IoRL deliverable D3.1
SDN/NFV environment in the home network
Definition, description and preliminary implementation of the IoRL home network

Editors:	Charilaos Zarakovitis - National Centre for Scientific Research Demokritos (NCSRD)
Deliverable nature:	Report (R)
Dissemination level: (Confidentiality)	Public (PU)
Contractual delivery date:	31 May 2018
Actual delivery date:	22 August 2018
Suggested readers:	network experts and engineers, telecommunication network operators, telecommunication equipment vendors, such as Nokia-Siemens, Samsung and Huawei
Version:	1.0
Total number of pages:	104
Keywords:	IoRL home network, NFV, orchestration, open stack, preliminary implementation, SDN, use case scenarios, VLC

Abstract

The internet of radio light (IoRL) home network is described together with its architectural design using software defined network (SDN), network function virtualisation (NFV) and virtual network function (VNF) virtualisation technologies jointly to support the identified use cases, implementation scenarios and fifth generation (5G) key performance indicators (KPIs). The preliminary testbed on which the IoRL home network will be implemented and tested is introduced and evaluated using practical scenarios and measures.

Disclaimer

This document contains material, which is the copyright of certain IoRL consortium parties, and may not be reproduced or copied without permission.

All IoRL consortium parties have agreed to full publication of this document.

The commercial use of any information contained in this document may require a license from the proprietor of that information.

Neither the IoRL consortium as a whole, nor a certain part of the IoRL consortium warrant that the information contained in this document is capable of use, nor that use of the information is free from risk, accepting no liability for loss or damage suffered by any person using this information.

The EC flag in this document is owned by the European Commission and the 5G PPP logo is owned by the 5G PPP initiative. The use of the EC flag and the 5G PPP logo reflects that IoRL receives funding from the European Commission, integrated in its 5G PPP initiative. Apart from this, the European Commission and the 5G PPP initiative have no responsibility for the content of this document.

The research leading to these results has received funding from the European Union Horizon 2020 Programme under grant agreement number 761992 — IoRL — H2020-ICT-2016-2018/H2020-ICT-2016-2.

Impressum

Internet of Radio Light

IoRL

WP3 Software Defined Home Network

Task 3.1 Establishment of the SDN/NFV environment in the Home Network

Definition, Description and Development of the IoRL Software Defined Home Network

Editor: Charilaos Zarakovitis - NCSRD

Work-package leader: Charilaos Zarakovitis - NCSRD

Estimation of PM spent on the Deliverable: 2.9

Copyright notice

2017 - 2018 Participants in IORL project

Executive summary

This document describes the infrastructural design of the IoRL home network with respect to recent advancements for building, managing and orchestrating virtual networks with 5G capabilities. It does so, by initially performing systematic survey analysis to identify contemporary findings on virtualisation technologies such as SDN and NFV, while classifying the key conjunction between these technologies to effectively approach the highly dynamic nature of the IoRL system and the interfaces for connecting and managing the remote radio heads to it. It then studies the components of the considered infrastructure such as physical nodes, compute node load, connections, hypervisor, number of virtual machines, etc. to design network function management mechanism with interconnections control, which can maintain consistency between the IoRL's abstractly defined topology and the actual user connections. The document concludes by developing a preliminary simulation testbed in an effort to capture the practical significance of the outcomes using experiments with measures considering the identified use cases and scenarios.

The results of this deliverable will guide the identification of the functional requirements and development of the system architecture, which will be presented in deliverables 3.2 and 3.3.

