configuration may take hours or sometimes days. It may therefore significantly affect the operators OpEx. It clearly appears that the operator has to re-think about its service implementation in order to provision dynamically and also to develop on-demand services. There are proposals in order to enhance new on-demand network resource provisioning. For instance, the GYESERS project [75], proposed a complex service lifecycle model for on-demand service provisioning.

This model includes five typical stages, namely service requests/SLA negotiation, composition/reservation, deployment/register and synchronization, operation (monitoring), decommissioning. The main drawback of this model rests on its inherent complexity. We argue this one may be reduced by splitting the global service lifecycle in two complementary and manageable viewpoints: client and operator view. Each one of both views captures only the information useful for the associated actor. The global view may however be obtained by composing the two partial views.

In a fully virtualized network based on SDN, the SDNC is administratively managed by the Service Operator. This one provides a programmable interface, called NBI, at the top of this SDNC allowing the OSS and Service Client applications to configure on-demand services. In order to analyze the service lifecycle, and to propose a global model of service lifecycle in this kind of networks, the application classification analysis is necessary. In Section 2.5.2 we made an intuitive classification of SDN applications. This classification allows us to analyze the service lifecycle on both operator and client sides.

4.1.1 Client side Service Lifecycle

Based on the application classification discussed in Section 2.5.2, we analyze the client side service lifecycle of the three main application types.

4.1.1.1 Client side Service Lifecycle managed by Type-1 applications

Type-1 applications consist of applications creating a network service using the NBI. This category doesn't monitor neither modify the service based on upcoming network events. Fig. 4.1 illustrates the client-side service lifecycle managed by this type of applications, containing two main steps:

— Service creation: The application specifies the service characteristics it needs, it negotiates the associated SLA which will be available for limited duration and finally it requests a new service creation. In the reminder of the text we will mark it **[Service creation]**.

— Service retirement: The application retires the service at the end of the negotiated duration. This step defines the end of the service life. In the reminder of the text we will mark it **[Service retirement]**.

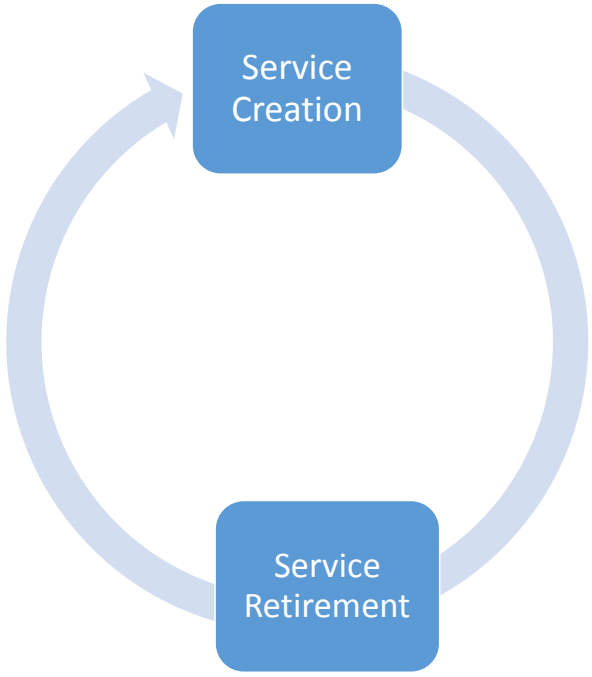

FIGURE 4.1: Client side Service Lifecycle of Type-1 applications

4.1.1.2 Client side Service Lifecycle managed by Type-2 applications

This category of applications, takes advantage of events coming up from NBI to monitor the service. It is worth to note that this service may be created by the same application which monitors the service. Fig. 4.2 illustrates the supplementary step added by this type of applications to the client-side service lifecycle. This lifecycle contains three main steps:

— Service creation: **[Service creation]** cf. Section 4.1.1.1.

— Service monitoring: Once created, the service may be used by the client for the negotiated duration. During this time some network and service parameters will be

monitored thanks to the upcoming events and statics sent from the SDNC to the application. In the reminder of the text we will mark it **[Service monitoring]**.

— Service retirement: **[Service retirement]** cf. Section 4.1.1.1.

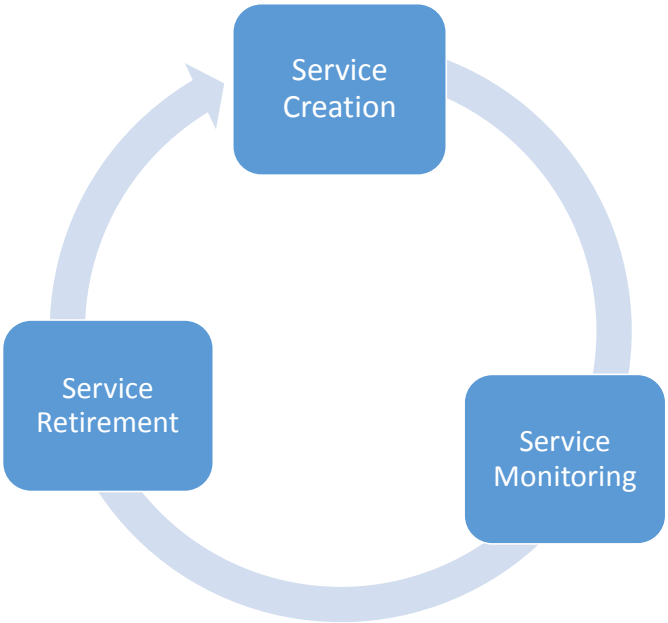

FIGURE 4.2: Client side Service Lifecycle of Type-2 applications

4.1.1.3 Client side Service Lifecycle managed by Type-3 applications

In a more complex case, an application may create the service through the NBI, monitors the service through this interface, and based on upcoming events reconfigure the network via the SDNC. This type of control adds a retroactive step to the client-side service lifecycle. This one is illustrated in Fig. 4.3 and contains four main steps:

— Service creation: **[Service creation]** cf. Section 4.1.1.1.
— Service monitoring: **[Service monitoring]** cf. Section 4.1.1.2.
— Service modification: Upcoming events and statistics may trigger an algorithm implemented inside the application (implemented at the top of the SDNC), the output of which reconfigures the underlying network resources through the SDNC. In the reminder of the text we will mark it **[Service modification]**.
— Service retirement: **[Service retirement]** cf. Section 4.1.1.1.

4.1.1.4 Global Client-side Service Lifecycle

A global client-side service lifecycle is illustrated in Fig. 4.4. This model contains all previous steps needed to manage three types of applications, discussed earlier. We introduce to this

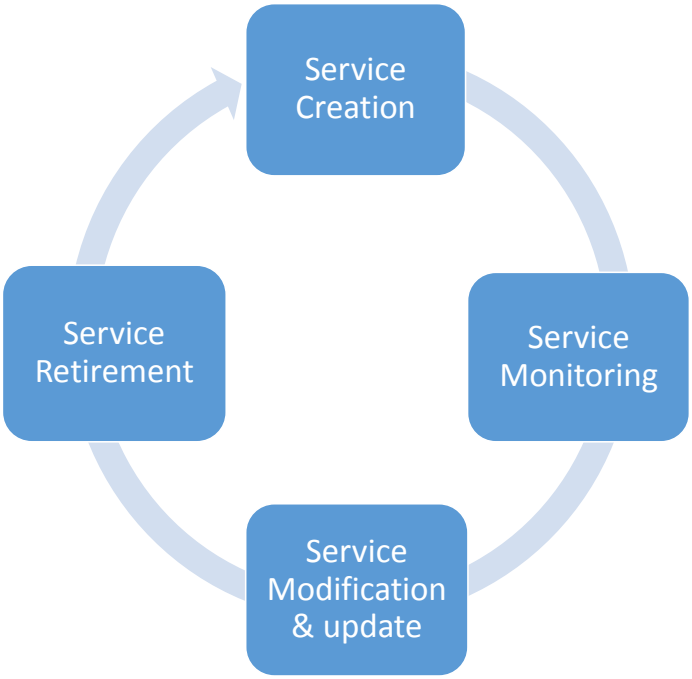

FIGURE 4.3: Client side Service Lifecycle of Type-3 applications

model a new step that is triggered by the operator side operations. This service lifecycle consists of five main steps:

— Service creation: **[Service creation]** cf. Section 4.1.1.1.
— Service monitoring: **[Service monitoring]** cf. Section 4.1.1.2.
— Service modification: **[Service modification]** cf. Section 4.1.1.3.
— Service modification and update: The management of the operator's network may lead to the update of the service. This update can be issued because of a problem occurring during the service consummation or a modification of the network infrastructure. This update may be minimal, such as modifying a rule in one of the underlying devices, or it may impact the previous steps, with consequences on the service creation and/or on the service consummation.
— Service retirement: **[Service retirement]** cf. Section 4.1.1.1.

4.1.2 Operator Side Service Lifecycle

The Operator-side service lifecycle is illustrated in Fig. 4.5. This service lifecycle consists of six main steps:

— Service request: Once a service creation or modification request arrives from the users' service portal (through the NBI), the request manager negotiates the SLA and a high level service specification in order to implement it. It is worth noting that before agreeing the SLA the operator should ensure that the existing resources can cope with the requested service at the time it will be deployed. In case of unavailability, the request will be enqueued.

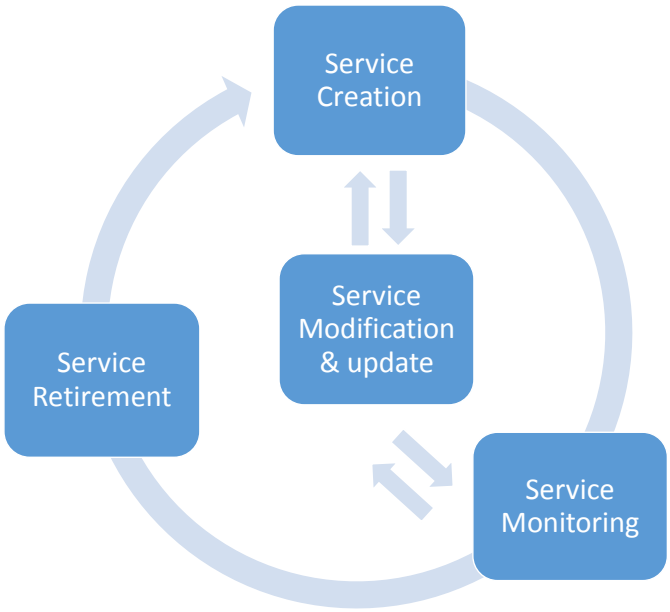

FIGURE 4.4: Global Client Side Service Lifecycle

— Service decomposition, compilation: The high level model of requested service is decomposed into several elementary service models which are sent to the service compiler. The compiler generates a set of network resource configurations which compose that service.
— Service configuration: Based on the previous set of network resource configurations, several instances of corresponding virtual resources will be created, initialized and reserved [1]. The requested service can then be implemented on these created virtual resources by deploying network resource configurations generated by the compiler.
— Service maintain, monitoring and operation: Once a service is implemented, its availability, performance and capacity should be maintained automatically. In parallel, a service log manager will monitor all service lifecycle.
— Service update: During the service exploitation the network infrastructure may necessitate changes due to some execution problems or technical evolution requirements, etc. It leads to update which may impact the service in different way. The update may be transparent to the service or it may require to re-initiate a part of the first steps of the service lifecycle.
— Service retirement: the service configuration will be retired from the infrastructure as soon as a retirement request arrives to the system. The service retirement issued by the operator is out of the scope of this work.

1. This aspect is not mentioned in this figure because it falls outside of the scope of the service lifecycle.

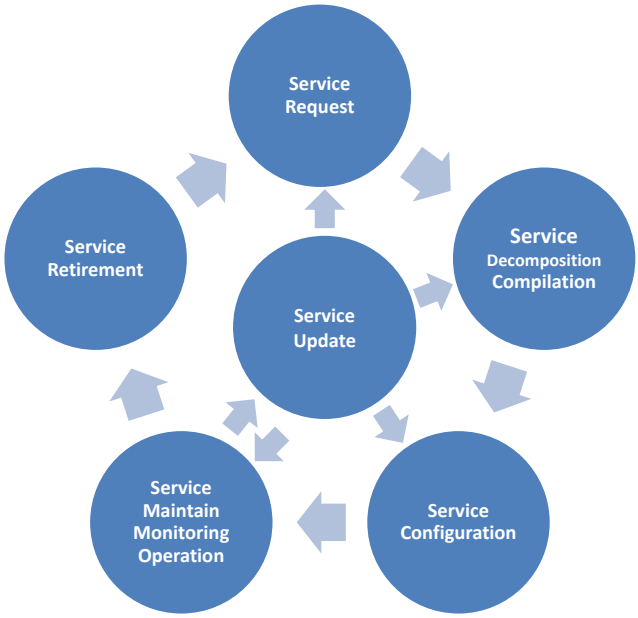

FIGURE 4.5: Operator Side Service Lifecycle

We argue that this service lifecycle on the provider side is generic enough to manage the three types of applications, discussed in Section 2.5.2.

4.1.3 The global view

The global service lifecycle is the combination of both service lifecycles explained in Sections 4.1.1 and 4.1.2. The Fig. 4.6 illustrates the interactions between these two service lifecycles. During the service run-time the client and the operator interact with each other using the NBI. This interface interconnects different phases of each part, as described below:

— Service Creation and Modification ↔ Service Request, Decomposition, Compilation and Configuration: the client-side service creation and specification phase leads to three first phases of the service lifecycle in the operator side; service request, decomposition, compilation and configuration.
— Service Monitoring ↔ Service Maintain, Monitoring and Operating: client-side service monitoring, which is executed during the service consummation, is in parallel with operator-side service maintain, monitoring and operation.
— Service Update ↔ Service Update: operator-side service maintain, monitoring and operation phase may lead to the service update phase in the client-side service lifecycle.
— Service Retirement ↔ Service Retirement: In the end of the service life, the client-side service retirement phase will be executed in parallel with the operator-side service retirement.

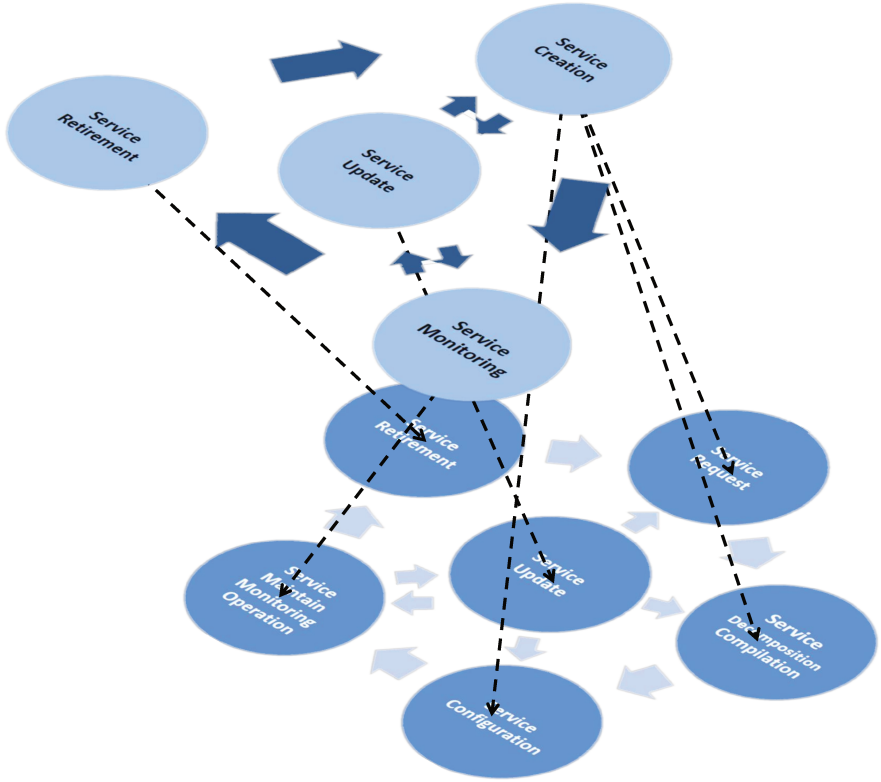

FIGURE 4.6: Global Service lifecycle

4.2 Service Data Model

Current service management systems are specified based on hard-coding of service definitions. This hard-coding model makes the definition of new service provisioning and new service activations relatively expensive. Moreover, these systems are containing numerous manual steps coupled with static hard-coded processes. This explodes the complexity in new service changes and new network element integration.

4.2.1 A two-layered approach

Service management models are mostly decomposed into two layers:

— Service layer described in natural languages

— Device layer the description of which is based on different APIs and CLIs

In order to deploy a service, mapping between these two layers will take place. Since the service layer is specified in an informal language, there will be a gap between services' descriptions and their equivalent on device configurations.

To overcome these issues, a unified data modeling should be used for both layers. On the service layer side, this well-defined model brings formalism into the model definition. On the resource side, it removes the complexity of device APIs. And the model transformation aspect allows the use of formal transformation methods permitting the conversion of formal data values from a source data model to a destination one.

To formalize data model on each layer and the transformation allowing to map one layer onto the other one, Moberg et al. [76] proposed to use the YANG data model [74] as the formal modeling language. Using the unified YANG modeling for both service and device layers was also previously proposed by Wallin et al. [77].

These three elements required to model a service, (i.e. service layer, device layer and transformation model), are illustrated in [76] where the authors specify with this approach IP VPN services. At the service layer the model consists in a formal description of VPN services presented to the customer, and derived from a list of VPN service parameters including BGP AS number, VPN name and a list of VPN endpoints (CE and/or PE). At the second layer the device model is the set of device configurations corresponding to a VPN service. This one is defined by all configurations done on PEs connected to all requested endpoints. This information includes PE IP address, RT, RD, BGP AS number, etc. Finally, for the third element which is the transformation template mapping one layer on the other one, the authors propose the use of a declarative model. In the example the template is based on Extensible Markup Language (XML) which according to service model parameters, generates a device model.

4.2.2 Applying the two-layered model approach on Service Lifecycle

Proposed model based on YANG data modeling language brings dynamicity and agility to the service management system. Its modular aspect allows to reduce the new service creation and modification costs. In this section we apply this model on the proposed service lifecycle, discussed in Section 4.1. This analysis aims to introduce a model allowing to formalize service lifecycle phases and their respective data models. An example of this analysis is presented in the Table 4.1 at the end of this Section.

4.2.2.1 Applying two-layered model on client side service lifecycle

The client side service representation is a minimal model containing informal information about the customers service. All steps of client side service lifecycle are relying on the negotiated service model, i. e. the service layer model of two-layered approach (Section 4.2.1) that is a representation of the service and its components. From the operator side point of view, data model used by this service lifecycle rests on the service layer model.

4.2.2.2 Applying two-layered model on operator side service lifecycle

The integration of new services and updating the service delivery platform involves the creation and updating data models used by the client side service lifecycle. Contrary to the client side, the operator side service lifecycle relies on several data models.

Following we describe service model(s) used during each step of operator side service lifecycle, discussed in Section 4.1.2:

- **Service Request:** to negotiate the service with the customer the operator relies on the service layer model. This model is the same as the model used on the client side service lifecycle. For example, for a negotiated VPN service, both Service Request step and client side service lifecycle, will use the same service layer model. An example of this model is discussed in Section 4.2.1 [76].
- **Service Decomposition and Compilation:** this step receives on the one hand, the service layer model and generates, on the other hand, device configuration sets. Comparing to proposed two-layered approach, this phases is equivalent to the intermediate layer transforming data models. A service layer model can be a fusion of several service models that for the sake of simplicity are merged into a global model. During the decomposition step this global model is broken down into elementary service models which are used in compilation step. They are finally transformed in sets of device models. The transformation of models can be done through two methods:
  - **Declarative method** is a straightforward template that makes a one to one mapping of a source data model of parameters to a destination one. For example a service model describing a VPN can be transformed to device configuration sets by one-to-one mapping of values given within the service model. In this case it is sufficient that the transformer retrieves required values from the first model to construct the device model based on a given template.
  - **Imperative method** is defined by an algorithmic expression used to map a data model to a second one. Usually this model contains some dynamic parameters, e.g. an unlimited list of interfaces. An example for this model can be a VPN service model in which each client's site, i.e. CE, has different number of up-links (1..n) connected to different number of PEs (1..m). In this case the transformation is not a simple one-to-one mapping any more, but rather an algorithmic process (here a loop) that creates one device model per service model.

  Using one of these methods, i.e. declarative or imperative data transformation, has its own advantage or drawback, hardly one of these methods would be superior than the other [78, 79]. We argue that the choice of the transformation method used on compilation phase rests on the service model, its related device model and the granularity of parameters within each model.
- **Service configuration:** to configure a resource, the device model generated by the transformation method of the previous step (i.e. compilation) is used. If this model is generated into the same data model known by the network element, no transformation method should be used. Otherwise another data transformation action should be done on the device model transforming the original device model to a network element compatible one. It is worth noting that since this transformation is a one-to-one mapping task, the data transformation can be done with the declarative method.
- **Service maintain, monitoring and operation:** since the service maintain and operation process is directly done on network elements, the data model used for this phase

is device model. Although the service model used for the monitoring task of this phase relies on the nature of the monitored resource. For example, if the service engineers and operators need to monitor the status of a resource they might use a monitoring method such as SNMP, BGP signaling-based monitoring [80], etc. the result of which is described in device model. Otherwise, if a service customer needs to monitor its service, e.g. monitoring the latency of two endpoints of a VPN connection, the monitoring information sent from the operator to the customer is transformed to a service data model. This bottom-up transformation can be done by declarative or imperative method.

— **Service update:** updating a service consists in updating network elements configurations, hence the data model used on this phase is a device data model. Nevertheless this update may derive a modification on the service model represented to the customer. In this case, at the end of the service update process, a new service model will be generated based on the final state of network elements. This new model is the result of the bottom-up data transformation done through of declarative or imperative methods.

— **Service retirement:** decommissioning a service is made up of all tasks done to remove service related configurations from network elements, and eventually to remove the resource itself. In order to remove device configurations, device data models are used. But, during the retirement phase the service model is also used. The data model transformation done in this phase entirely depends on the source of retirement process. Indeed, if the service retirement is requested from the customer, hence the request is arrived from the client side described in a service model. Consequently, the service model - device model transformation is a top-bottom model transformation. Otherwise, if the service retirement is triggered by the service operator, a new service model should be represented to the customer. This one requires a bottom-up model transformation done with one of explained methods.

Service lifecycle	Phase	Data model(s)	Transformation method(s)
Client side	Service creation	SM	N/A
	Service monitoring	SM	N/A
	Service update	SM	N/A
	Service retirement	SM	N/A
Operator side	Service creation	SM	N/A
	Service configuration	DM	D
	Service maintain, monitoring and operation	SM - DM	D - I
	Service update	SM - DM	D - I
	Service monitoring	SM - DM	D - I
	Service retirement	SM - DM	D - I

TABLE 4.1: Service lifecycle phases and their related data models and transformation methods

SM - Service model, DM - Device model, D - Declarative, I - Imperative

4.3 Conclusion

In this chapter we conducted an analysis of service lifecycle in an SDN-based ecosystem. This analysis has led us to two general service lifecycles: client-side and service-side. On the first side we discussed how an application implementing network services using an SDNC can contribute to the client-side service lifecycle. For this reason, for each application category discussed in Section 2.5.2, we presented a client-side service lifecycle model, by discussing additional steps that each category may add to this model. Finally, a global client-side service lifecycle is presented. This global model, contains all steps needed to deploy each type of applications. We also presented a global model concerning the operator-side service lifecycle. It represents the model that an operator may take into account to manage a service from the service negotiation to the service retirement phases.

In the second part of this chapter we discussed the data model used by each service lifecycle phase. Through an example we explained in details the manner in which a data model is transformed from a source model into a destination one.

We argue that presenting a service lifecycle model on one side, allows the implementation of a global SDN orchestration model managed by an operator. On the other side, this model will help us to understand the behavior of applications. In this way it will simplify the specification of the NBI in forthcoming studies. Presenting the data model also describes in details the behavior of the management system on each service lifecycle step. It also permits the definition of operational blocks and their relations allowing to implement the operator side service lifecycle.

Chapter 5

An SDN-based Framework For Service Provisioning

Contents

2.1 Technological context **7**
2.2 Modeling programmable networks **8**
2.3 Fundamentals of programmable networks **9**
2.4 Software-Defined Networking (SDN) **11**
2.4.1 Architecture 11
2.4.2 SDN Infrastructure 11
2.4.3 SDN Southbound Interface (SBI) 12
2.4.4 SDN Controller 14
2.4.5 SDN Northbound Interface (NBI) 15
2.5 SDN Applications Analysis **16**
2.5.1 SDN Applications 16
2.5.2 Intuitive classification of SDN applications 18
2.5.3 Impact of SDN Applications on Controller design 19
2.6 Network Function Virtualization, an approach to service orchestration . . **20**

In this chapter we present a framework involving a minimal set of functions required to manage any network service conform to the service lifecycle model presented in the previous chapter. We organize this set of functions in two orchestrators, one dedicated exclusively to the management of the resources: the *resource orchestrator*, and the other one grouping the remaining functions: the *service orchestrator*. The general framework structuring the internal architecture of SDN is presented in Section 5.2 and illustrated with an example. This framework is externally limited by NBI and SBI and internally clarifies the border between the two orchestrators by identifying an internal interface between them, called the middle interface.

5.1 Illustrating Service Deployment Example

The operator side service lifecycle is presented in Section 4.1.2. This model represents all processes an operator may take into account to manage a service. We introduce a service and resource management platform which encapsulates an SDNC and provides through other functional modules, the capabilities to implement each step of the service lifecycle presented before. Fig. 5.1 illustrates this platform with the involved modules together with special data required and generated by each module. It shows the diversity of information needed to manage a service automatically.

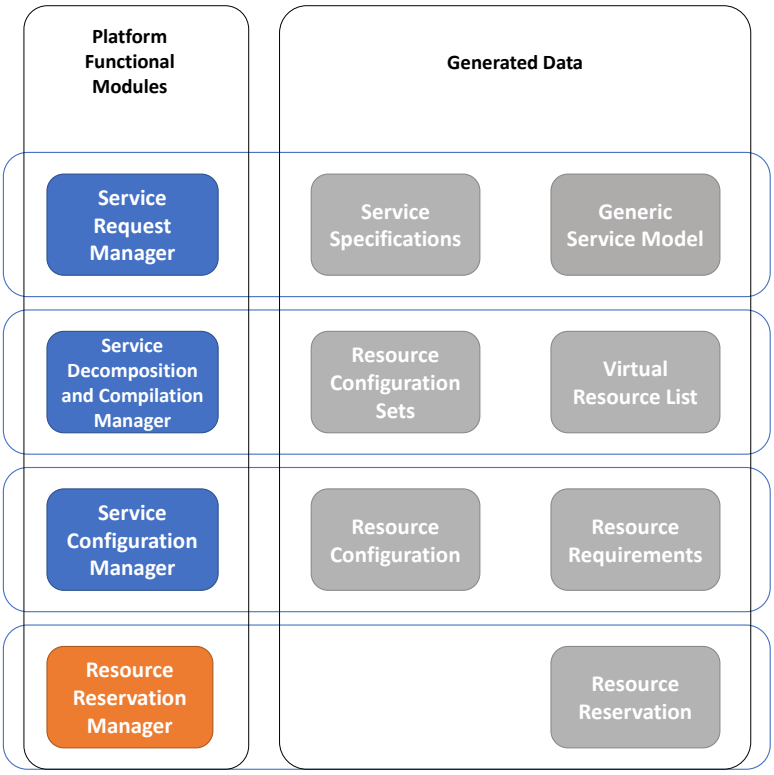

FIGURE 5.1: Functional modules of the service and resource management platform.

We will detail the different modules of this platform in the next section. We prefer now to illustrate this model by describing the main processes through the example of a VPN service connecting two remote sites of a client connected to physical routers: PE1 and PE2. In MPLS networks, each CE is connected to a VRF instance hosted in a PE. In our example we call these instances, (i.e. VRFs) respectively vRouter1 and vRouter2. The first step of the service lifecycle which consists in the "Service Creation" gives rise in the nominal case to a call flow the details of which are presented in Fig. 5.2.

In the first step (arrow 1 of Fig. 5.2), the client requests a VPN service and negotiates the service specifications, such as QoS, bandwidth, etc. This negotiation is done through the NBI with the Service Request Manager (SRM). In this case, the negotiated service model can be presented as a general virtual router with two interfaces, each one connected to one

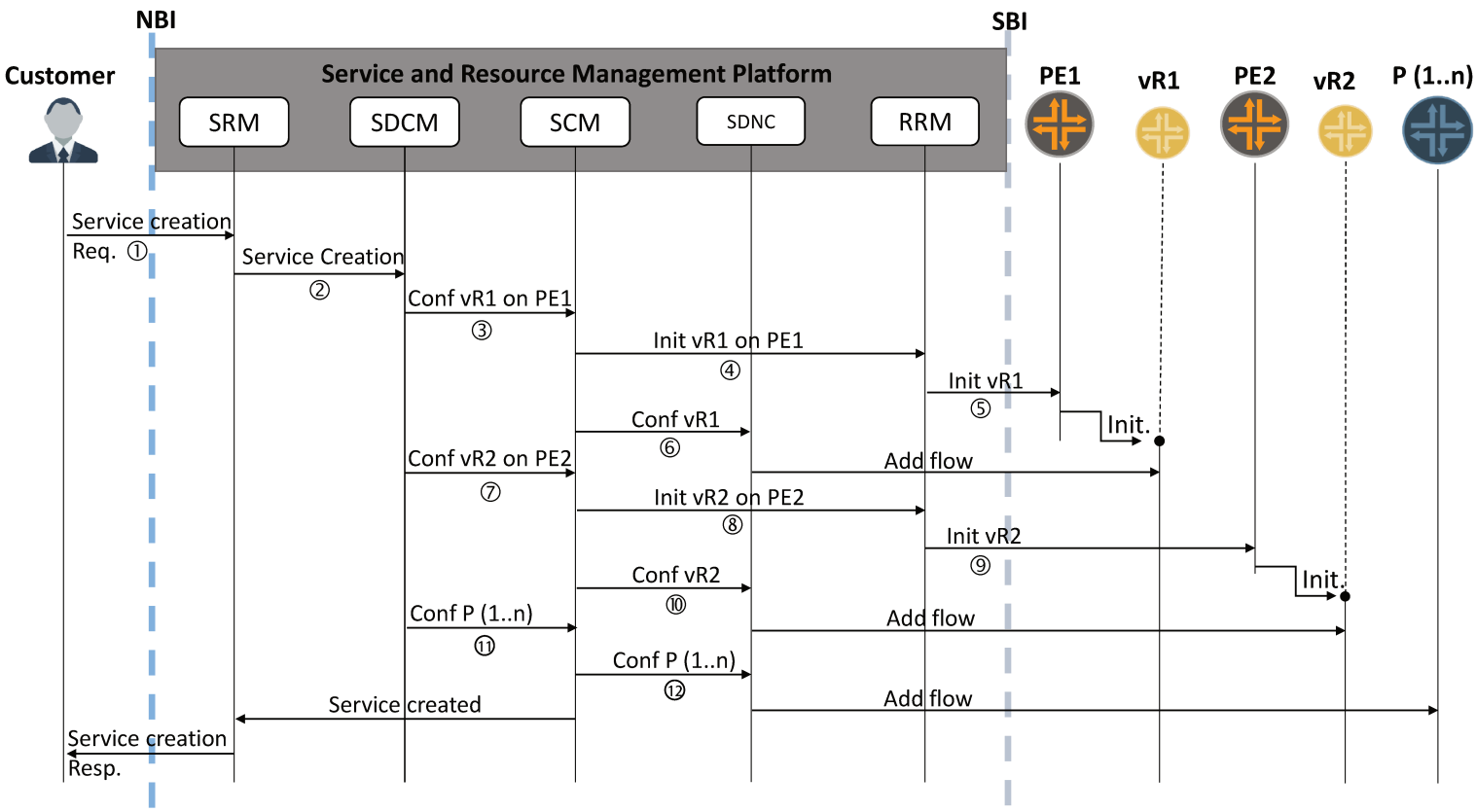

FIGURE 5.2: Service Deployment Call Flow

of the clients' sites. Once the service specification is finalized, the SRM sends the negotiated service model to the Service Decomposition Compilation Manager (SDCM) (arrow 2 of Fig. 5.2). This unit contains a service configuration compiler which translates a general service described in a simplified model into one or several technical models. In our case the compiler will analyze the requested service model and determine firstly the physical routers (PEs) on which virtual routers should be instantiated. In order to determine these PE nodes, this module uses a topology data base integrated inside the framework. This module generates secondly three configuration instructions (e.g. an instruction described in XML or YANG data model [74]) used to configure each virtual router. Once physical routers are localized and their configurations generated, this information will be sent to the Service Compilation Manager (SCM) (arrows 3, 7 and 11 of Fig. 5.2). If the new configuration requires the initiation of one or several virtual resources, or an upgrade of an existing resource, this module may use the Resource Reservation Manager (RRM) unit to initiate and reserve these resources (arrows 4 and 8 of Fig. 5.2). In our example, assuming that the requested service is asked to interconnect two new sites, two virtual routers will be required: vRouter1 and vRouter2. In this case the RRM initiates these two virtual resources (arrows 5 and 9 of Fig. 5.2). Once these resources are initiated an interface will be established between them and the SDNC.

Once resources are initiated and the control interface is established, the SCM will use the

SDNC NBI to program these resources by recently generated instructions (arrows 6, 10, 12 of Fig. 5.2). Finally at the end of the service deployment, the client will be informed about the service implementation through the SRM (arrows "Service Creation Resp." of Fig. 5.2).

5.2 Orchestrator-based SDN Framework

Service management processes, as illustrated in the previous example, can be divided into two more generic families: the first one managing all steps executing service based tasks from service negotiation to service configuration and service monitoring, and the second one managing all resource based operations. These two families managing together all operator-side service lifecycle (discussed in 4.1.2) can be represented as a framework illustrated in Fig. 5.3. The model is composed of two main orchestration layers:

— Service Orchestrator (SO)
— Resource Orchestrator (RO)

The "Service Orchestrator" will be dedicated to the service part operations and is conform to the operator side service lifecycle, cf. Section 4.1.2:

— Service Request
— Service Compilation/Decomposition
— Service Configuration
— Service Maintain
— Service Monitoring
— Service Retirement

The "Resource Orchestrator" will manage resource part operations:

— Resource Reservation
— Resource Monitoring

Service Orchestrator (SO): This orchestrator receives service orders and initiates the service lifecycle by decomposing complex and high level service requests to elementary service models. These models allow to derive the type and the size of resources needed to implement that service. The SO will demand the virtual resource reservation from the lower layer and deploy the service configuration on the virtual resources through an SDNC.

Resource Orchestrator (RO): This orchestrator, which manages physical resources, will reserve and initiate virtual resources. It maintains and monitors physical resources states using the southbound interface.

5.2.1 Internal structure of the Service Orchestrator

As mentioned in Fig. 5.3, the first orchestrator, SO, contains five main modules:

— SRM
— SDCM
— SCM

— SDNC
— Service Monitoring Manager (SMM)

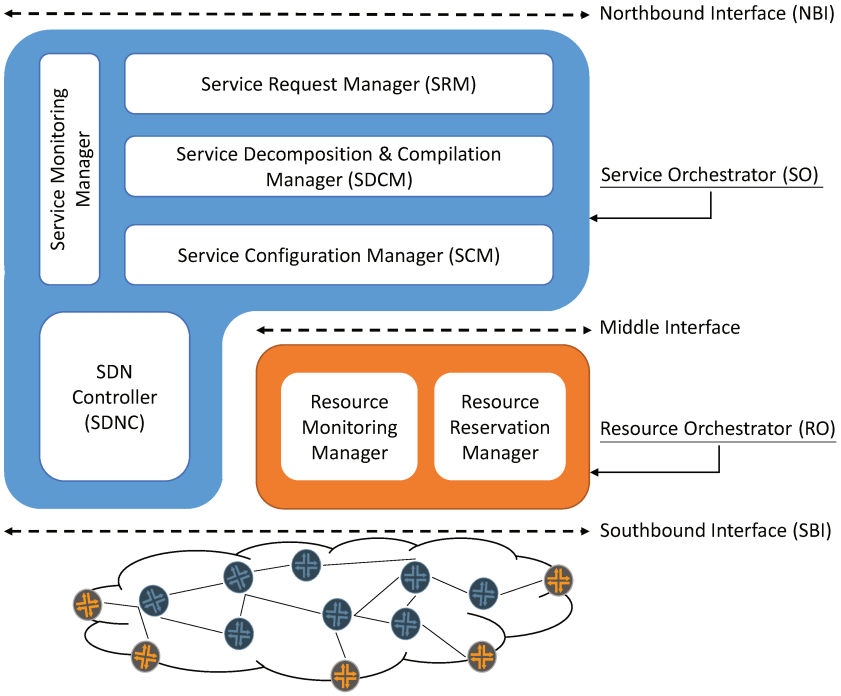

FIGURE 5.3: Proposed orchestrator-based SDN Framework

5.2.1.1 Service Request Manager (SRM)

The Service Request Manager handles clients' service requests and negotiates the service specifications, such as IP range of its LAN networks, SLA and QoS. On the one hand, this module communicates with the service portal, client-side application, or an external orchestrator through the NBI. On the other hand, it communicates with the internal modules all the requests coming from applications upon NBI, starting with SDCM.

5.2.1.2 Service Decomposition and Compilation Manager (SDCM)

A negotiated service can be an elementary service model known by the orchestrator or a composition of several elementary ones. In the first step, the SDCM breaks-down all received service demands to one or several elementary service models which are resource configuration models. In order to discover the elementary service models and to break down a complex model to elementary ones, the SDCM uses a built-in Service Model Database (SMDB) describing elementary service characteristics, such as network devices types, the number of network interfaces needed on each device, etc.

We describe this process by tackling the example of a customer, called Customer 1, wishing to update its VPN network by adding a new site, called Site D. The Fig. 5.4 illustrates the physical architecture of this service in details. Two existing sites (A and B) are already

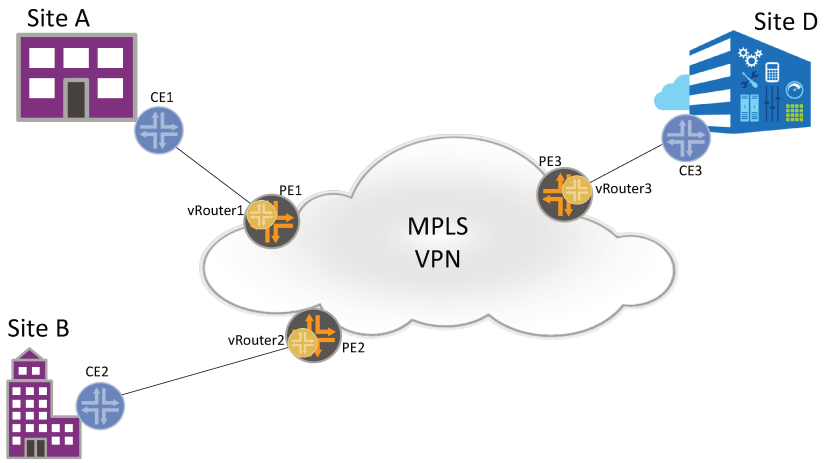

FIGURE 5.4: Physical topology of the negotiated MPLS VPN service

connected to an MPLS network, the new site (D) is requested to be connected to the same MPLS network.

At the customer faced service portal, a service can be represented in a commercial model. For example a VPN service can be represented as a simple router interconnecting different remote sites. In our example as illustrated in Fig. 5.5, the service is presented as a router interconnecting two remote sites A and B. Adding a new interface, called D, will join the new clients of Site D, to this router.

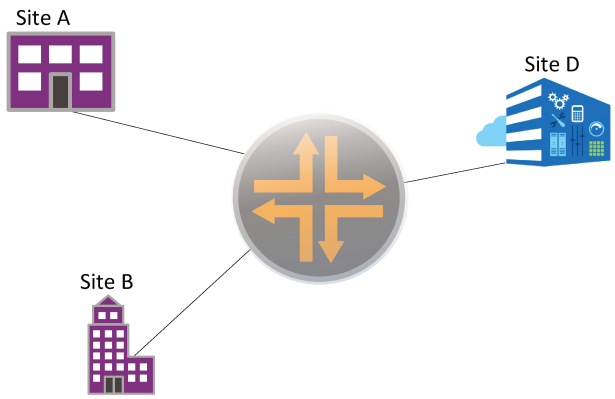

FIGURE 5.5: Abstract architecture of the negotiated MPLS VPN service

Once the service is negotiated and a high-level, commercial service model is generated via the service portal, the SRM sends the model to the SDCM (cf. arrow 2 of Fig. 5.6). By receiving the service model, the SDCM decomposes it to elementary service models by the help of the SMDB. In our example, the update of the negotiated service is "decomposed" to "new MPLS site creation" workflow.

Using its integrated SMDB, the SDCM generates information required to execute this workflow. This information can be the customers specific information, such as number of sites, MPLS Labels, etc. This module disposes also of another integrated database called Topology

Database (TDB), allowing to get resources information needed to fulfill the requested service. Information such as PE IDs, device type (virtual, or physical), etc. are stocked in this database.

Based on these parameters and information, the SDCM "compiles" the requested service to different resource configuration sets sent to the lower modules. In our example, for a VPN service model received from the decomposition phase, the SDCM generates configuration sets needed to create and configure a Virtual Router (vRouter) on the PE3 connecting the Site D (cf. arrows 3 of Fig. 5.6). SDCM generates updates for all vRouters connecting the customers' sites to the VPN, and for the P router connecting directly to PE3 (cf. arrows 7, 9 and 11 of Fig. 5.6).