 © IoRL consortium 2018

List of authors

Company	Author	Contribution
NCSRD	Anastasios Kourtis Andreas Foteas Charilaos Koumaras Charilaos Zarakovitis Michail-Alexandros Kourtis Themistocles Anagnostopoulos	VNF deployment and configuration process in the IoRL home network, NFVI-PoP monitoring architecture and functional entities, NFVI monitoring metrics, Preliminary implementation of VLC and WLAN modules in the IoRL home Network, Network service lifecycle management, Tables/References editing
Eurescom	Adam Kapovits	Coordination
Brunel University	John Cosmas Mukhald Salih Nawar Jawad	VNF for security monitoring, VNF for intra-handover, OMNET emulator
Joada	Mathias Lacaud	VNF for multiple source streaming
Warsaw University of Technology	Krzystof Cabaj	VNF for security monitoring
Fraunhofer Institute for Integrated Circuits IIS	Rudolf Zetik	VNF for location sensing
Leicester University	Yue Zhang	Database for location sensing
Tsinghua University	Yong Li Jintao Wang	VNF for inter-handover

© IoRL consortium 2018

Table of contents

 © IoRL consortium 2018

List of figures and tables

List of tables:

 © IoRL consortium 2018

Abbreviations

5G	Fifth Generation
AL	Abstraction Layer
AP	Access Point
API	Application Programming Interface
ARIA	Agile Reference Implementation of Automation
ASIC	Application-Specific Integrated Circuit
B2B	Business-to-Business
B2C	Business-to-Customers
BSS	Business Support System
CA	Controller Adapter
CAPEX	Capital Expenditure
CC	Compute Controller
CDN	Content Delivery Network
CE	Control Elements
CHDC	Cloud Home Data Center
CLC	CLoud Controller
CLI	Command Line Interface
CLO	Cross-Layer interface Orchestration
COTS	Commercial Off-The-Shelf
CP	Cyclic Prefix
CPE	Customer Premises Equipment
CPRI	Common Public Radio Interface
D-NFV	Distributed Network Function Virtualization
DoS	Denial of Service
DPDK	Data Plane Development Kit
DPI	Deep Packet Inspection
EMS	Elemental Management Systems
EPC	Evolved Packet Core
ESCAPE	Extensible Service Chain Prototyping Environment

© IoRL consortium 2018

ETSI	European Telecommunications Standards Institute
FE	Forward Elements
FG	Forwarding Graph
ForCES	Forwarding and Control Element Separation
FSM	Finite State Machine
FW	FireWall
GNF	Glasgow Network Functions
GO	Global Orchestrator
GUI	Graphical User Interface
HeNodeB	Home eNodeB
HetNets	Heterogeneous Networks
HIPG	Home Internet Protocol Gateway
HTTP	HyperText Transfer Protocol
IaaS	Infrastructure-as-a-Service
ICT	Information and Communications Technology
IDPS	Intrusion Detection Prevention Systems
IDS	Intrusion Detection
IETF	Internet Engineering Task Force
IL	Infrastructure Layer
IMS	Internet Protocol Multimedia Subsystem
INBI	Intent-Based Northbound Interface
IoRL	Internet of Radio Light (project)
IoT	Internet of Things
IP	Internet Protocol
ISF	Integrated Security Framework
ISG	Industry Specification Group
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
KVM	Kernel-based Virtual Machine
LAN	Local Area Network

 © IoRL consortium 2018

LED	Light Emitting Diode
LFBs	Logical Functional Blocks
LoS	line-of-Site
LTE	Long-Term Evolution
LVAP	Light Virtual Access Point
MAC	Medium Access Control
MANO	MANagement and Orchestration
MDNS	Multi-Domain Network Service
MEC	Mobile Edge Computing
MIMO	Multiple Input Multiple Output
mmWave	Millimeter Wave
MNH	Multi-domain Network Supervisor
MSO	Multi-domain SDN Orchestrator
MTP	Mobile Transport and computing Platform
MVNO	Mobile Virtual Network Operators
NaaS	Network-as-a-Service
NC	Network Controller
NDB	Network Debugger
NE	Network Elements
NF	Network Function
NFV	Network Function Virtualization
NFVI	Network Function Virtualization infrastructure
NFVIaaS	Network Function Virtualization infrastructure-as-a-Service
NFVO	Network Function Virtualization Orchestrator
NIC	Network Intent Composition
NICE	Notes Install Cleanup Executable
NMS	Network Management System
NOS	Network Operating System
NR	New Radio
NRM	Node Resource Management