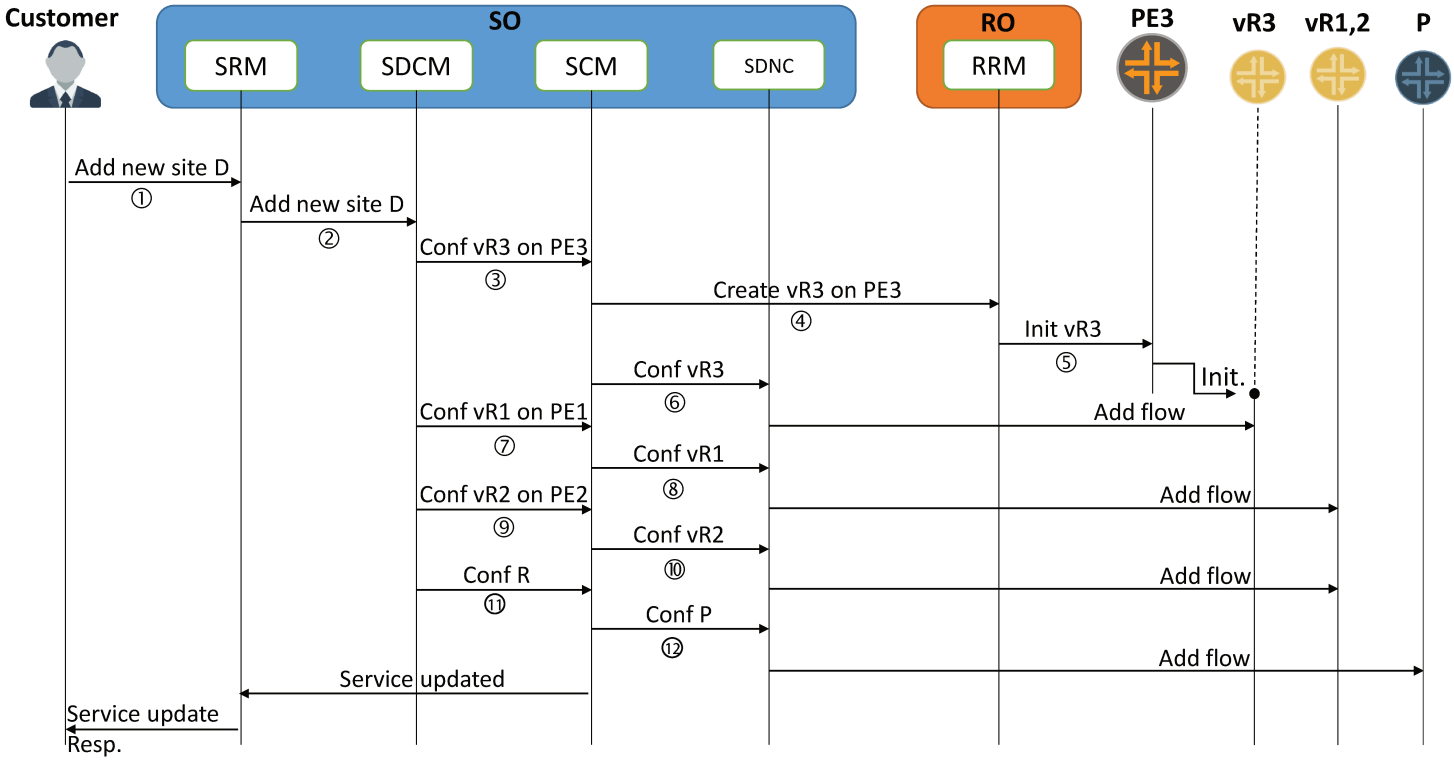

FIGURE 5.6: Service Update Call Flow

5.2.1.3 Service Configuration Manager (SCM)

Generated configurations are sent from the SDCM to the lower module, called SCM (cf. arrows 3, 7, 9 and 11 of Fig. 5.6). This module is used to configure physical or virtual resources. Like all previous modules, the SCM may use a database to find the appropriate device used to fulfill the service. Firstly, SCM uses this database to discover the management IP address of the resource. Secondly, it configures the resource by pushing the configuration instructions, generated by upper layers, through the SDNC (cf. arrows 6, 8, 10 and 12 of Fig. 5.6).

The SCM can be considered as a resource driver of the SO. This module is the interface between the orchestrator and resources. Creating such a module facilitates the processes run at upper layers of the orchestrator where the service can be managed independently of existing technologies, controllers and protocols implementing and controlling resources. On the one hand, this module communicates to different resources through its SBI. On the other hand, it exposes a universal resource model to other SO modules, specifically to SDCM.

Configuring a service by SCM requires a decomposition into two tasks: creating the resource on the first step (if the resource doesn't exist, cf. arrow 4 of Fig. 5.6), and configuring that resource at the second step (cf. arrow 5 of Fig. 5.6). In our example, once the PE3 ID and the required configuration is received from the SDCM side, the SCM, firstly fetches the management IP address of the PE3 from its database. Secondly if the requested vRouter is missing on the PE, it creates a vRouter (cf. arrow 4 of Fig. 5.6). And thirdly, it configures that vRouter to fulfill the requested service.

In order to create the required resource (i.e. to create the vRouter on the PE3), SCM sends a resource creation request to the RO (arrow 4 of Fig. 5.6). Once the virtual resource (vRouter) is initiated, the RO acknowledges the creation of the resource by sending the management IP address of that resource to SCM. All what this latter needs to do, is to push the generated configuration to that vRouter using its management IP address (arrow 6 of Fig. 5.6). The configuration of the vRouter can be done via different methods. In our example the vRouter is an OpenFlow-enabled device programmable via the NBI of an SDNC. To configure the vRouter, SCM uses its interface with the SDNC controlling this resource.

5.2.1.4 SCM - SDN Controller (SDNC) Interface

As we explained, the configuration of part or all of virtual resources used to fulfill a service can be done through an SDNC. In Section 3.2.2.1 we analyzed the architecture of the OpenDaylight controller providing a rich set of modules allowing to program network elements. This controller exposes on its NBI some Representational State Transfer (REST) APIs allowing to program flows thanks to its internal Flow Programmer module. These APIs allow to program the behavior of a switch based on a "match" and "action" tuple. Among all actions done on a received packet, the Flow Programmer allows to push, pop and swap MPLS labels.

In order to program the behavior of the initiated vRouter, we propose to use the API provided by OpenDaylight Flow Programmer. The vRouters role is to push MPLS label into packets going out from Site D to other sites (A and C), and to pop the MPLS labels from incoming packets sent from these remote sites. To program each flow OpenDaylight requires the Datapath ID (DPID) of the vRouter, inbound and outbound port numbers, MPLS labels to be pushed, popped and swapped, and the IP address of next hops where the packet should be sent to. In the following we will discuss how these information is managed to be sent to the SDNC.

DPID: During the resource creation time, the vRouter is programmed to be connected automatically to OpenDaylight. The connection establishment between these two entities is explained in OpenFlow specification, where the vRouter sends its DPID to the controller via OpenFlow features reply. This DPID, known by SCM and SDNC, is further used as the unique ID of this virtual resource.

Port numbers: Inbound and outbound port numbers are practically interface numbers of the vRouter created by the SO. To create a virtual resource, the SCM relies on a resource template explaining the interface ordering of that resource. This template describes which interface is used for management purpose, which interface is connected to the CE and which one is connected to the P router inside the MPLS network. This template is registered inside the database of the SCM, and this module uses this template to generate REST requests sent to the SDNC.

MPLS labels: MPLS labels are other parameters needed to program the flow inside the vRouter. These labels are generated and managed by SDCM. This module controls the consistency of labels inside a managed MPLS network. Labels are generated in this layer and are sent to the SCM to use in service deployment step.

Next hop IP address: When a packet enters to vRouter from the CE side, the MPLS label will be pushed into the packet and it will be sent to the next LSR. Knowing that the MPLS network, including LSRs, is managed and configured by the SO, this one has an updated vision of the topology of this network. The IP address of the P router directly connected to the PE is one of information that can be exported from the topology database of SO managed by SCM.

Once the vRouter is created and configured on the PE3, the LSP of the MPLS network also should be updated. At the end of the vRouter creation step, the customer owns three sites, each one connected to a PE hosting a vRouter. The SCM configures on each vRouter (1 and 2) the label that should be pushed to each packet sent to Site D and vice versa (cf. arrows 6, 8, 10 of Fig. 5.6). It configures also the P router connected directly to PE3 to take into account the label used by the vRouter3 (cf. arrow 12 of Fig. 5.6).

5.2.1.5 Service Monitoring Manager (SMM)

In parallel to the three main modules explained previously, the SO contains a monitoring system, called SMM, that monitors vertically the functionality of all orchestrators modules from the SRM to the SCM and its SDNC. This module has two interfaces to external part of the orchestrator. On the one hand, it receives upcoming alarms and statistics from the lower orchestrator, RO, and on the other hand it communicates the service statistics to the external application via the NBI.

5.2.2 Internal architecture of the Resource Orchestrator

As it is mentioned in previous sections, 4.1.2 and 5.2.1.3, during the service configuration phase, the RO will be called to initiate resources required to implement that service. In the service configuration step, if a resource is missing, the SCM will request the RO to initiate the resource on the specified location. The initiated resource can be virtual or physical according to the operator politic and/or negotiated service contract.

Existing cloud orchestration systems, such as OpenStack platform [81], are good candidates to implement a RO. OpenStack is a modular cloud orchestrator that permits providing and managing a large range of virtual resources, from computing resource, using its Nova module, to L2/L3 LAN connection between the resources, using its Neutron module. The flexibility of this platform, the variety of supported hypervisors and its optimized resource management [82] can help us to automatically provision virtual resources, including virtual servers or virtual network units. We continue exploiting the proposed framework based on a RO implemented by the help of OpenStack.

In order to implement and manage required resources needed to bring up a network service, an interface will be created between the SO and the RO where the SCM can communicate to the underlying OpenStack platform providing the virtual resource pool. This interface provides a resource management abstraction to SO. The resource request will be passed through various internal blocks of the OpenStack, such as Keystone that controls the access to the platform. As our study is mostly focused on service management, in this proposal we don't describe the functionality of each OpenStack module in details. In general, the internal architecture of the required RO is composed of two main modules, one used to provide virtual resources, a composition of Nova, Cinder, Glance and Swift modules of OpenStack, and another one used to monitor these virtual resources, thanks to the Ceilometer module of OpenStack.

If the RO faces an issue it will inform the SO which is consuming the resource. The service run-time lifecycle and performance is monitored by the SO. When it faces an upcoming alarm sent by the RO or a service run-time problem occurring on virtual resources, it will either perform some task to resolve the problem autonomously or send an alarm to the service consumer application (service portal).

Creating a virtual resource requires a set of information, software and hardware specifications, such as the version of the firmware installed inside that resource, number of physical interfaces, the capacity of its Random-Access Memory (RAM), and its startup configurations like the IP address of the resource. For example to deploy a vRouter, the SO needs a software image which installs the firmware of this vRouter. Like all computing resources, a virtual one also requires some amount of RAM and Hard Disk space to use. In OpenStack world, these requirements are gathered within a Flavor.

Fig. 5.7 illustrates a REST call, sent from the SCM to the RO, requesting the creation of the vRouter. In this example, the SCM requests the creation of the "vPE1", that is a vRouter, on

a resource called “PE1” using an image called “cisco-vrouter” and the flavor “1”.

```
curl -X POST -H "X-Auth-Token:\$1" -H "Content-Type: application/json" -d '
{ "server": {
        "name": "vPE1",
        "imageRef": "cisco_vrouter",
        "flavorRef": "1",
        "availability-zone" : "SP::PE1",
        "key_name" : "OrchKeyPair" }
}
' http://resourceorchestrator:8774/v2/admin/servers | python -m json.tool
```

FIGURE 5.7: REST call allowing to reserve a resource

5.2.3 Framework interfaces

The composition of this framework requires the creation of three interfaces (cf. Fig. 5.3). The first one, the NBI, provides an abstracted service model enriched by some value-added services to the third party application or service portal. The second one, the SBI, interconnects the SO to the resource layer through the SDNC. This interface permits the SCM to configure and control virtual or physical resources. Inter-orchestrator (middle) interfaces, is the third interface that is presented for the first time in this framework. This interface interconnects the SO to the ROs. The modular aspect created by this interface permits to implement a distributed orchestration architecture. This architecture allows one or several SO(s) to control and communicate to one or several RO(s).

5.3 Implementation

In order to describe the internal architecture of the framework, we implement different layers of the Service Orchestrator through the MPLS VPN deployment example.

5.3.1 Hardware architecture

Fig. 5.8 shows the physical architecture of our implementation. This one is composed mainly by three servers each one implementing one of the main blocks:

— Server1 implements the Mininet Platform [83]. For the sake of simplicity and because of lack of resources, we implement the infrastructure of our implementation based on a Mininet platform. This one implements all resources, routers and hosts, needed to deploy our desired architecture.

— Server2 implements OpenDaylight SDN controller [32]. For this implementation we use the Carbon version of the OpenDaylight. From its SBI this controller manages resources implemented by the Mininet platform based on OpenFlow protocol.

— Server3 implements the Service Orchestrator (SO). This server is a simple Linux based machine implementing the SO and the customer faced interface. In order to control the underlying network, the SO uses the RESTful NBI of the OpenDaylight controller.

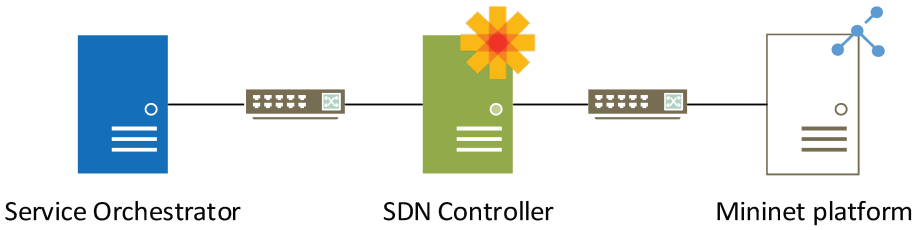

FIGURE 5.8: Physical architecture of the implemented framework

5.3.2 Network architecture

Fig. 5.9 shows the underlying network architecture implemented within the Mininet platform. Switches deployed in this architecture are OpenFlow enabled one, controlled each one by the SDNC.

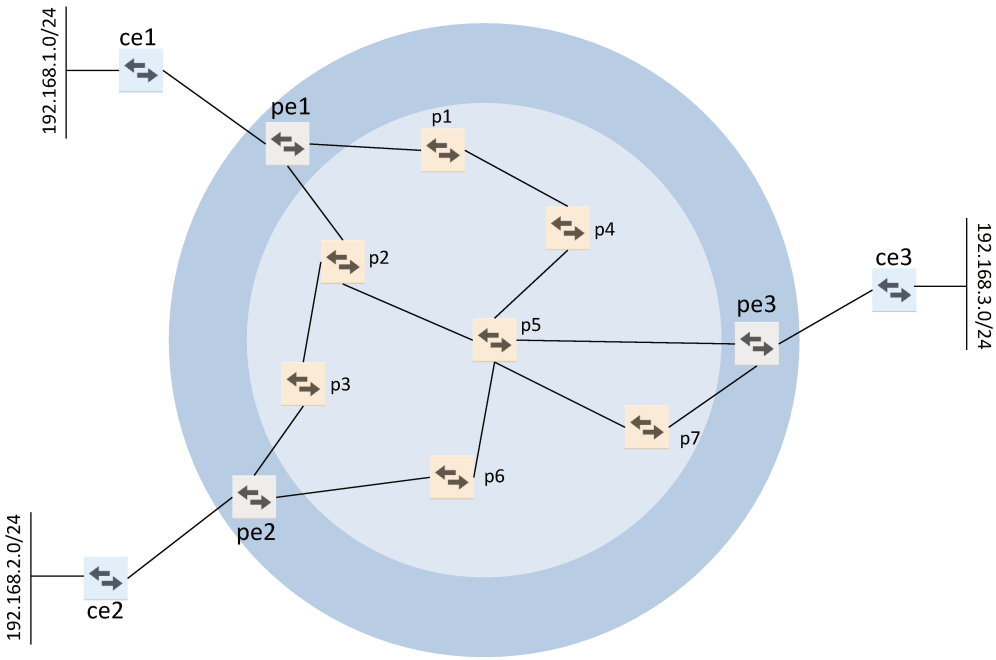

FIGURE 5.9: Network architecture of the implemented service

In the implemented architecture ce1, ce2 and ce3 represent CE devices connecting customer sites to the core network. pe1, pe2 and pe3 switches represent PE devices and p1..7 switches are equivalent to P routers deploying the LSP.

For this implementation we study the case where all three customer sites are already connected to the core network and the physical connection between CE and PE routers is established.

5.3.3 Software architecture

Given that our analysis focuses on the architecture of the SO, in this implementation we study the case where the required resource already exists. In this case the deployment of the service relies on the SO and its related SDNC. Fig. 5.10 shows the internal architecture and the class diagram of the implemented SO.

The architecture of the orchestrator is based on the object oriented paradigm developed in Python 2.7. In our implementation each SOs layer is developed in a separated package:

Service Request Manager (SRM): contains several classes including Service_request, Customer and Service_model. On the one hand it implements a REST API used by the customer, on the other it manages all available services proposed to the customer and the service requested arrived from the customer. For this, it uses two other objects (classes) each one controlling the resources managed in this layer. The first one, Customer class, manages the customer, its subscribed services and available services to him. The second one, the Service_model, manages customer face service models. This model is used to make a representation of the service to the customer.

Fig. 5.11 shows the negotiated MPLS VPN service model requested by the customer. In this model the customer requests creating a VPN connection between three remote sites each one connected to a CE (ce1, ce2, and ce3).

Service Decomposition and Compilation (SDCM): contains several classes and a submodule called Service_transformers. The SDCM sends the requested VPN service model to the specified transformer implemented within the Service transformers package. In order to transform the VPN service model to its equivalent device models, we implemented a Mpls_vpn_transformer class converting the higher level model to a lower level one.

Fig. 5.12 shows the simplified algorithm implemented within the MPLS_vpn_transformer. In the first step, this module retrieves related PE list connected to each remote site from the Topology module. Using the integrated Dijkstra engine of the Topology module, it calculates the shortest path to reach other sites from each PE. And using the labels generated by the Label_manager, and device model templates managed by the Flow_manager, it generates a list of device models to be deployed on the underlying network devices to create the required LSP.

In our implementation we use a device model database containing all models needed to create a MPLS network on an OpenFlow based infrastructure. This database is managed by the Flow_manager module. The entries of this database are each one a flow template

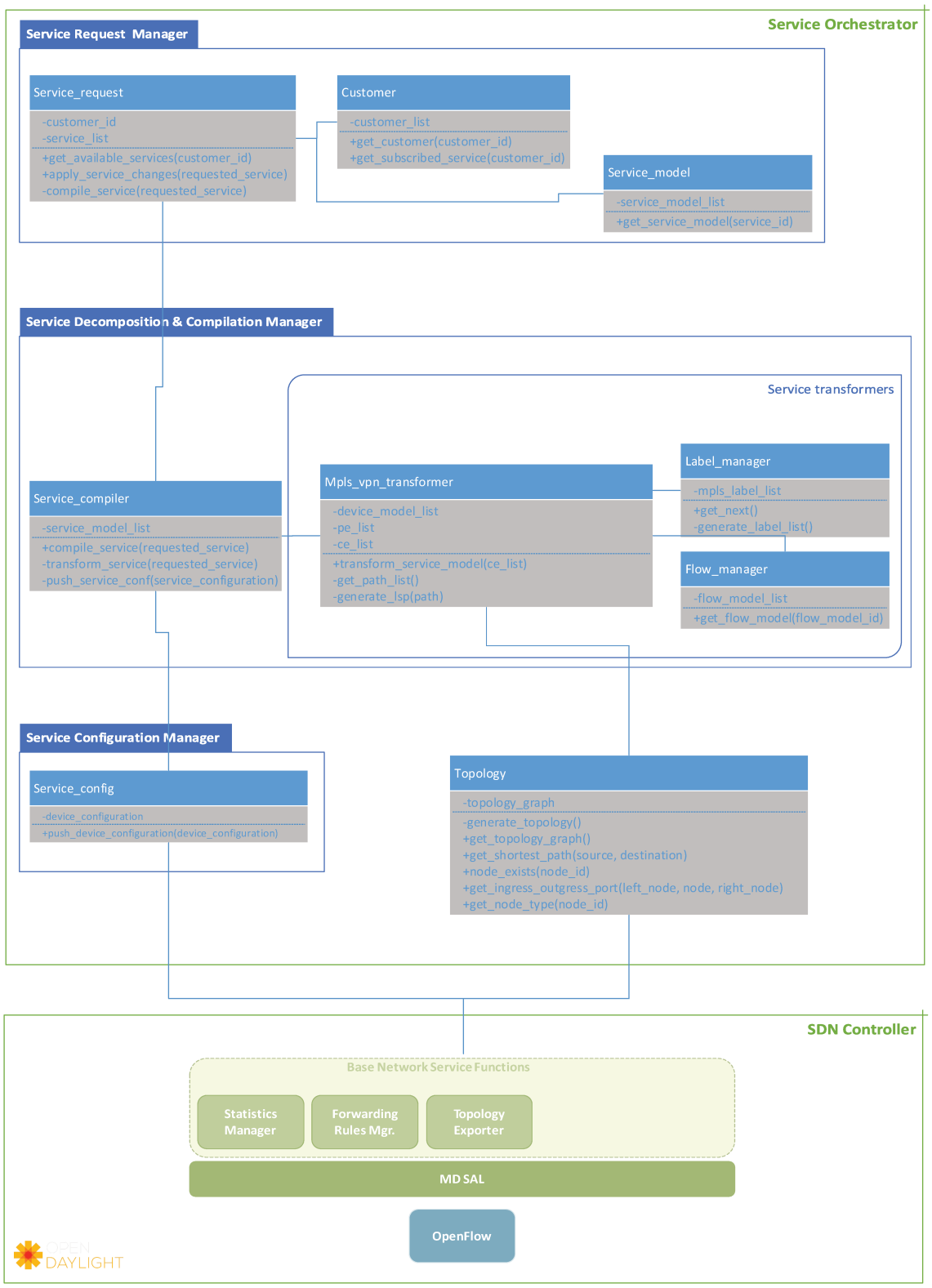

FIGURE 5.10: Class diagram of the implemented framework

containing empty match conditions, and forwarding actions fields. These empty fields are replaced by values calculated by the Mpls_vpn_transformer, to generate a device-specific flow. Fig. 5.13 shows a part of a flow template (i.e. device model template) allowing to push

```
{
"service_type": "mpls_vpn",
"customer_id": "customer_1",
"properties": {
  "ce_list": [{
    "ce_id": "ce1",
      "lan_net": "192.168.1.0/24"
    },
    {
      "ce_id": "ce2",
      "lan_net": "192.168.2.0/24"
    },
    {
      "ce_id": "ce2",
      "lan_net": "192.168.3.0/24"
    }]
  }
}
```

FIGURE 5.11: Negotiated MPLS VPN service model

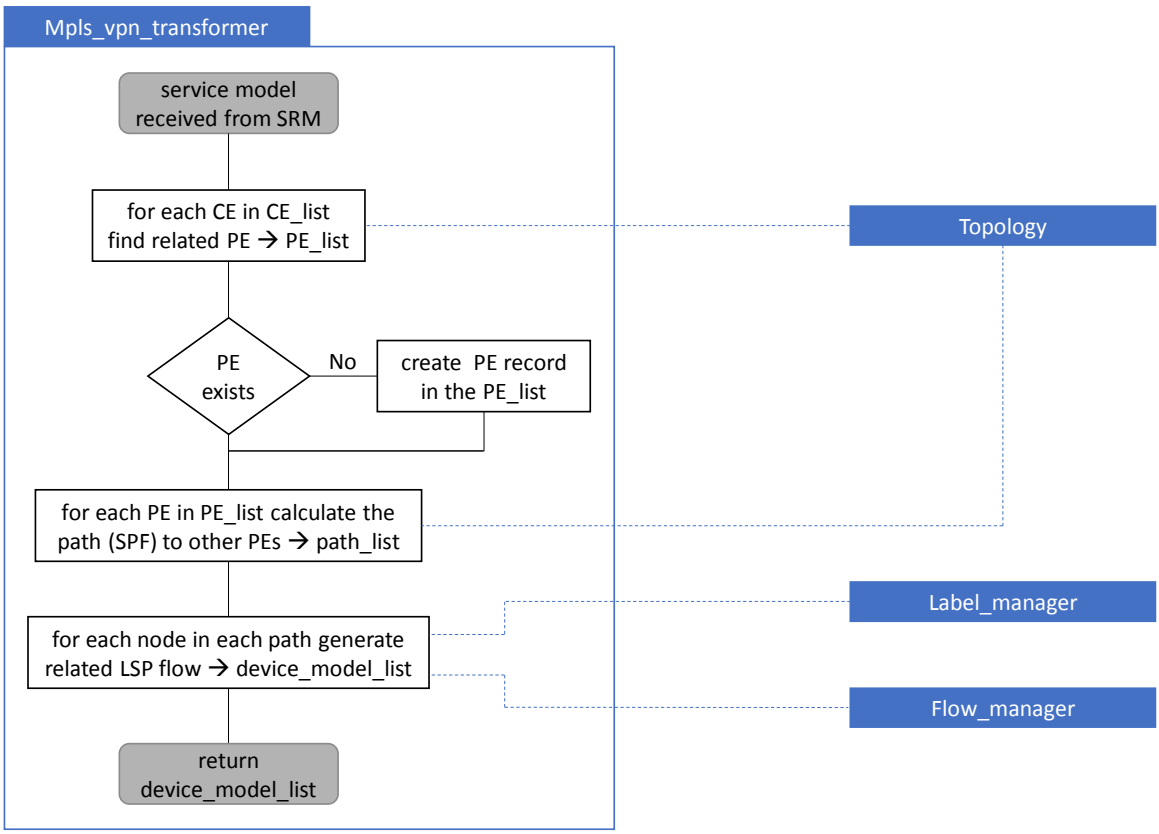

FIGURE 5.12: Implemented MPLS VPN transformer simplified algorithm

a MPLS label to a packet. This template can be used to program an ingress LSR.

```
{
"model_id": "mpls_push",
"flow": {
  ...
  "instructions": {
    "instruction": {
       ...
       "apply-actions": {
        "action": [{
          "push-mpls-action": {
            "ethernet-type": "34887"
          },
          ...
        },
        {
          "set-field": {
            "protocol-match-fields": {
              "mpls-label": ""
            }
          },
          ...
        },
        {
          "output-action": {
            "output-node-connector": ""
          },
          ...
        }]
      }
    }
  },
  ...
  "match": {
    "ethernet-match": {
      "ethernet-type": {
        "type": "2048"
      }
    },
    "in-port": "",
    "ipv4-destination": ""
  },
  ...
}
}
```

FIGURE 5.13: A simplified device model template used for Ingress LSR

Once all device models are generated by the Mpls_vpn_transformer, the Service_compiler

sends them to the lower layer, the **Service Configuration Manager (SCM)**. In this implementation the SCM pushes generated flows to the OpenDaylight through the REST interface of the controller. It's worth noting that to create these rules within underlying devices, the Forwarding Rules Manager module of OpenDaylight is used.

The **Topology** module is another key element implemented in this orchestrator. This module retrieves the topology information of the infrastructure via the REST interface of the Topology Explorer of the controller. It generates a topology graph usable by other blocks of the orchestrator.

5.4 Conclusion

In this chapter, we proposed a SDN framework derived from the operator-side service lifecycle discussed in 4.1.2. This framework which is structured in a modular way encapsulates SDNC with two orchestrators, SO and RO, dedicated respectively to the management of services and resources. The proposed framework is externally limited by NBI and SBI and internally clarifies the border between the two orchestrators by identifying an internal interface between them, called the middle interface, which provides a virtual resource abstraction layer on the top the RO. Our approach gives the foundation for the rigorous definition of the SDN architecture.

It is important to note the difference between the SO and its complementary system, RO. The RO provisions, maintains and monitors physical devices hosting several virtual resources. It doesn't dispose any perspective of running configuration on each virtual resource. Unlike RO, the SO manages the internal behavior of each resource. It is also the responsible of interconnecting several virtual resources to conduct a required service.

Finally, we described in 5.3 the implementation of the main components of the proposed framework based on OpenDaylight controller and Mininet platform. In this prototype we study the service data model transformation, discussed in 4.2, through a simple MPLS VPN service deployment.

Chapter 6

Bring Your Own Control (BYOC)

Contents

3.1 Introduction to MPLS networks 23
3.1.1 MPLS data plan 23
3.1.2 MPLS control plan 24
3.1.3 MPLS VPN Sample Configuration 26
3.1.4 MPLS VPN Service Management 27
3.2 SDN-based MPLS 28
3.2.1 OpenContrail Solution 28
3.2.2 OpenFlow-based MPLS Networks 29
3.3 Outsourcing problematics 34

NBI refers to the software interfaces between the controller and the applications running atop. These ones are abstracted through the application layer consisting in a set of applications and management systems acting upon the network behavior at the top of the SDN stack through the NBI. The centralized nature of this architecture brings large benefits to the network management domain, including the third party network programmability access. Network applications, on the highest layer of the architecture, achieve the desired network behavior without knowledge of detailed physical network configuration. The implementation of the NBI relies on the level of the network abstraction to be provided to the application and the type of the control that the application brings to the controller, called SO in our work.

NBI appears as a natural administrative border between the SDN Orchestrator, managed by an operator, and its potential clients residing in the application layer. We argue that providing to the operator the capability of mastering SDN's openness on its northbound side should largely be profitable to both operator and clients. We introduce such a capability through the concept of *BYOC: Bring Your Own Control* which consists in delegating all or a part of the network control or management role to a third party application called Guest Controller (GC) and owned by an external client. An overall structure of this concept is presented in Fig.6.1, which shows the logical position of the Bring Your Own Control (BYOC) application in the traditional SDN architecture, that includes partly the Control Layer and the Application one.

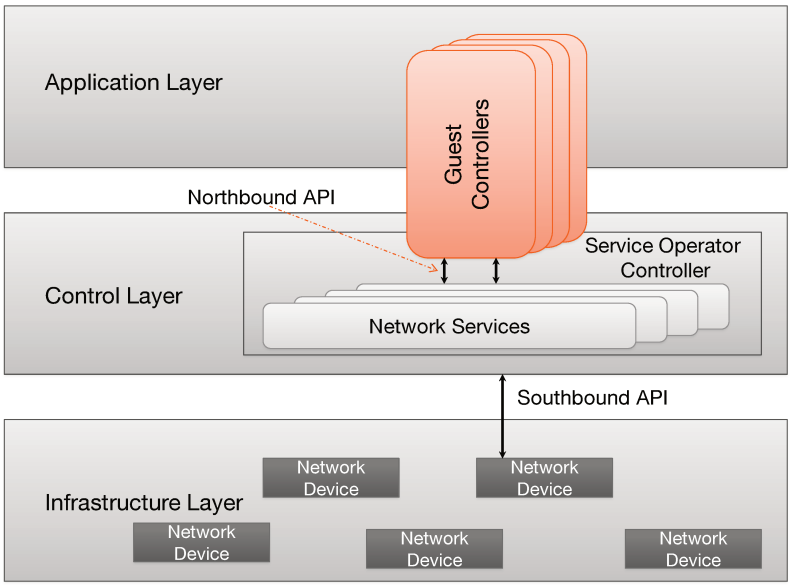

FIGURE 6.1: Positioning the BYOC in the SDN architecture

BYOC should clearly allow to reduce the controller processing load. Indeed, existing SDN architectures and proposals mostly centralize all network control and decision logic in one single entity. This one has to support a huge load when providing a large number of services all deployed in the same entity. Such a complexity is clearly an issue which BYOC may help to solve by outsourcing a part of the control to a third-party application. Preserving service customer application's confidentiality is another nice point brought by BYOC. In fact, centralizing the network control in one system and passing all data from that controller may create some confidential problems which may prevent the end user, we call Service Chain (SC), to use the SDNC. Last but not least, BYOC may help the operator to noticeably refine his SDN-based business model by delegating nearly "à la carte" control through dedicated APIs. Such an approach may smartly be exploited accordingly to the new paradigm of *"Earn as You Bring" (EaYB)* we introduce and describe below. Indeed, an external client owning a proprietary sophisticated algorithm may want to sell the associated specialized treatments to other clients via the SDN operator who might appear as a broker of this kind of capabilities becoming by the way easily available to interested SC. It should be pointed out that such BYOC benefits may partly be compensated by the non trivial task of verifying the soundness of the decisions taken by the outsourced intelligence which have to be at least conform to the various policies implemented by the operator in the controller. This point which deserves more investigation, may be the subject of future research work.

6.1 Analysis of BYOC concept

6.1.1 Outsourcing services

Services which can be outsourced to the Guest Controller (GC) to influence the network behavior through the NBI, can be classified in three types. These services configured, and for some of them managed, through SDN applications are implemented at the north side of a SDNC. Recently we presented a classification of SDN application in Section 2.5.2. According to this analysis, SDN applications are classified in three types. We presented in the Fig. 2.8 this classification. For the sake of simplicity the GCs are fully integrated to the application layer and each GC is considered as a SDN application.

6.1.2 SDN Service Lifecycle and NBI APIs

SDN services induce in the controller complex tasks. We suggest to exploit their lifecycle in order to precisely define which part of their management or control may be outsourced to the Guest Controller. We use a service lifecycle model defined in 4.1 which is structured in two articulated points of view: service client view and service operator view, each one captures only the information useful for the associated actor. The former, client-side service lifecycle, represents series of tasks executed by the service client in order to create, monitor, update and finaly retire a service. The latter, operator-side service lifecycle, concerns the tasks that are executed in the operator side during the life of a service, including: service request management, service decomposition/compilation, service configuration, service maintain/monitoring and operation and service update. It is shown in the cited section that the NBI can be structured around two main APIs as illustrated in Fig. 6.2. The first API concentrates on service creation, modification and retirement, and the second one focuses on monitoring tasks.

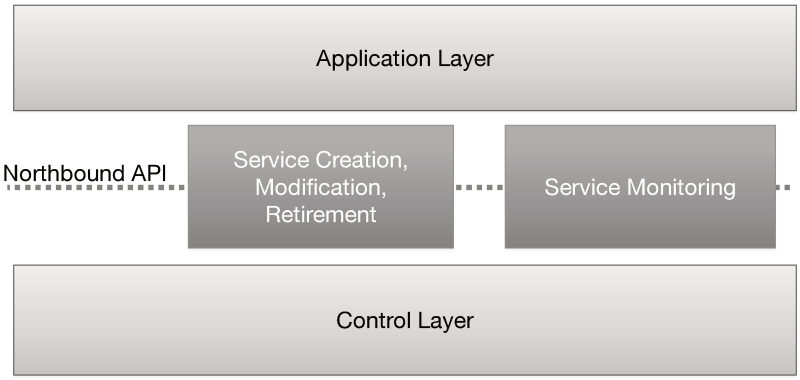

FIGURE 6.2: Main APIs structuring the NBI

Outsourcing a part of management and control tasks modifies the global service lifecycle model. It consists indeed in translating to the client side, parts of some tasks initially belonging to the operator one. A careful analysis allows us to identify the compilation and the

monitoring tasks, done at the operator-side service lifecycle, as potentially interesting candidates, some parts of which may be delegated to the GC. Such an outsourcing moreover leads to enrich in some ways the APIs described in Fig. 6.2.

6.1.2.1 Applying the BYOC concept to Type 1 services

Configuring a service in the SO is initiated after the Service compilation phase of the operator side service lifecycle 4.1.2. This one translates the abstracted network models into detailed network configurations thanks to integrated network topology and statement databases. In order to apply the BYOC concept, all or a part of the service compilation phase may be outsourced to the application side represented by the GC. For example, the resource configuration set of a requested VPN service, discussed in Section 5.1, can be generated by a GC. This delegation needs an interface between the SDCM and the GC. We suggest to enrich the first API with dedicated primitives allowing the GC to proceed to the delegated part of the complete compilation process (cf. the primitive "*Outsourced (Service Compilation)*" in Fig. 6.3).

It is worth pointing out that the compilation process assigned to the GC could be partial because the operator may want to maintain the confidentiality of sensitive information, as for example the topology of its infrastructure.

6.1.2.2 Applying the BYOC concept to Type 2 services

In this case the application may configure a service and monitor it via the NBI. This type involves the compilation phase, discussed earlier, and the monitoring one. Outsourcing the monitoring task from the Controller to the GC, thanks to the BYOC concept, requires an asynchronous API that permits to transfer the real-time network events to GC during the monitoring phase. The control application implemented in the GC observes the network state thanks to the real-time events sent from the Controller. A recent work [84] proposed an XMPP-based push/pull solution to implement an NBI that permits to communicate the networks real-time events to the application for a monitoring purpose. The outsourced monitoring is located in the second API of Fig. 6.2 and could be expressed by refining some existing primitives of the API (cf. "*Outsourced (Service Monitoring Req./Resp.)*" of Fig. 6.3).

6.1.2.3 Applying the BYOC concept to Type 3 services

This type concerns the application that configures a service (Type 1) and monitors it (Type 2), according to which it may modify the network configuration and re-initiate the partial compilation process (Type 1). The second API of Fig. 6.2 should be sufficient to implement such type of service even if it may necessitate non trivial refinements or extensions in order to be able to collect the information needed by GC. The delegation of the control induced by this kind of GC comes exactly from the computation of the new configuration together with its re-injection in the network, through the SDNC, in order to modify it.

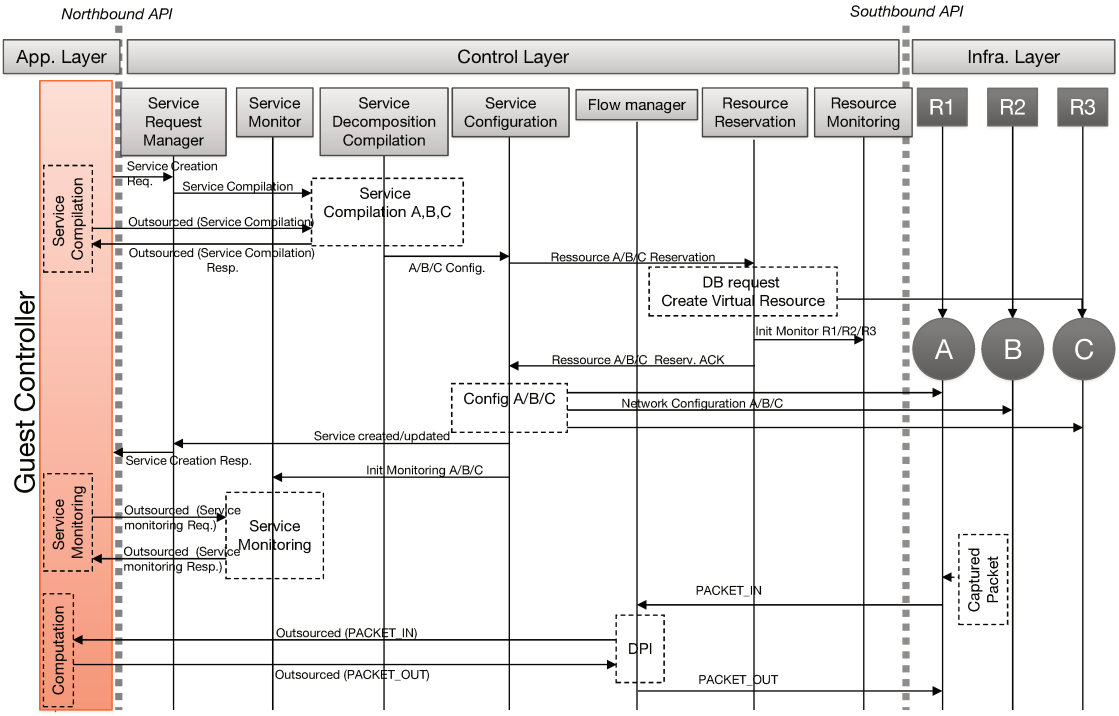

FIGURE 6.3: Nominal call-flow for the three types of BYOC services applied to the "MPLS networks" use case

Type 3 BYOC can be fundamentally characterized by its real-time retroactive capability which, on the one hand, makes it really precious for the SC who may benefit from service-adaptable monitoring. The SC may indeed react very quickly to unexpected harmful behaviors of its network by correctly tuning and adapting its configuration and eventually its monitoring policy. The operator, on the other hand, can fruitfully exploit it in providing a quality of auto-controlled elasticity to its sold network services.

If the proprietary treatments carried by SC through its associated GC have a very high added-value, SC could be interested to sell this capability to other potential clients without however being directly in contact with them. The operator may wisely appear as the right intermediary between SC and these potential clients. This role of broker may by the way, allows to the operator to develop new business models following the original paradigm of Earn as You Bring (EaYB). This one can be conjugated at least as "Earn as You Bring" or "Earn much more than You Bring", etc. A very interesting benefit of EaYB concerns its transparency relatively to the newly introduced BYOC APIs on which it has no impact. Indeed, the open interface between the SO and the GC, allows to delegate natively the control of all or part of a customers service to a GC brought by a third party customer or a virtual operator. This requires however the integration of a Broker unit in the SO layer, the specification of which is not trivial and deserves further work.

6.2 Northbound Interface permitting the deployment of a BYOC service

6.2.1 Requirements for specification of the NBI

The GC is connected to the SO through the NBI. This is where the service operator communicates with the service customer and sometimes couples with the client side applications, orchestrators, and GC(s). In order to accomplish these functionalities certain packages should be implemented. These packages maintain two categories of tasks: 1) Service creation, configuration and modification, and 2) Service monitoring and BYOC service control.