 © IoRL consortium 2018

NS	Network Service
NS-3	Network Simulator 3
NSD	Name Server Daemon
NSO	Network Service Orchestration
OAN	Open Access Network
OBSAI	Open Base Station Architecture Initiative
ODL	OpenDayLight
OFDMA	Orthogonal Frequency-Division Multiple Access
OFRewind	OpenFlow Rewind
OFTest	OpenFlow Test
OL	Orchestration Layer
ONF	Open Networking Foundation
ONOS	Open Network Operating System
OPEX	Operational Expenditure
OPNFV	Open Platform Network Function Virtualization
ORI	Open Radio Interface
OS	Operating System
OSGi	Open Services Gateway initiative
OSM	Open Source Management
OSS	Operations Support System
OTT	Over-The-Top
OVS	Open vSwitch
OVSDB	Open vSwitch Database
PaaS	Platform-as-a-Service
PNF	Physical Network Function
PoC	Proof of Concept
PoE	Power over Ethernet
POF	Plastic Optical Fiber
PoP	Point of Presence
QoE	Quality of Experience

 © IoRL consortium 2018

QoS	Quality of Service
RAN	Radio Access Network
REST	Representational State Transfer
RFC	Request for Comments
RO	Resource Orchestration
RRLH	Remote Radio-Light Heads
SDHN	Software Defined Home Network
SDI	Software Defined Infrastructure
SDK	Software Development Kit
SDN	Software Defined Network
SDWAN	Software Defined Wide Area Network
SFC	Service Function Chaining
SG	Service Graph
SL	Service Layer
SLA	Service Level Agreement
SLApp	Service Layer Application
SMTP	Simple Mail Transfer Protocol
SNR	Signal to Noise Rate
SO	Service Orchestrator
SON	Self-Organizing Network
SPAN	Switch Port Analyser
SR-IOV	Single-Root Input/Output Virtualization
STD	SDN Troubleshooting System
TAP	Terminal Access Point
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UI	User Interface
URI	Uniform Resource Identifier
URLLC	Ultra-Reliable and Low-Latency Communications
USB	Universal Bus System

 © IoRL consortium 2018

VAP	Virtual Access Point
VDU	Virtual Deployment Unit
vHG	Virtual Home Gateway
VIM	Virtualized Infrastructure Manager
VIM-MM	Virtualised Infrastructure Monitoring Manager
VLAN	Virtual Local Area Network
VLC	Visible Light Communication
VM	Virtual Machine
vMB	Virtual Middleboxes
VNF	Virtual Network Function
VNFaaS	Virtual Network Function-as-a-Service
VNFC	Virtual Network Function Components
VNFD	Virtual Network Function Descriptor
VNFFG	Virtual Network Function Forwarding Graph
VNFM	Virtual Network Function Manager
VNF-MA	Virtual Network Function Monitoring Agent
VR	Virtual Reality
VTN	Virtual Tenant Network
W3C	World Wide Web Consortium
WAN	Wide Area Network
WICM	WAN Infrastructure and Connectivity Manager
WiFi	Wireless Fidelity
WIN	WAN Infrastructure Management
WMN	Wireless MESH networks
XaaS	Everything-as-a-Service
xDPd	Extensible Data-Path daemon
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol
YANG	Yet Another Next Generation

 © IoRL consortium 2018

Definitions

IoRL home network

An IoRL home network is a wireless system setting for indoor environments composed by visible light communication (VLC), wireless fidelity (WiFi) and millimeter wave (mmW) remote radio-light heads (RRLH) access points sharing same virtual infrastructure.

Indoor environment

An indoor environment is a single building setting in which the VLC, WiFi and mmW access points are placed.

RRLH access point

A RRLH access point is a transceiver operating either using VLC, WiFi or mmW radio access technology.

Virtual infrastructure

A virtual infrastructure is a system consisted by virtual network components such as controllers, switches, monitoring tools, etc.

Intra-handover

Intra-handover refers to the handover process, which occurs between two coverage areas of single RRLH controller, and integrates by routing packets to the RRLH controller medium access control (MAC) addresses by the software defined network (SDN) controller.