1. **Synchronous vs Asynchronous interactions**
   The former uses a synchronous interaction that implements a simple request/reply communication that permits the client-side application to send service requests and modifications, while the latter uses an asynchronous interaction where a notification will be pushed to the subscribed service application. The asynchronous nature of this package makes it useful for sending control messages to the GC.

   From its SBI, the SO tracks the network events sent by resources. Based on the service profile related to the resources, it sends events to concerning modules implemented either inside the SO or within an external GC.

2. **Push/Pull paradigm to structure the interactions**
   The communication between the GC and the SO is based on Push-and-Pull (PaP) algorithm [85] that is basically used for the HTTP browsing reasons. In this proposal we try to adapt this algorithm to determine the communication method of the NBI which will use publish/submit messaging paradigm. The GC subscribes to the SO. To manage BYOC-type services, Decision Engine (DE) and Service Dispatcher (SD) modules are implemented within the SO. The DE receives messages sent by network elements and based on them it decides whether to treat the message inside the SO or forward the message to the GC. For messages needed to be analyzed within a GC, the DE sends them to the SD. The SD distributes these messages to every GC that has subscribed to the corresponding service.

   Figure 6.4 illustrates the communication between components of the system in detail.

   a: The service customer requests a service from the SRM, through the service portal. In addition to the service request confirmation, the system sends the subscription details about the way that service is managed, internal or BYOC.

   b: Using the subscription details, the user connects to the SD unit and subscribes to the relevant service.

   c: When a control message, e.g. OpenFlow PackeIn message, is sent to the DE, the DE creates a notification and sends it to the SD.

   d: The SD unit pushes the event to all subscribers of that specific service.

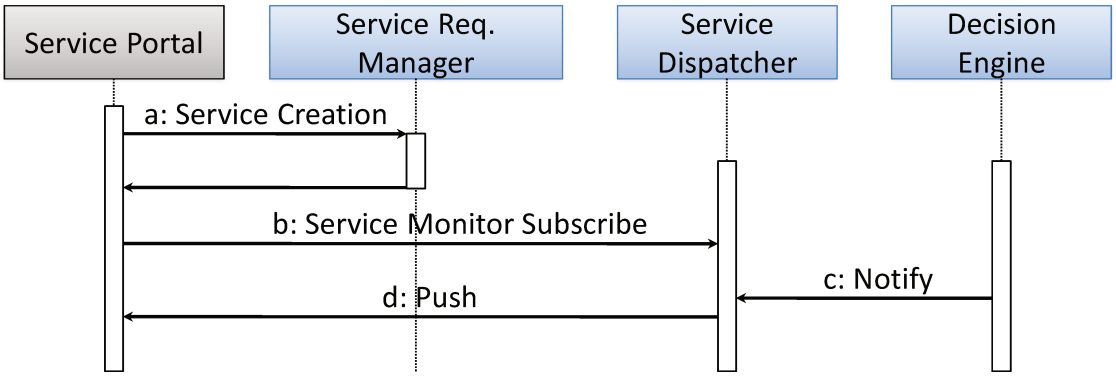

FIGURE 6.4: Communication between components

3. **Managing the secured multi-access**
   The importance of a service provisioning system is relied on its NBI which connects the service portal to the service provisioning system (SO) in an SDN environment. This interface provides a service-layer abstraction view and essential functions to create, modify and destroy a service, and, as described above, it takes into account the externalized control units called GC. This interface is a shared access point between different customers, each one controlling specific services with an associated subscription to certain events and notifications. It's therefore important that this shared interface implements an isolated environment to provide a multi-tenant access. This one should be controlled by the help of a built-in authentication and authorization system.

6.2.2 SDN NBI Implementations

Nowadays numerous controllers are trying to offer innovative services through their open NBIs. They use the HTTP-based technologies to implement the NBIs by the help of the Web Services. This one is a distributed middleware technology that uses simple XML-based protocol to allow applications to exchange data across the Web. REST [86] is the most common technology used to implement Web Services. It formalizes how applications use functions provided by the SDN controller. Functions like path computation, loop avoidance, routing, security may be provided to the application layer through the NBI APIs. These APIs may also allow orchestration systems, such as OpenStack Neutron [81] or VMware vCloud Director [87] to manage network services in a cloud environment and provide Network as a Service (NaaS) and Infrastructure as a Service (IaaS) services through the SDN controller.

6.2.2.1 Representational state transfer (REST)

REST is a resource-based architectural style of the Web, known as the easiest way to implement Web services [88], developed by W3C. It introduces a Web-like designing by using a fixed set of operations consisting of the basic HTTP verbs, including: GET, PUT, POST and DELETE. These verbs are used to access to REST resources, each one represented by

a unique Uniform Resource Locator (URL). Resources are application's state and functionality which are represented by a uniform interface to transfer the state between the client and server. Unlike most of web services architecture, it is not necessary to use XML as a data interchange format in REST. The implementation of the REST is standard-less and the format of exchanged information can be in XML, JavaScript Object Notation (JSON), Comma-Separated Values (CSV), plain text, Rich Site Summary (RSS) or even in HyperText Markup Language (HTML), i.e. REST is ambivalent.

The simplicity, performance and scalability of REST are the reasons of its popularity in SDN controllers' world. REST is easier to use and is more flexible. In order to interact with the Web Services, no expensive tools are required. Comparing to our requirements explained in section 6.2.1, the fundamental limitation of this method rests on the absence of asynchronous capabilities and managing a secured multi-access.

Traditional Web Services solutions, such as Simple Object Access Protocol (SOAP) have previously been used to specify NBI but quickly abandoned in favor of the RESTful approach. The initiative concerning the WebSockets [89] should eventually be interesting, but actually this solution is still under development. As mentioned in the work of Franklin and Zdonik [90], push systems are actually implemented with the help of a periodic pull that may cause an important charge in the network. Alternative solutions like Asynchronous JAvascript and Xml (AJAX) also rely on client's initiated messages. We argue that XMPP [91] could be a good candidate due to its maturity and simplicity it may cope with all the previous requirements.

6.2.3 XMPP As An Alternative Solution

XMPP [91], also known as Jabber, is originally developed as an Instant Messaging (IM) protocol by the Jabber community. This protocol, formalized by the IETF, uses an XML streaming technology in order to exchange XML elements, called stanza, between any two entities across the network, each one identified by a unique Jabber ID (JID). The JID format is composed of three elements: "node@domain/resource" where the "node" can be a username, the "domain" is a server and the "resource" can be a device identifier. XMPP Standard Foundation tries to enlarge the capability of this protocol by providing a collection of XMPP Extension Protocols (XEP)s [92], XEP-0072 [93], for example, defines methods for transporting SOAP messages over XMPP. Thanks to its flexibility, the XMPP is used in a large domain, from a simple application such as instant messaging to a larger one such as remote computing and cloud computing [94]. The work [95] shows how XMPP is a compelling solution for cloud services and how its push mechanism eliminates unnecessary polling. XMPP forms a push mechanism where nodes can receive messages and notifications whenever they occur on the server. This asynchronous nature eliminates the need for periodic pull messages.

6.2.3.1 XMPP-based NBI

In this section we introduce an XMPP-based NBI. The main reason of selecting this protocol to implement the NBI of the service provisioning system relies on its asynchronous interaction model that, by the help of its natural push system, empowers the implementation of a BYOC service.

The global architecture of this proposition is represented in Figure 6.5 where the GC, SRM and SD modules contain an XMPP client each one identified by a unique JID.

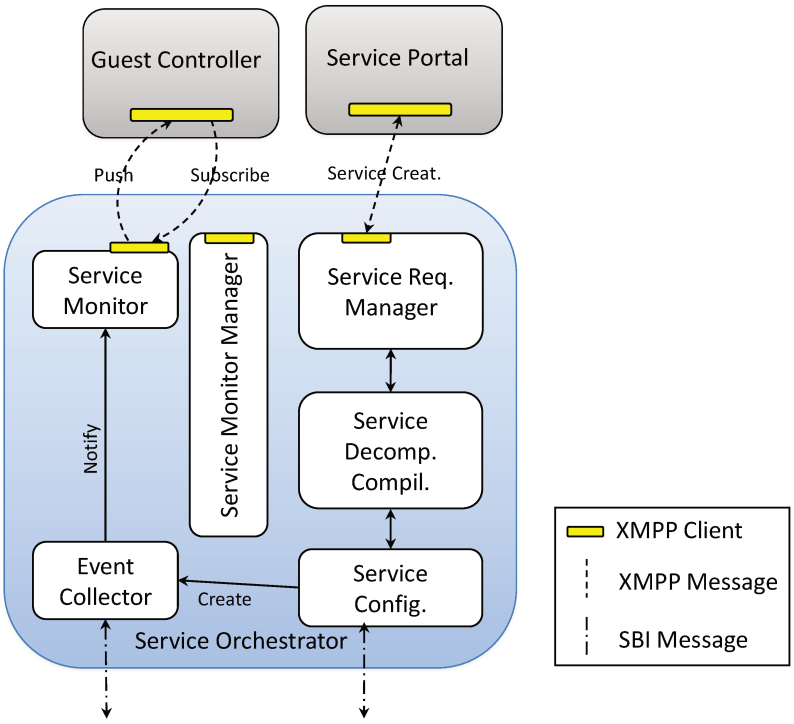

FIGURE 6.5: The XMPP-based NBI

6.2.3.2 Global design

The service customer sends service creation or modification messages by the help of the XMPP protocol to the SRM through the Service Portal. As mentioned recently, he negotiates the SLA and can specify certain service metrics and part of service whose control should be externalized during the service lifecycle. This negotiation is done by the help of the XMPP described in the XML. Once the service is negotiated, it will be deployed in the underlying network. Finally, the client will receive a network service deployment confirmation attaching the JID of the SD block, by the help of which the client subscribes his GC to the service monitoring messages. Once the GC is subscribed to a specific service control message, the SD can push the notifications by the help of the XMPP in an asynchronous and near real-time manner. The Publish-Subscription process can be implemented either by the XMPP core protocol or by the help of existing XMPP extension (XEP-0060) [96] specified for generic publish-subscribe functionality. Choosing XMPP as the NBI permits to implement

two main crucial packages listed in the beginning of this section; packages that execute 1) Service creation, configuration and modification, and 2) Service monitoring and BYOC service control. The NBI security problem also is considered in this proposal. The XMPP specifications describe security functions as the core parts of the protocol [91] and all XMPP libraries support these functionalities by default. XMPP provides a secure communication through an encrypted channel (Transport Layer Security (TLS)) and restricts the client access via the Simple Authentication and Security Layer (SASL) that permits XMPP servers to accept only encrypted connections. All this signifies that XMPP is well suited for constructing a secured NBI allowing to deploy a BYOC service.

6.2.3.3 NBI Data Model

In order to hide the service implementation complexity, services can be represented as a simple resource model described in a data modeling language. YANG data modeling language [74] can be a good candidate for this purpose. A YANG data model is translated into an equivalent XML syntax called YANG Independent Notation (YIN) that, on the one hand, allows the use of a rich set of XML-based tools and, on the other hand, can be easily transported through the XMPP-based NBI.

6.2.4 Simulation results

In order to evaluate this proposal, we assessed the XMPP NBI implementation performance in term of delay and overhead costs by comparing a simple GC using an XMPP-based NBI with the same GC using a RESTful-based one. Once the term "near to real-time" is used to develop a system, the delay is the first parameter to be reduced. In a multi-tenant environment the system charge is also the other important parameter to take into account. To measure these parameters and compare them in the XMPP case versus the REST one, we need to implement a simple GC that exploits the NBI to monitor packets belonging to some specific service, in our case HTTP filtering service. The underlying network is simulated thanks to Mininet [83] project. We implemented two NBIs, XMPP-based and RESTful) which accessed to the NBI, in parallel. We use the term "event" to describe control messages sent from the SO to the GC. In the XMPP-based NBI case this event is pushed near-to real-time, thanks to the XMPP protocol. But for the RESTful one, the event message should be stored in a temporary memory before the GC pull them up REST requests. In this case, to simulate a real-time process and to reduce the delay, REST requests are sent in a little time intervals.

In the case of the XMPP-based NBI, the event is sent in a delay of 0.28 ms. The NBI overhead of this NBI is 530 Bytes which is the size of an XMPP message needed to carry the event. In the other case, with the RESTful NBI, the GC will pull periodic message to push this information, a request/response message that will be at least 293 Bytes. In order to reduce the delay, the time interval between each request should be scaled down. This periodic

request/response messages will create a huge overhead in the NBI. The Fig. 6.6 shows the Overhead charge of the NBI obtained during this test. The NBI Overhead charge rests constant for the XMPP-based case and varies for the RESTful one. The overhead charge of the NBI in the simulated real-time case (less that 1 ms of delay) for the RESTful NBI is about 3 MB. To reduce this charge and achieve the same Overhead as XMPP-based one, we need to increase the time interval up to 200 ms. This time interval will have a direct effect on the event transfer delay.

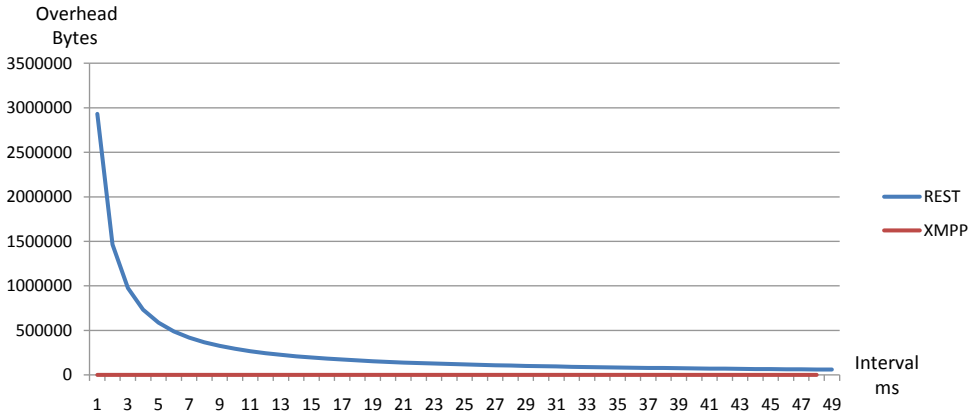

FIGURE 6.6: NBI Overhead Charge

6.3 Conclusion

In the first part of this chapter we introduced BYOC as a new concept providing a convenient framework structuring the openness of the SDN on its northbound side. We derived from the lifecycle characterizing the services deployed in an SDN, the parts of services the control of which may be delegated by the operator to external GC through dedicated APIs located in the NBI.

We presented EaYB business model through which the operator monetizes the openness of its SDN platform thanks to the BYOC concept. Several use cases are briefly presented, that have potential interest to be implemented by the BYOC concept.

In the second part we determined basic requirements to specify an NBI that tightly couples the SDN framework, presented recently, with the GC. We proposed an XMPP-based NBI conforming to previously discussed requirements and allowing to deploy the BYOC service.

Apart all the numerous advantages of the XMPP-based NBI, the main limitation concern the transfer of large service descriptions. These ones are restricted by the "maximum stanza site" value that limits the maximum size of the XMPP message processed and accepted by the server. This value can however be parameterized when deploying the XMPP server.

Chapter 7

BYOC Use Case

Contents

4.1 Service Lifecycle **38**
4.1.1 Client side Service Lifecycle 38
4.1.2 Operator Side Service Lifecycle 41
4.1.3 The global view 43
4.2 Service Data Model **44**
4.2.1 A two-layered approach 44
4.2.2 Applying the two-layered model approach on Service Lifecycle . . . 45
4.3 Conclusion **48**

In the last chapter we proposed a new service implementing BYOC-based services. Lastly in chapter 4 we analyzed the lifecycle of a service based on two service actors, the client and the operator, view points. Dividing the service lifecycle into two parts refines our analysis and helped us to present lastly a SDN framework in chapter 5, where we discussed a framework through which a negotiated service model can be vertically implemented. This framework is issued from the operator-side service lifecycle steps, where that permits not only to implement and control a service, but also to manage its lifecycle. The second part of service lifecycle, client-side, presents all steps that every applications types, presented in section 2.5.2, may take to deploy, monitor and reconfigure a service through the SDNC. In this chapter we rapidly showed how the lastly presented framework permits to deploy a BYOC-type service. We also presented an XMPP-based NBI allowing to open the interface to the GC.

7.1 IPS Control Plane as a Service

7.1.1 Referenced architecture

The architecture of proposed service is based on the Intrusion Prevention System (IPS) architecture divided into two entities. The first one is Intrusion Detection System (IDS)-end that is implemented in key points of the network and observes real-time traffics. The second one, called Security Manager (SM), is a central management system that, thanks to its

centralized view of the network, controls one or several IDS-end entities. This architecture, illustrated in Fig. 7.1, is proposed for the first time in the NM-DIPS work [97].

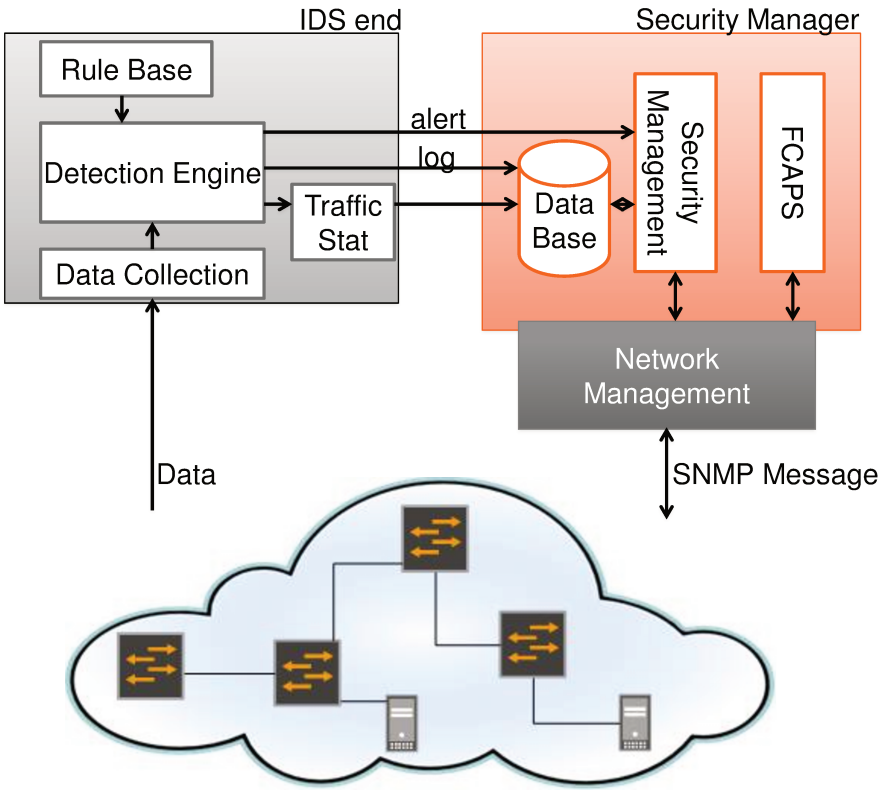

FIGURE 7.1: NM-DIPS architecture (source [97])

In order to ensure the network security several IDS-ends can be installed in the key points of the network. One of these points can be the gateway connecting the LAN to the Internet. The IDS-end analyzes the traffic passing through routers and firewalls, thanks to the SPAN mechanism. Once an anomaly is detected; it sends firstly the suspicious traffic packets to a database located within the SM, and then, it sends an alert message to the SM. By receiving the alert message, the SM analyzes these unusual packets sent into the database and it makes a decision, for example: blocking a special IP address. To deploy the decision into the underlying network, the SM sends an update message through the SNMP protocol.

In the case of an SDN based architecture, the SM can benefit the centralized control capacity of a SDN controller to apply its decision in the network. With the same vision, recently the ALADDIN project [98] proposes a security administration module that protects the network against DDoS attacks. The architecture of this proposal is presented in Fig. 7.2. This architecture is composed of three principal units:

— Vespa agents implemented within core network devices
— Network controller, equivalent to the SDN controller
— Network administration block (Vespa)

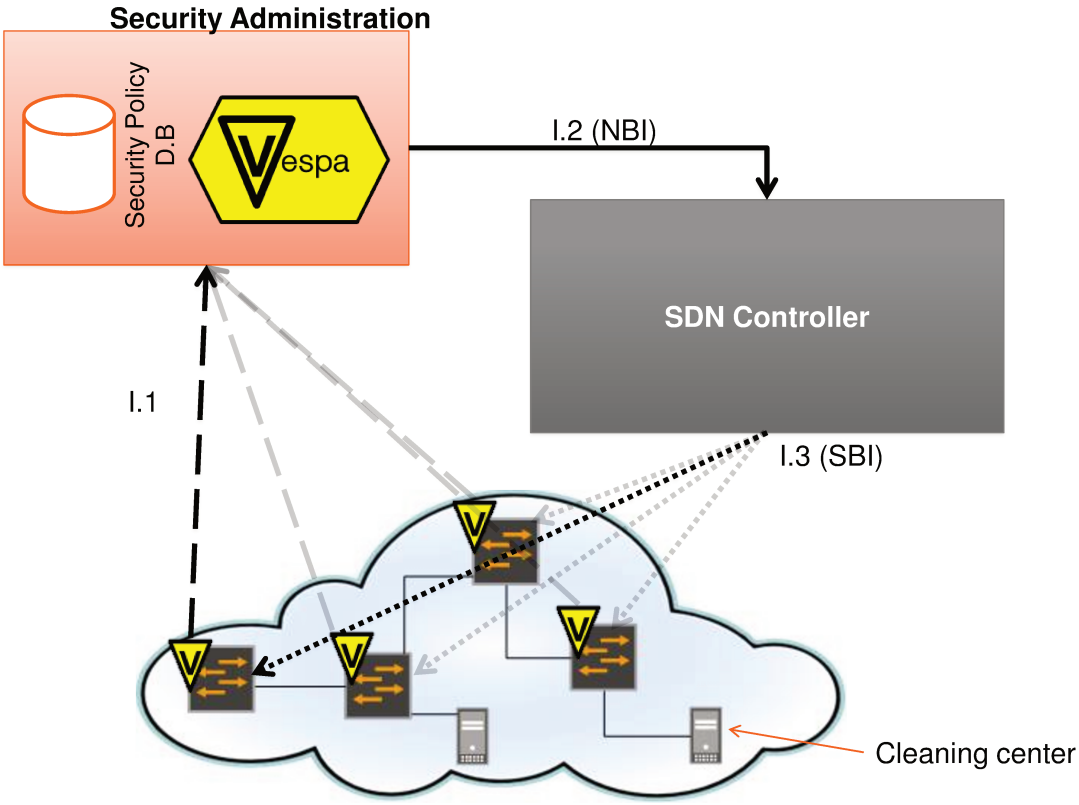

FIGURE 7.2: ALADDIN architecture (source [98])

This solution proposes also three interfaces: the first interface between Vespa agents and the security administration module (I.1), the second interface between the security administration module and the SDN controller (I.2), and the last one between the SDN controller and the network devices (I.3). The security administration block implements an attack detection engine using the Vespa application. This engine has a direct interface with Vespa agents implemented within the network devices. Thanks to these agents, the Vespa application analyzes traffics passing the network. Once an anomaly is detected it configures network devices, thanks to the SDN controller, to redirect the suspicious traffic to "a cleaning center". Creating the direct interface between Vespa agents and the security management block, makes the underlying network layer dependent to the SM. A network operator may wishe to implement different types of agents proposed by several suppliers, may faces numerous security management blocks that all implement the same service.

7.1.2 Proposed solution

In our solution we propose to integrate the SDN framework, presented in the chapter 5, into the architecture proposed by NM-DIPS. The architecture of this proposal is illustrated in Fig. 7.3. In this proposal the direct interface between the IDS-end and its SM is replaced by the SO that separates two modules via its NBI and SBI. By integrating this framework into the architecture, the physical deployment of IDS-end into the network will be done thanks

to the RO. It means that the SO not only manages the service lifecycle but it also opens an interfaces allowing the control of IDS-end through an externalized SM.

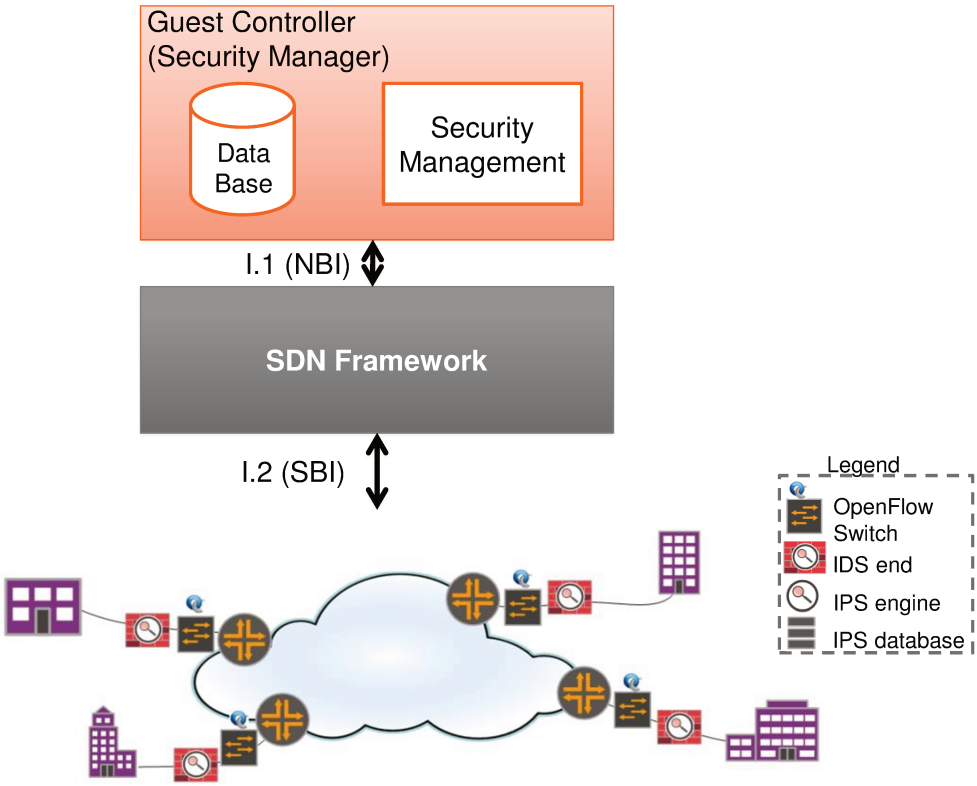

FIGURE 7.3: Global architecture of the proposal

7.1.3 BYOC Use Case : A VPN service secured by a BYOC-based IPS

In order to deploy a secured VPN connectivity protected by an IPS service, IDS-end blocks are implemented into the gateway of each site. These blocks analyze the traffic and send to the SO all suspicious packets. By receiving these packets, the SO sends them, via NBI, to a SM that analyzes the traffic in details and takes a decision. This decision can be, for example, blocking the source of the traffic. The SM sends back its decision to the SO, via NBI. Thanks to its centralized perspective of the network, the SO configures network devices to block the traffic. In the following we will firstly present the service deployment of this secured VPN. Secondly we will discuss how the control of IDS-end blocks is externalized to a GC containing a SM.

7.1.3.1 Implementing a secured VPN

Deploying a VPN service using the SDN framework is described in section 5. In the current example we enrich this service by adding an IPS service to that. The control of the VPN connection is done by the Service Operator, and the control of the IPS is done by a GC

owned and managed by an external actor or the SC. In our example we suppose that the GC is owned and managed by the SC. The GC contains all blocks permitting to deploy a SM.

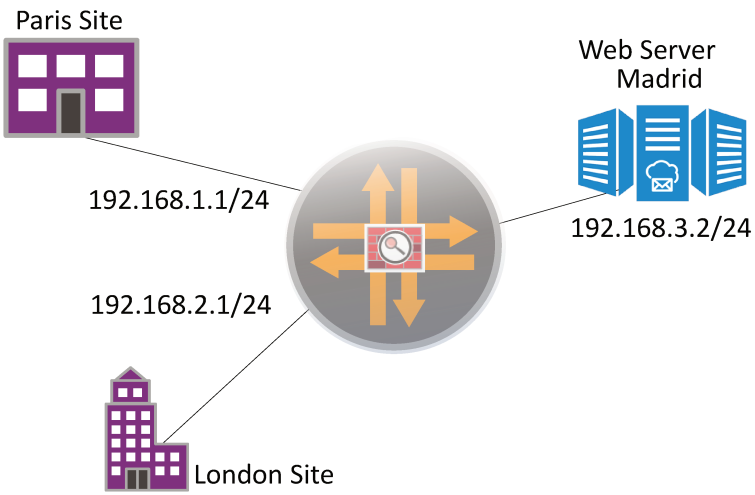

FIGURE 7.4: Physical topology of the negotiated Secured VPN service

The service model presented to the SC is illustrated in the fig. 7.4 where the service is equivalent to a router that interconnects three sites of the SC. In this representation the router embeds an IPS functionality. The fig. 7.10 illustrates the description of the negotiated service in XML. We introduce in this model the “Control Type” variable that can be “internal” or “byoc”. As it is shown in this figure the IPS service control type is “byoc”. This one allows the SO to create required blocks in order to open an NBI to the client’s GC permitting to control its IPS(s). A second variable presented in this model is the "Distribution", that describes the way the negotiated service should be deployed. In this example the IPS service is a distributed one.

Once the service is negotiated, the SRM sends the model to the SDCM. This one decomposes the general model to elementary ones. In our example, for the embedded IPS service, the SDCM generates three elementary service descriptions each one related to an IDS-end deployed within a client site. It is worth nothing to mention that each IDS-end is coupled by an OpenFlow switch that, in this example, in additional to its traditional switching role it plays the role of a firewall. The fig. 7.5 illustrates the internal architecture of the IDS-end block implemented on each site. After creating elementary service models, the SDCM sends them to the SCM. This block, firstly, creates required resources through the RO, via the interface I.7. Secondly it configures these resources in order to deploy the negotiated service, via interfaces I.5 and I.6. In our example two virtual resources, an IDS-end and its related OpenFlow switch, are created in each sites. In order to create these resources, the RO owns virtual resource templates permeating to deploy and push a first configuration on them. Once virtual resources are initiated, the SO configures the OpenFlow switch, through I.6, to pass the inbound traffic through the IDS-end.

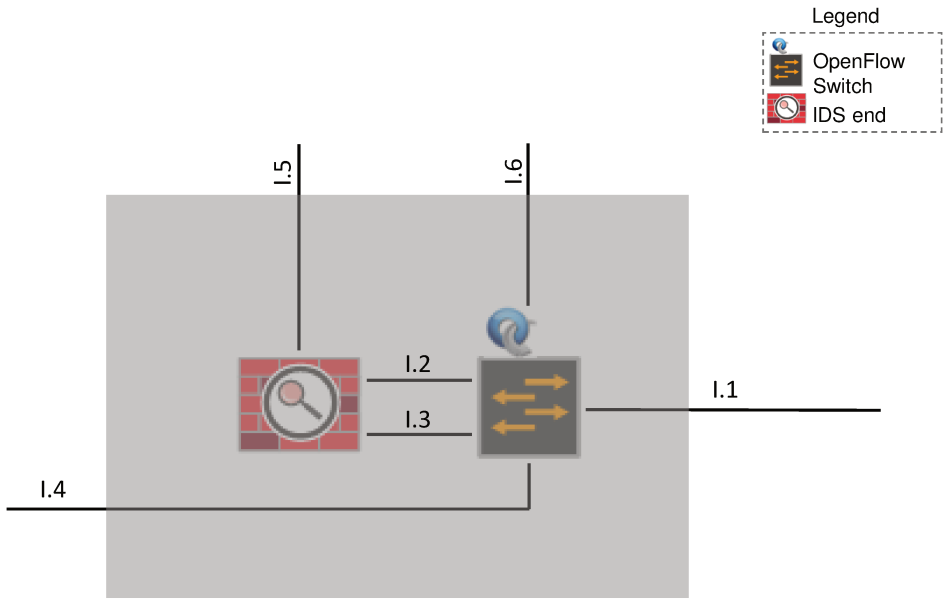

FIGURE 7.5: Internal architecture of an IDS-end module deployed on each site

7.1.3.2 Opening an IPS control interface

Once the service is deployed the SC can plug its GC into the SO thanks to the JID created during the service deployment process. The connected GC is subscribed to IDS-end "log" and "alert" messages. These messages are sent from the IDS-end to the SO through OpenFlow packets, the interface I.5 of the fig 7.5. These ones are identified as external controlled messages and are sent by the SO to the appropriate GC. The SO needs to distinguish the PacketIn messages containing "log" or "alert". For this reason, IDS-end has to specify a negotiated value for each message type. We propose here to use the value 143 and 144 for the "protocol number" field specified in the PacketIn message header. This field is specified in the OpenFlow PacketIn message by the ONF. In order to distinguish the flow concerning each GC, a DE is integrated inside the SO. By receiving PacketIn messages and by analyzing the header of these messages, the SO distinguishes the flow profile. Using this profile the DE recognizes whether the request should be processed locally in the SO or if treatment should be performed in a GC. To determine the profile of the flow, the DE uses a Decision Base (DB) which contains information on outsourced services and the identifier of the concerning GC (identified by gcId). The fig. 7.6 shows a simplified version of the DB.

Once the decision is made, the content of the received packet carrying logs or alerts, is transmitted to the appropriate GC via the SD. This one is responsible for GCs isolation, and transmission of different resource messages to their respective GC via NBI (the I.1 interface of the fig. 7.3).

N°	Data Path ID	InPort N°	IPSrc	IPDst	TCPSrc	IPProto	Action
1	00:00:00:00:00:00:00:01	1	Any	Any	Any	143	SM1
2	00:00:00:00:00:00:00:01	1	Any	Any	Any	144	SM1
3	00:00:00:00:00:00:00:03	12	Any	Any	Any	143	SM2
4	00:00:00:00:00:00:00:03	12	Any	Any	Any	144	SM2

FIGURE 7.6: A model of Decision Base implemented within the SO

7.1.3.3 Decision Engine (DE)

Fig. 7.7 shows all of tasks performed in the DE. The contribution of the BYOC concept concludes on the left side of the FlowChart shown in this figure where the DE transmits the flow coming from an outsourced security service to an external unit, called a GC. For this, by comparing the received message header and the "action" rules installed in the DB for that specific flow profile, the DE determines if the message concerns a flow coming from an IDS-end, then it finds the identifier of the related GC (gcId).

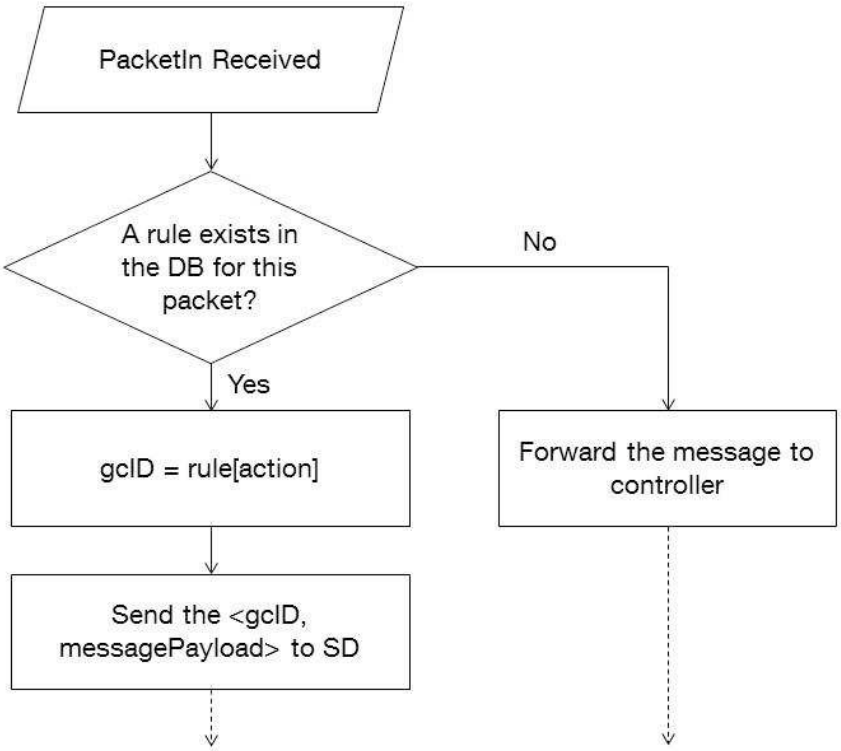

FIGURE 7.7: Decision Engine task flowchart

7.1.3.4 Decision Base (DB)

This database is comparable with the database used normally within Firewalls where there are actions like: ACCEPT, REJECT, and DROP. The difference between these ones and the DB presented in BYOC is in fields "IPProto" and "Action" where we record the type of message log / alert (in the IPProto field) and the ID of the GC, gcId (in the Action field). The values stored in this database are configured by the network administrator (operator) that manages the entire infrastructure.

7.1.3.5 Service Dispatcher (SD) and NBI

The SD is the module directly accessible by GCs. It identifies GCs using the identifier of each one (gcId), registered in the Action field of the DB. We propose here to use the XMPP protocol to implement the interface between the SD and GCs where each endpoint is identified by a JID.

7.1.3.6 Detailed components of the Guest Controller (GC)

The Fig. 7.8 shows the detailed components of the SM implemented the GC. As stated recently the SD sends the log and alert messages arriving from IDS-ends to an appropriate GC. These messages contain the specific values (143, 144) in their IPProto field. By receiving a message, the SM needs to know the origin of this message, whatever it is: log and alert. For this we propose to implement Security Proxy (SP). By examining the IPProto field, the SP decides whether a message relates to a log or an alert.

7.1.3.7 Applying the GC decision on the infrastructure

Once a decision is made by the GC, it sends a service update message to the SDCM. This decision may update a series of devices. In our example, to block an attacking traffic, the decision just updates the OpenFlow switch that is installed in front of the IDS-end. This new configuration, deployed on the switch, allows the GC to block the inbound traffic entering the customers sites (interface I.1 on the figure 7.5). The update message sent from the GC contains a service data model equivalent to the model presented in the service creation phase. Thanks to this homogeneity of models, the BYOC service update becomes transparent for the SO and the update process will be done through existing blocks of the SO.

7.1.4 Distributed IPS control plane

Opening a control interface on the IDS-end equipment through the SO allows to break down the inner modules of the SM between several GCs. The fig. 7.9 illustrates this example in details. In this example, an attack signature database is shared between multiple SMs.

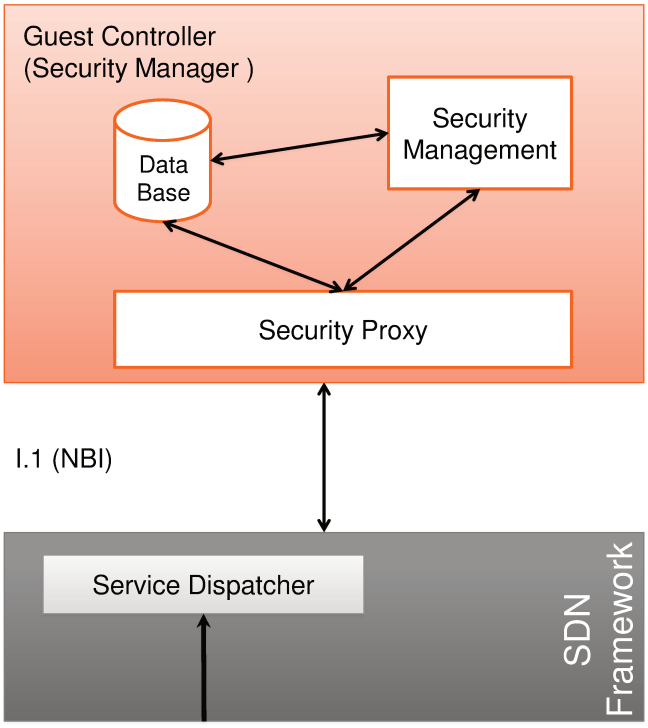

FIGURE 7.8: Internal architecture of the Security Manager

7.2 Conclusion

In this chapter we presented a BYOC-based service externalizing IPS control planes.

The architecture of this service, IPS, is based on two main blocks: a SM and its related IDS-ends. The latter are implemented in the data path and analyze the network traffic. They send via the SO all suspicious packets to the SM. This one, located at the GC, analyzes the suspicious traffic and blocks the traffic through the SO, thanks to its centralized knowledge of the network.

The deployment of a BYOCed IPS service is done through a framework presented in chapter 5. In addition to this chapter we briefly presented the way this framework can facilitate and automate the deployment of resources needed to offer a BYOC service.

Presenting a new control model, called BYOC, adds into SO architecture some internal blocks. Through the first example, we presented these new blocks in details and described the role of each one in opening a BYOC interface. The fact of using the XMPP protocol as the NBI, allows us to distribute some functions hosted within a GC. As the last proposal, we presented a distributed control plane that permits different actors to share, for any reason, a part of control implemented within their GCs.

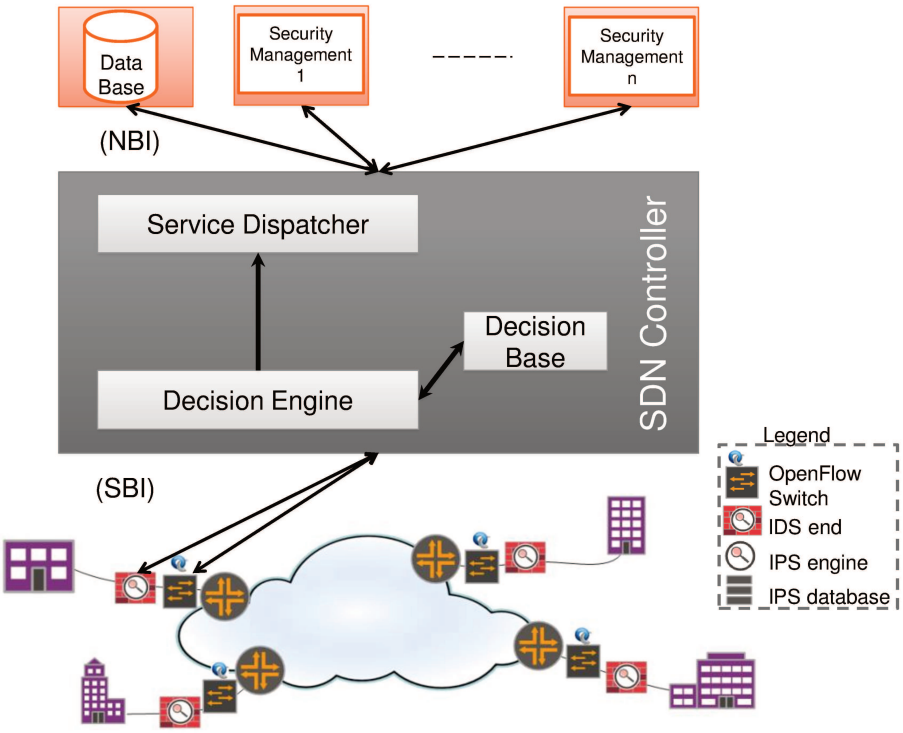

FIGURE 7.9: Distributed IPS control plane

```xml
<?xml version="1.0" encoding="utf-8"?>
<services>
        <service>
                <name>VPN</name>
                <control_type>internal</control_type>
                <interface_1>
                        <service_type>Connectivity</service_type>
                        <name>Paris_Site</name>
                        <zone>Paris</zone>
                        <configuration>
                                <ip_address>192.168.1.1</ip_address>
                                <netmask>255.255.255.0</netmask>
                        </configuration>
                </interface_1>
                <interface_2>
                        <service_type>Connectivity</service_type>
                        <name>London_Site</name>
                        <zone>London</zone>
                        <configuration>
                                <ip_address>192.168.2.1</ip_address>
                                <netmask>255.255.255.0</netmask>
                        </configuration>
                </interface_2>
                </interface_3>
                        <service_type>Compute</service_type>
                        <name>Web_Server</name>
                        <zone>Madrid</zone>
                        <configuration>
                                <ip_address>192.168.3.2</ip_address>
                                <netmask>255.255.255.0</netmask>
                                <protocol>TCP</protocol>
                                <port>80</port>
                        </configuration>
                </interface_3>
        </service>
        <service>
                <name>IPS</name>
                <control_type>byoc</control_type>
                <distribution>distributed</distribution>
        </service>
</services>
```

FIGURE 7.10: A simplified abstract model of the Secured VPN service

Chapter 8

Conclusions and Future Research

Contents

5.1 Illustrating Service Deployment Example 50
5.2 Orchestrator-based SDN Framework 52
5.2.1 Internal structure of the Service Orchestrator 52
5.2.2 Internal architecture of the Resource Orchestrator 58
5.2.3 Framework interfaces 59
5.3 Implementation 59
5.3.1 Hardware architecture 59
5.3.2 Network architecture 60
5.3.3 Software architecture 61
5.4 Conclusion 65

This dissertation is setted out to investigate the role that SDN plays in various aspects of network service control and management, and to use an SDN based framework as service management system. In this final chapter, we will review the research contributions of this dissertation, as well as discuss directions for future research.

8.1 Contributions

The following are the main research contributions of this dissertation.

— **A double-sided service lifecycle and data model** (Chapter 4)
At the beginning of this dissertation the SDN based service management was one of the non-answered questions. There were several initiatives to define the perimeter of SDN architecture layers, several SDN controllers were in the design and development phase, and developed SDN controllers and frameworks were deployed each one for specific research topics. Some of SDN-based services were deployed by internal SDN controller's functions and some of them were controlled by applications developed at the top of the controller programing the network via the controllers NBI. Due to the nature of ongoing projects, and the fact that there were not any clear definition of SDN controller core functions and northbound applications, defining the border of these two layers, i.e. SDN controller and SDN applications, helping to delimitate

their perimeter and aiming to define an NBI was almost impossible. ONF had just started the NBI group activities aiming to define an NBI answering requirements of most of applications. However, this work was far from being realized, because defining a standard NBI, that is an application interface, requires a careful analysis of several implementations and the feedback gained by all those implementations.

In order to define a reference SDN-based service provisioning framework allowing to define the control and application layer edge, a service lifecycle analysis had to take place. At the first time, in Section 4.1, we presented the service lifecycle analysis in two point of views: client and operator. The fist view, client-side service lifecycle, discusses different phases in which a service customer (or client) can be during the service lifecycle. This analysis is held based-on the application and service classification that we have previously done in Section 2.5.2. According to this classification, a service customer can use the service management interface to manage three types of services. The first one is the case where the customer requests and configures a service. The second type is the customer who monitors his service, and the third one is the customer who, using the management interface, receives some service parameters based on which he reconfigures or updates that service. Based on this analysis, the client-side service lifecycle can be modified. In this section we analyzed all phases that each service type might add to the service lifecycle. On the other side, the operator-side service lifecycle analysis presents a service lifecycle model representing all phases an operator should cross to deploy, configure and maintain a service. This double-sided analysis allows to determine actions that each service customer and operator can take on a service that is the common object between a customer and an operator.

At the second time, we presented the data model of each lifecycle sides based on a double-layered data model approach. In this approach a service can be modeled in two data models: service and device, and an elementary model, called transformation, defines how one of these two models can be transformed to the other one. The service model is a general and simplified model of the service presented to the service customer. And the device model is the technical definition of the device configuration generated based on the negotiated service model. The service object, shared between the operator and the customer is described in the service model. Consequently, the client-side service lifecycle is using the service model and all phases of the lifecycle are based on this model. The service model crosses down the operator-side lifecycle and is transformed to one or different device models. In Section 4.2 we discuss the model used or generated by each operator-side service lifecycle phase. In this section we discussed also the transformation type each step might do to convert a model from a service to a device one.

— **A service management framework based on SDN paradigm** (Chapter 5)

The service lifecycle analysis gives us a tool to determine all activities an operator should do to manage a service. In Chapter 5, based on the operator-side service lifecycle, we propose a framework through which a service model presented to the

customer, is transformed to device models deployed on resources.
The architecture of this framework is based on a double-layered system managing the service lifecycle through two orchestrators: service orchestrator and resource orchestrator. The first one puts together all functions allowing operator to manage a service vertically, and the second one manages resources needed by the first one to deploy a service.

— **Bring Your Own Control BYOC service** (Chapter 6) The proposed framework gives rise to a system deploying and managing services. It opens an interface to the customers' side. In this chapter we present a new service control model, called Bring Your Own Control (BYOC) that follows the Type 3 applications model discussed in Section 2.5.2.
In the first part of this chapter we introduce BYOC as a concept allowing to delegate, through the NBI, the control of all or a part of a service to an external controller, called "*Guest Controller (GC)*". The latter might be managed by the same customer requesting and consuming the service or by a third party operator.
Opening a control interface at the top of the SDN platform requires some specifications at the NBI level. We discussed at the second part of this chapter the requirements of the NBI allowing to open the BYOC API. Based on these requirements we proposed the use of XMPP as the protocol allowing to deploy such an API.

8.2 Future researches

The framework and its multilevel service provisioning interface introduced in this dissertation, provides a new service type, called BYOC, to future research. While this work has demonstrated the potential of opening a tuned control access to a service though a dynamic IPS service in Chapter 7, many opportunities for extending the scope of this thesis remain. In this section we discuss some of these opportunities.

8.2.0.1 A detailed study of the theoretical and technical approach of the BYOC

Opening up the control interface to a GC by BYOC concept may create some new revenue resources. Indeed, BYOC allows not only to the service customer to implement its personalized control algorithm and fully managing its service, but also it allows the operator to monetize the openness of its SDN-based system. We presented the Earn as You Bring (EaYB) business model allowing the operator to resell a service to a customer controlled by third party GC [99].

Opening the control platform and integrating an external Controller in a service production chain, however, may create some security and complexity problems. One of the fundamental issues concerns the impact of the BYOC concept on the performance of the network

controller. In fact, externalizing the control engine of a service to a GC may create a significant delay on decision step of the controller, the delay that will have a direct effect on the QoS. The second issue concerns the confidentiality of information available to the GC. By opening its control interface, the operator provides the GC with information that may be confidential. To avoid this type of security problem, a data access control mechanism must take place, through which the operator controls all the data communicated between the controller and the GC while maintaining the flexibility of the BYOC model [100].

The analysis of advantages of BYOC model and the complexity and security issues that BYOC may bring to the service management process can be the subject of a future work. This analysis requires a more sophisticated study of this concept, the potential business model that it can introduce (ex. EaYB), the methods and protocols used to implement the northern interface and to control the access to resources exposed to the GC, and the real impact of this type of services on the performance of services.

8.2.1 BYOC as a key enabler to flexible NFV service chaining

A NFV SC defines a set of Service Function (SF)s and the order of these SF through which a packet should pass in the downlink and uplink traffic. Chaining network elements to create a service is not a new subject. Indeed, legacy network services are made of several network functions which are hardwired back-to-back. These solutions however remain difficult to deploy and expensive to change.

As soon as software-centric networking technologies, such as SDN and NFV brought the promise of programmability and flexibility to the network, the flexible service chaining became one of the academic challenges. The flexible service chaining consists in choosing the relevant SC through the analysis of traffic. There are several initiatives trying to propose an architecture for creation of Service Function Chaining (SFC) [101, 102, 103]. Among these solutions, IETF [102] and ONF [101] propose to use a traffic classifier at the ingress point of the SC allowing to classify traffic flows based on policies. This classification allows to specify a path ID to the flow used to forward the flow on a specific path, called Service Function Path (SFP). In the ONF proposal the SDNC has a central role, where it sets up SFPs by programming Service Function Forwarder (SFF) to steer the flow through the sequence of SF instances. It also locates and program the flow classifier through the SBI allowing to classify a flow.

Applying the BYOC concept to the approach proposed by ONF consists in opening a control interface between the SDNC and a GC that implements all functions needed to classify the flow and reconfigure the SFF, and the flow classifier based on new flows arrived on the classifier. Delegating the control of the SFC to the customer, gives more flexibility, visibility and freedom to the customer to create a flexible SFC based on its customized path computation algorithms and its applications requirements. On the other hand, a BYOC based SFC allows the Service Provider to lighten the service OpEx.

8.2.2 BYOC as a key concept leading to 5G dynamic network slicing

5th generation (5G) networks needs to support new demands from a wide variety of service groups from e-health to broadcast services [104]. In order to cover all these domains 5G networks need to support diverse requirements in terms of network availability, throughput, capacity and latency [105]. In order to deliver services to such wide domains and to answer these various requirements, network slicing has been introduced in 5G networks [106, 107, 108]. Network slicing allows operators to establish different capabilities for each service group and serve multiple tenants in parallel.

SDN will play an important role in shifting to dynamic network slicing [109, 110, 111]. The control and forwarding plane decoupling leads to separation of software from hardware, the concept that allows to share the infrastructure between different tenants each one using one or several slices of the network. In [112] "Dynamic programmability and control" brought by SDN, is presented as one of the key principles guiding the dynamic network slicing. In this work the authors argue that "the dynamic programming of network slices can be accomplished either by custom programs or within an automation framework driven by analytics and machine learning."

Applying the BYOC concept to 5G networks leads to externalizing the control of one or several slices to a GC owned or managed by a customer, an Over The Top (OTT), or an OSS. We argue that this openness is totally in line with the dynamic programmability and control principle of 5G networks presented in [112]. The innovative algorithms implemented within the GC controlling the slice of the network empowers promising value-added services and business models. However, this externalization creates some management and orchestration issues presented previously in [109].

Bibliography

[1] Andreas Metzger and Clarissa Cassales Marquezan. "Future Internet Apps: The Next Wave of Adaptive Service-Oriented Systems?" In: *Towards a Service-Based Internet: 4th European Conference, ServiceWave 2011, Poznan, Poland, October 26-28, 2011. Proceedings*. Ed. by Witold Abramowicz et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 230–241. ISBN: 978-3-642-24755-2. DOI: 10.1007/978-3-642-24755-2_22. URL: http://dx.doi.org/10.1007/978-3-642-24755-2_22.

[2] Theophilus Benson, Aditya Akella, and David Maltz. "Unraveling the Complexity of Network Management". In: *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*. NSDI'09. Boston, Massachusetts: USENIX Association, 2009, pp. 335–348. URL: http://dl.acm.org/citation.cfm?id=1558977.1559000.

[3] Nick McKeown. "Software-defined networking". In: vol. 17. 2. 2009, pp. 30–32.

[4] Hyojoon Kim and N. Feamster. "Improving network management with software defined networking". In: *Communications Magazine, IEEE* 51.2 (2013), pp. 114–119. ISSN: 0163-6804. DOI: 10.1109/MCOM.2013.6461195.

[5] R. Mijumbi et al. "Network Function Virtualization: State-of-the-Art and Research Challenges". In: vol. 18. 1. 2016, pp. 236–262. DOI: 10.1109/COMST.2015.2477041.

[6] B. Han et al. "Network function virtualization: Challenges and opportunities for innovations". In: vol. 53. 2. 2015, pp. 90–97. DOI: 10.1109/MCOM.2015.7045396.

[7] D. Kreutz et al. "Software-Defined Networking: A Comprehensive Survey". In: *Proceedings of the IEEE* 103.1 (2015), pp. 14–76. ISSN: 0018-9219. DOI: 10.1109/JPROC.2014.2371999.

[8] Internet World Stats. *"World Internet Usage and Population Statistics"*. 2017. URL: http://www.internetworldstats.com/stats.htm.

[9] Andrew T. Campbell et al. "A Survey of Programmable Networks". In: *SIGCOMM Comput. Commun. Rev.* 29.2 (Apr. 1999), pp. 7–23. ISSN: 0146-4833. DOI: 10.1145/505733.505735. URL: http://doi.acm.org/10.1145/505733.505735.

[10] J. Biswas et al. "The IEEE P1520 standards initiative for programmable network interfaces". In: *IEEE Communications Magazine* 36.10 (1998), pp. 64–70. ISSN: 0163-6804. DOI: 10.1109/35.722138.

[11] D. L. Tennenhouse et al. "A survey of active network research". In: *IEEE Communications Magazine* 35.1 (1997), pp. 80–86. ISSN: 0163-6804. DOI: 10.1109/35.568214.

[12] C. Tsarouchis et al. "A policy-based management architecture for active and programmable networks". In: *IEEE Network* 17.3 (2003), pp. 22–28. ISSN: 0890-8044. DOI: 10.1109/MNET.2003.1201473.

[13] J. V. Millor and J. S. Fernandez. "A network management approach enabling active and programmable Internets". In: *IEEE Network* 19.1 (2005), pp. 18–24. ISSN: 0890-8044. DOI: 10.1109/MNET.2005.1383436.

[14] John Strassner. "Policy-Based Network Management". In: *Policy-Based Network Management*. Ed. by John Strassner. The Morgan Kaufmann Series in Networking. Burlington: Morgan Kaufmann, 2004. DOI: https://doi.org/10.1016/B978-155860859-7/50033-5. URL: http://www.sciencedirect.com/science/article/pii/B9781558608597500335.

[15] R. Yavatkar, D. Pendarakis, and R. Guerin. *RFC 2753: A Framework for Policy-based Admission Control*. Tech. rep. IETF, 2000. URL: www.ietf.org/rfc/rfc2753.txt.

[16] J. Boyle et al. *The COPS (Common Open Policy Service) Protocol*. Ed. by D. Durham. United States, 2000.

[17] HP. *HP 8200 zl Switch Series*. 2013. URL: http://www8.hp.com/uk/en/products/networking-switches/product-detail.html?oid=3437443.

[18] Arista. *Arista 7150 Series*. 2013. URL: https://www.arista.com/en/products/7150-series.

[19] O.E. Ferkouss et al. "A 100Gig network processor platform for openflow". In: *7th International Conference on Network and Service Management (CNSM), 2011* (2011), pp. 1–4.

[20] Nick McKeown et al. "OpenFlow: Enabling Innovation in Campus Networks". In: *SIGCOMM Comput. Commun. Rev.* 38.2 (Mar. 2008), pp. 69–74. ISSN: 0146-4833. DOI: 10.1145/1355734.1355746. URL: http://doi.acm.org/10.1145/1355734.1355746.

[21] ONF. *"Open Networking Foundation"*. 2015. URL: \url{https://www.opennetworking.org}.

[22] B. Salisbury. *OpenFlow: Proactive vs Reactive Flows*. 2013. URL: http://networkstatic.net/openflow-proactive-vs-reactive-flows/.

[23] Open Networking Foundation. *OpenFlow Switch Specification, Version 1.5.0 (Protocol version 0x06)*. TS ONF TS-020. Open Networking Foundation, Dec. 2014. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf.

[24] Nick McKoewn. *OpenFlow (Or: "Why can't I innovate in my wiring closet?* 2014. URL: https://www.coursehero.com/file/10419252/WK7-McKeown-OpenFlow/.

[25] Natasha Gude et al. "NOX: Towards an Operating System for Networks". In: *SIGCOMM Comput. Commun. Rev.* 38.3 (July 2008), pp. 105–110. ISSN: 0146-4833. DOI: 10.1145/1384609.1384625. URL: http://doi.acm.org/10.1145/1384609.1384625.

[26] Yiannis Yiakoumis et al. "Slicing Home Networks". In: HomeNets '11 (2011), pp. 1–6. DOI: 10.1145/2018567.2018569. URL: http://doi.acm.org/10.1145/2018567.2018569.

[27] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. "A Scalable, Commodity Data Center Network Architecture". In: *SIGCOMM Comput. Commun. Rev.* 38.4 (Aug. 2008), pp. 63–74. ISSN: 0146-4833. DOI: 10.1145/1402946.1402967. URL: http://doi.acm.org/10.1145/1402946.1402967.

[28] Martin Casado et al. "Ethane: Taking Control of the Enterprise". In: *SIGCOMM Comput. Commun. Rev.* 37.4 (Aug. 2007), pp. 1–12. ISSN: 0146-4833. DOI: 10.1145/1282427.1282382. URL: http://doi.acm.org/10.1145/1282427.1282382.

[29] *Ryu SDN Controller*. Accessed: 2013-03-30. URL: https://osrg.github.io/ryu/.

[30] *Trema SDN Controller*. Accessed: 2013-04-28. URL: https://trema.github.io/trema/.

[31] "Floodlight OpenFlow Controller". In: 2014. URL: http://www.projectfloodlight.org/floodlight.

[32] *The OpenDaylight SDN Platform*. Accessed: 2014. URL: https://www.opendaylight.org/.

[33] Pankaj Berde et al. "ONOS: Towards an Open, Distributed SDN OS". In: *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*. HotSDN '14. Chicago, Illinois, USA: ACM, 2014, pp. 1–6. ISBN: 978-1-4503-2989-7. DOI: 10.1145/2620728.2620744. URL: http://doi.acm.org/10.1145/2620728.2620744.

[34] R. Khondoker et al. "Feature-based comparison and selection of Software Defined Networking (SDN) controllers". In: *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*. 2014, pp. 1–7. DOI: 10.1109/WCCAIS.2014.6916572.

[35] Ed. R. Enns et al. *Network Configuration Protocol (NETCONF)*. RFC 6241. 2011. URL: https://tools.ietf.org/html/rfc6241.

[36] I. GUIS. *The SDN Gold Rush To The Northbound API*. 2012. URL: https://www.sdxcentral.com/articles/contributed/the-sdn-gold-rush-to-the-northbound-api/2012/11/.

[37] B. SALISBURY. *The Northbound API- A Big Little Problem*. 2012. URL: http://networkstatic.net/the-northbound-api-2/.

[38] I. Pepelnjak. *SDN CONTROLLER NORTHBOUND API IS THE CRUCIAL MISSING PIECE*. 2012. URL: http://blog.ipspace.net/2012/09/sdn-controller-northbound-api-is.html.

[39] B. Casemore. *Northbound API: The Standardization Debate*. 2012.

[40] R. G. Little. *ONF to standardize northbound API for SDN applications?* 2013. URL: http://searchsdn.techtarget.com/news/2240206604/ONF-to-standardize-northbound-API-for-SDN-applications.

[41] A. Josey. *POSIX - Austin Joint Working Group*. URL: POSIX-AustinJointWorkingGroup.

[42] Andreas Voellmy, Hyojoon Kim, and Nick Feamster. "Procera: A Language for High-level Reactive Network Control". In: *Proceedings of the First Workshop on Hot Topics*

in Software Defined Networks. HotSDN '12. Helsinki, Finland: ACM, 2012, pp. 43–48. ISBN: 978-1-4503-1477-0. DOI: 10.1145/2342441.2342451. URL: http://doi.acm.org/10.1145/2342441.2342451.

[43] Nate Foster et al. "Frenetic: A Network Programming Language". In: *SIGPLAN Not.* 46.9 (Sept. 2011), pp. 279–291. ISSN: 0362-1340. DOI: 10.1145/2034574.2034812. URL: http://doi.acm.org/10.1145/2034574.2034812.

[44] Matthew Monaco, Oliver Michel, and Eric Keller. "Applying Operating System Principles to SDN Controller Design". In: *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*. HotNets-XII. College Park, Maryland: ACM, 2013, 2:1–2:7. ISBN: 978-1-4503-2596-7. DOI: 10.1145/2535771.2535789. URL: http://doi.acm.org/10.1145/2535771.2535789.

[45] Pascal Menezes et al. "North Bound Interface Working Group (NBI-WG) Charter". In: Open Networking Foundation, 2013. URL: https://www.opennetworking.org/images/stories/downloads/working-groups/charter-nbi.pdf.

[46] Fei Hu, Qi Hao, and Ke Bao. "A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation". In: *IEEE Communications Surveys Tutorials* 16.4 (2014), pp. 2181–2206. ISSN: 1553-877X. DOI: 10.1109/COMST.2014.2326417.

[47] Nikhil H et al. "Aster*x: Load-Balancing Web Traffic over Wide-Area Networks". In: ().

[48] Richard Wang, Dana Butnariu, and Jennifer Rexford. "OpenFlow-based Server Load Balancing Gone Wild". In: Hot-ICE'11 (2011), pp. 12–12. URL: http://dl.acm.org/citation.cfm?id=1972422.1972438.

[49] Nikhil Handigol et al. "Plug-n-Serve: Load-Balancing Web Traffic using OpenFlow". In: (2009). URL: http://conferences.sigcomm.org/sigcomm/2009/demos/sigcomm-pd-2009-final26.pdf.

[50] Brandon Heller et al. "ElasticTree: Saving Energy in Data Center Networks". In: NSDI'10 (2010), pp. 17–17. URL: http://dl.acm.org/citation.cfm?id=1855711.1855728.

[51] M. Scharf et al. "Dynamic VPN Optimization by ALTO Guidance". In: (2013), pp. 13–18.

[52] Xuan-Nam Nguyen et al. "Optimizing Rules Placement in OpenFlow Networks: Trading Routing for Better Efficiency". In: *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*. HotSDN '14. Chicago, Illinois, USA: ACM, 2014, pp. 127–132. ISBN: 978-1-4503-2989-7. DOI: 10.1145/2620728.2620753. URL: http://doi.acm.org/10.1145/2620728.2620753.

[53] Manu Bansal et al. "OpenRadio: A Programmable Wireless Dataplane". In: *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*. HotSDN '12. Helsinki, Finland: ACM, 2012, pp. 109–114. URL: http://doi.acm.org/10.1145/2342441.2342464.

[54] Jonathan Vestin et al. "CloudMAC: Towards Software Defined WLANs". In: *SIGMOBILE Mob. Comput. Commun. Rev.* 16.4 (Feb. 2013), pp. 42–45. ISSN: 1559-1662.

DOI: 10.1145/2436196.2436217. URL: http://doi.acm.org/10.1145/2436196.2436217.

[55] Aditya Gudipati et al. "SoftRAN: Software Defined Radio Access Network". In: HotSDN '13 (2013), pp. 25–30. DOI: 10.1145/2491185.2491207. URL: http://doi.acm.org/10.1145/2491185.2491207.

[56] P. Dely, A. Kassler, and N. Bayer. "OpenFlow for Wireless Mesh Networks". In: (2011), pp. 1–6.

[57] L.E. Li, Z.M. Mao, and J. Rexford. "Toward Software-Defined Cellular Networks". In: (2012), pp. 7–12.

[58] Srikanth Sundaresan et al. "Broadband Internet Performance: A View from the Gateway". In: *Proceedings of the ACM SIGCOMM 2011 Conference*. SIGCOMM '11. Toronto, Ontario, Canada: ACM, 2011, pp. 134–145. URL: http://doi.acm.org/10.1145/2018436.2018452.

[59] Martin Casado et al. "SANE: A Protection Architecture for Enterprise Networks". In: *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*. USENIX-SS'06. Vancouver, B.C., Canada: USENIX Association, 2006. URL: http://dl.acm.org/citation.cfm?id=1267336.1267346.

[60] R. Braga, E. Mota, and A. Passito. "Lightweight DDoS flooding attack detection using NOX/OpenFlow". In: *IEEE 35th Conference on Local Computer Networks (LCN), 2010*. 2010, pp. 408–415.

[61] V.K. Gurbani et al. "Abstracting network state in Software Defined Networks (SDN) for rendezvous services". In: *IEEE International Conference on Communications (ICC), 2012*. 2012, pp. 6627–6632. DOI: 10.1109/ICC.2012.6364858.

[62] ETSI. *Network Functions Virtualisation (NFV); Management and Orchestration*. TS ETSI GS NFV-MAN 001, DGS/NFV-MAN001. European Telecommunications Standards Institute, Dec. 2014. URL: http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf.

[63] E. Rosen, A. Viswanathan, and R. Callon. "Multiprotocol Label Switching Architecture, RFC 3031". In: Jan. 2001. URL: https://tools.ietf.org/html/rfc3031.

[64] Y. Rekhter and S. Hares. "A Border Gateway Protocol 4 (BGP-4), RFC 4271". In: Jan. 2006. URL: https://tools.ietf.org/html/rfc4271.

[65] J. Moy. "OSPF Version 2, RFC 1247". In: July 1991. URL: https://tools.ietf.org/html/rfc1247.

[66] T. Bates et al. "Multiprotocol Extensions for BGP-4, RFC 4760". In: Jan. 2007. URL: https://tools.ietf.org/html/rfc4760.

[67] L. Andersson, I. Andersson, and B. Thomas. "LDP Specification, RFC 5036". In: Oct. 2007. URL: https://tools.ietf.org/html/rfc5036.

[68] Krzysztof Szarkowicz and Antonio Monge. *MPLS in the SDN Era*. O Reilly Media, Jan. 2016.

[69] Ankur Singla and Bruno Rijsman. *Day One: Understanding OpenContrail Architecture*. Juniper Networks, Nov. 2013.

[70] Ali Reza Sharafat et al. "MPLS-TE and MPLS VPNS with Openflow". In: *Proceedings of the ACM SIGCOMM 2011 Conference*. SIGCOMM '11. Toronto, Ontario, Canada: ACM, 2011, pp. 452–453. ISBN: 978-1-4503-0797-0. DOI: 10.1145/2018436.2018516. URL: http://doi.acm.org/10.1145/2018436.2018516.

[71] S. Das et al. "MPLS with a simple OPEN control plane". In: *2011 Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference*. 2011, pp. 1–3.

[72] Chappell Caroline. "Creating the Programmable Network, The Business case for Netconf/YANG in network devices". In: Oct. 2011. URL: http://www.tail-f.com/wordpress/wp-content/uploads/2013/10/HR-Tail-f-NETCONF-WP-10-08-13.pdf.

[73] "Tail-f Network Control System (NCS) – Datasheet". In: 2012. URL: http://www.tail-f.com/wordpress/wp-content/uploads/2014/01/Tail-f-Datasheet-NCS.pdf.

[74] M. Bjorklund. *YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)*. RFC 6020. 2010.

[75] Yuri Demchenko and Xiaomin Chen. "GYESERS Project, Service Delivery Framework and Services Lifecycle Management in on-demand services/resources provisioning. WP2/WP3 Technical document, Version 0.2". In: 2012.

[76] C. Moberg and S. Vallin. "A two-layered data model approach for network services". In: vol. 54. 3. 2016, pp. 76–80. DOI: 10.1109/MCOM.2016.7432175.

[77] Stefan Wallin and Claes Wikström. "Automating Network and Service Configuration Using NETCONF and YANG". In: *Proceedings of the 25th International Conference on Large Installation System Administration*. LISA'11. Boston, MA: USENIX Association, 2011, pp. 22–22. URL: http://dl.acm.org/citation.cfm?id=2208488.2208510.

[78] Paul Pichler et al. "Imperative versus Declarative Process Modeling Languages: An Empirical Investigation". In: *Business Process Management Workshops: BPM 2011 International Workshops, Clermont-Ferrand, France, August 29, 2011, Revised Selected Papers, Part I*. Ed. by Florian Daniel, Kamel Barkaoui, and Schahram Dustdar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 383–394. ISBN: 978-3-642-28108-2. DOI: 10.1007/978-3-642-28108-2_37. URL: http://dx.doi.org/10.1007/978-3-642-28108-2_37.

[79] Dirk Fahland et al. "Declarative versus Imperative Process Modeling Languages: The Issue of Understandability". In: *Enterprise, Business-Process and Information Systems Modeling: 10th International Workshop, BPMDS 2009, and 14th International Conference, EMMSAD 2009, held at CAiSE 2009, Amsterdam, The Netherlands, June 8-9, 2009. Proceedings*. Ed. by Terry Halpin et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 353–366. ISBN: 978-3-642-01862-6. DOI: 10.1007/978-3-642-01862-6_29. URL: http://dx.doi.org/10.1007/978-3-642-01862-6_29.

[80] G. Di Battista, M. Rimondini, and G. Sadolfo. "Monitoring the status of MPLS VPN and VPLS based on BGP signaling information". In: *2012 IEEE Network Operations and Management Symposium*. 2012, pp. 237–244. DOI: 10.1109/NOMS.2012.6211904.

[81] Omar Sefraoui, Mohammed Aissaoui, and Mohsine Eleuldj. "OpenStack: Toward an Open-source Solution for Cloud Computing". In: *International Journal of Computer Applications* 55.3 (2012), pp. 38–42.

[82] Yu Huanle, Shi Weifeng, and Bai Tingting. "An OpenStack-Based Resource Optimization Scheduling Framework". In: *Sixth International Symposium on Computational Intelligence and Design (ISCID), 2013*. Vol. 1. 2013, pp. 261–264. DOI: 10.1109/ISCID.2013.72.

[83] R.L.S. de Oliveira et al. "Using Mininet for emulation and prototyping Software-Defined Networks". In: *IEEE Colombian Conference on Communications and Computing (COLCOM), 2014*. 2014, pp. 1–6. DOI: 10.1109/ColComCon.2014.6860404.

[84] A. Aflatoonian et al. "An asynchronous push/pull communication solution for Northbound Interface of SDN based on XMPP". In: *10eme Conference International Gestion de REseaux et de Services - GRES 2014, Paris, France.* 2014.

[85] Manish Bhide et al. "Adaptive Push-Pull: Disseminating Dynamic Web Data". In: *IEEE Trans. Comput.* 51.6 (June 2002), pp. 652–668. ISSN: 0018-9340. DOI: 10.1109/TC.2002.1009150. URL: http://dx.doi.org/10.1109/TC.2002.1009150.

[86] Xinyang Feng, Jianjing Shen, and Ying Fan. "REST: An alternative to RPC for Web services architecture". In: *ICFIN 2009. First International Conference on Future Information Networks, 2009.* Beijing, China., 2009, pp. 7–10. DOI: 10.1109/ICFIN.2009.5339611.

[87] VMWare. "VMWare vCloud Director". In: 2014. URL: http://www.vmware.com/products/vcloud-director/.

[88] Roy Thomas Fielding. "Architectural Styles and the Design of Network-based Software Architectures". PhD thesis. 2000. ISBN: 0-599-87118-0. URL: www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[89] I Fette and A Melnikov. "The WebSocket Protocol, RFC 6455". PhD thesis. Decembre, 2011. URL: http://tools.ietf.org/html/rfc6455.

[90] Michael Franklin and Stan Zdonik. "Data in Your Face: Push Technology in Perspective". In: *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*. SIGMOD '98. Seattle, Washington, USA: ACM, 1998, pp. 516–519. ISBN: 0-89791-995-5. DOI: 10.1145/276304.276360. URL: http://doi.acm.org/10.1145/276304.276360.

[91] P Saint-Andre and Ed. "Extensible Messaging and Presence Protocol (XMPP): Core, RFC 3920". In: 2004. URL: http://www.ietf.org/rfc/rfc3920.txt.

[92] "XMPP Extensions". In: URL: http://xmpp.org/xmpp-protocols/xmpp-extensions/.

[93] "XEP-0072: SOAP Over XMPP". In: URL: http://xmpp.org/extensions/xep-0072.html.

[94] A. Hornsby and R. Walsh. "From instant messaging to cloud computing, an XMPP review". In: *Consumer Electronics (ISCE), 2010 IEEE 14th International Symposium on*. Germany, 2010, pp. 1–6. DOI: 10.1109/ISCE.2010.5523293.

[95] Johannes Wagener et al. "XMPP for cloud computing in bioinformatics supporting discovery and invocation of asynchronous web services". English. In: vol. 10. 1. BioMed Central, 2009, pp. 1–12. DOI: 10.1186/1471-2105-10-279. URL: http://dx.doi.org/10.1186/1471-2105-10-279.

[96] "XEP-0060: Publish-Subscribe". In: 2010. URL: http://www.xmpp.org/extensions/xep-0060.html.

[97] Xinyou Zhang, Chengzhong Li, and Wenbin Zheng. "Intrusion prevention system design". In: *The Fourth International Conference on Computer and Information Technology, 2004. CIT '04.* 2004, pp. 386–390. DOI: 10.1109/CIT.2004.1357226.

[98] Legouge Pascal et al. "ALADDIN : Fighting Cloud DDoS!" In: *Orange Labs Research Exhibition Demonstration*. 2013.

[99] A. Aflatoonian et al. "BYOC: Bring Your Own Control a new concept to monetize SDN's openness". In: *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. 2015, pp. 1–5. DOI: 10.1109/NETSOFT.2015.7116147.

[100] Hao Jiang et al. "A Secure Multi-Tenant Framework for SDN". In: *Proceedings of the 9th International Conference on Security of Information and Networks*. SIN '16. Newark, NJ, USA: ACM, 2016, pp. 40–44. ISBN: 978-1-4503-4764-8. DOI: 10.1145/2947626.2947641. URL: http://doi.acm.org/10.1145/2947626.2947641.

[101] ONF. *"L4-L7 Service Function Chaining Solution Architecture"*. 2015. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/L4-L7_Service_Function_Chaining_Solution_Architecture.pdf.

[102] Ed. J. Halpern and Ed. C. Pignataro. *Service Function Chaining (SFC) Architecture*. 2015. URL: https://tools.ietf.org/html/rfc7665.

[103] ETSI. *Network Functions Virtualisation (NFV); Architectural Framework*. TS ETSI GS NFV 002, DGS/NFV-002. European Telecommunications Standards Institute, Oct. 2013. URL: http://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf.

[104] NGMN Alliance. "5G white paper". In: (2015).

[105] Salah Eddine Elayoubi et al. "5g service requirements and operational use cases: Analysis and metis ii vision". In: *European Conference on Networks and Communications (EuCNC), 2016*. IEEE. 2016, pp. 158–162.

[106] NGMN Alliance. "Description of network slicing concept". In: *NGMN 5G P* 1 (2016).

[107] A Galis, K Makhijan, and D Yu. *Autonomic Slice Networking-Requirements and Reference Model, October 2016*. URL: https://tools.ietf.org/html/draft-galis-anima-autonomic-slice-networking-01.

[108] Menglan Jiang, Massimo Condoluci, and Toktam Mahmoodi. "Network slicing management & prioritization in 5G mobile systems". In: *European Wireless*. 2016.

[109] Jose Ordonez-Lucena et al. "Network Slicing for 5G with SDN/NFV: Concepts, Architectures, and Challenges". In: *IEEE Communications Magazine* 55.5 (2017), pp. 80–87.

[110] *TR-526, Applying SDN Architecture to 5G Slicing*. 2016. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/Applying_SDN_Architecture_to_5G_Slicing_TR-526.pdf.

[111] Akram Hakiri and Pascal Berthou. "Leveraging SDN for the 5G networks: trends, prospects and challenges". In: *arXiv preprint arXiv:1506.02876* (2015).

[112] *Dynamic end-to-end network slicing for 5G*. 2016. URL: https://resources.ext.nokia.com/asset/200339.

Résumé

Au cours des dernières décennies, les fournisseurs de services (SP) ont eu à gérer plusieurs générations de technologies redéfinissant les réseaux et nécessitant de nouveaux modèles économiques. Cette évolution continue du réseau offre au SP l'opportunité d'innover en matière de nouveaux services tout en réduisant les coûts et en limitant sa dépendance auprès des équipementiers. L'émergence récente du paradigme de la virtualisation modifie profondément les méthodes de gestion des services réseau. Ces derniers évoluent vers l'intégration d'une capacité « à la demande » dont la particularité consiste à permettre aux clients du SP de pouvoir les déployer et les gérer de manière autonome et optimale. Pour offrir une telle souplesse de fonctionnement, le SP doit pouvoir s'appuyer sur une plateforme de gestion permettant un contrôle dynamique et programmable du réseau. Nous montrons dans cette thèse qu'une telle plate-forme peut être fournie grâce à la technologie SDN (Software-Defined Networking).

Nous proposons dans un premier temps une caractérisation de la classe de services réseau à la demande. Les contraintes de gestion les plus faibles que ces services doivent satisfaire sont identifiées et intégrées à un modèle abstrait de leur cycle de vie. Celui-ci détermine deux vues faiblement couplées, l'une spécifique au client et l'autre au SP. Ce cycle de vie est complété par un modèle de données qui en précise chacune des étapes.

L'architecture SDN ne prend pas en charge toutes les étapes du cycle de vie précédent. Nous introduisons un Framework original qui encapsule le contrôleur SDN, et permet la gestion de toutes les étapes du cycle de vie. Ce Framework est organisé autour d'un orchestrateur de services et d'un orchestrateur de ressources communiquant via une interface interne. L'exemple du VPN MPLS sert de fil conducteur pour illustrer notre approche. Un PoC basé sur le contrôleur OpenDaylight ciblant les parties principales du Framework est proposé.

Nous proposons de valoriser notre Framework en introduisant un modèle original de contrôle appelé BYOC (Bring Your Own Control) qui formalise, selon différentes modalités, la capacité d'externaliser un service à la demande par la délégation d'une partie de son contrôle à un tiers externe. Un service externalisé à la demande est structurée en une partie client et une partie SP. Cette dernière expose à la partie client des API qui permettent de demander l'exécution des actions induites par les différentes étapes du cycle de vie. Nous illustrons notre approche par l'ouverture d'une API BYOC sécurisée basée sur XMPP. La nature asynchrone de ce protocole ainsi que ses fonctions de sécurité natives facilitent l'externalisation du contrôle dans un environnement SDN multi-tenant. Nous illustrons la faisabilité de notre approche par l'exemple du service IPS (système de prévention d'intrusion) décliné en BYOC.

Mots clefs : Réseau logiciel programmable, Interface nord, Interface de programmation applicative, Apporter votre propre contrôle, Externalisation / Délégation, Multi-client

Abstract

Over the past decades, Service Providers (SPs) have been crossed through several generations of technologies redefining networks and requiring new business models. The ongoing network transformation brings the opportunity for service innovation while reducing costs and mitigating the locking of suppliers. Digitalization and recent virtualization are changing the service management methods, traditional network services are shifting towards new on-demand network services. These ones allow customers to deploy and manage their services independently and optimally through a well-defined interface opened to the SP's platform. To offer this freedom to its customers, the SP must be able to rely on a dynamic and programmable network control platform. We argue in this thesis that this platform can be provided by Software-Defined Networking (SDN) technology.

We first characterize the perimeter of this class of new services. We identify the weakest management constraints that such services should meet and we integrate them in an abstract model structuring their lifecycle. This one involves two loosely coupled views, one specific to the customer and the other one to the SP. This double-sided service lifecycle is finally refined with a data model completing each of its steps.

The SDN architecture does not support all stages of the previous lifecycle. We extend it through an original Framework allowing the management of all the steps identified in the lifecycle. This Framework is organized around a service orchestrator and a resource orchestrator communicating via an internal interface. Its implementation requires an encapsulation of the SDN controller. The example of the MPLS VPN serves as a guideline to illustrate our approach. A PoC based on the OpenDaylight controller targeting the main parts of the Framework is proposed.

We propose to value our Framework by introducing a new and original control model called BYOC (Bring Your Own Control) which formalizes, according to various modalities, the capability of outsourcing an on-demand service by the delegation of part of its control to an external third party. An outsourced on-demand service is divided into a customer part and an SP one. The latter exposes to the former APIs which allow requesting the execution of the actions involved in the different steps of the lifecycle. We present an XMPP-based Northbound Interface (NBI) allowing opening up a secured BYOC-enabled API. The asynchronous nature of this protocol together with its integrated security functions, eases the outsourcing of control into a multi-tenant SDN framework. We illustrate the feasibility of our approach through a BYOC-based Intrusion Prevention System (IPS) service example.

Keywords: Sofware Defined Networking, Northbound Interface, API, Bring Your Own Control, Outsourcing, Multi-tenancy