Inter-handover

Inter-handover refers to the handover process, which occurs between the gNodeB (inside the IoRL home network) and the e/gnodeB (outside the IoRL home network), and integrates by switching the routing of packets from the Internet MAC address to the service gateway MAC address, and vice versa.

 © IoRL consortium 2018

1 Introduction

1.1 Objective of this document

The main objectives of this document are to:

- Identify the key research challenges of the synergy between SDN and NFV virtual network technologies towards drawing a clear view on how to effectively implement the IoRL home network in practice, taking into account outcomes of relevant 5G projects.
- Design the IoRL home network by selecting most suitable among contemporary virtualisation tools by means of their stability, efficiency and monitoring, and identify practical use cases with scenarios for being considered for demonstrations in the project, using the outcomes of task 2.1.
- Develop preliminary testbed to interpret SDN/NFV principles in the IoRL home network, while setting the basis towards extended use cases and implementation scenarios, which can potentially improve the outcomes of tasks 2.4-2.6.

1.2 Structure of this document

The rest of the document is organised as follows:

- Section 2 studies the interpretation of joint NFV and SDN technologies in the IoRL home network architecture based on systematic survey of recent network virtualisation advancements, which is important to identify the key challenges of the IoRL implementation,
- Section 3 uses the outcomes of Section 2 to develop contemporary SDN and NFV joint solutions for IoRL-specific virtual processes, functions, services modelling, and integrate the outcomes into the considered home network using first-step preliminary testbed with real-world experiments.

 © IoRL consortium 2018

2 Interpretation of SDN/NFV approach in the IoRL home network

The interpretation of joint SDN/NFV virtualisation in the IoRL home network can be perceived by answering the question: "*What are the key differences among the architectural designs of joint NFV/SDN solutions?*". This question seeks clarification on how ETSI, 5GPP bodies along with the relevant literature have proposed designing architectures of integrated joint NFV/SDN solutions. To answer the question above in this section we will initially describe a taxonomy approach that organises the various decision-making levels for the design of joint NFV/SDN architectures. Then, we will clarify the recent developments in the content of SDN/NFV technologies to highlight the orchestrators, controllers, switchers, processes, etc. that will be considered for the integration of the IoRL home network.

2.1.1 Generalised taxonomy approach of joint SDN and NFV structural elements

The proposed taxonomy approach will shed light on the structural architecture of SDN elements in NFV frameworks. The purpose is to produce useful insight for simplifying the role of SDN technology when applied on NFV solutions, such as IoRL. Our viewpoint is that NFV/SDN architectures design can be divided into two sides namely NFV-side and SDN-side.

The NFV-side is to decide whether or not to use the below features in the architectural framework:

- Distributed NFV (D-NFV), which describes how the management and orchestration (MANO) framework installs the virtual network functions (VNFs) such as data centres, forwarding devices, virtualised customers premises equipment (CPE);
- Multiple virtualised infrastructure managers (VIMs), which are to support the multi-domain administration and can be placed either in different NFV infrastructure points of presence (NFVI-PoPs) or in the same NFVI-PoP to increase scalability and performance;
- NFV MANO tools, which can provide complete solutions for MANO such as OpenMANO, OpeNodeBaton, etc. frameworks.

The SDN-side is to place the SDN elements in the NFV framework such as:

- SDN Resources, which comprise of both physical and virtual switches and routers such as 1) physical switch or router, 2) virtual switch or router, 3) e-switch (software-based SDN-enabled switch in a server network interface card (NIC)), 4) switch/router as VNF;
- SDN Controllers, which are responsible for controlling the SDN resources, determining the behavior of network traffic such as 1) merged with the virtualised infrastructure manager (VIM), 2) virtualised as a VNF, 3) as part of the NFVI (and not as a VNF), 4) as part of the operations support systems and business support systems (OSS/BSS), 5) as a physical network function (PNF);
- SDN applications, which are interfaces with one or multiple SDN controllers to enforce high-level network policy, such as firewall, network address translation, QoS and network management, and can be integrated 1) as part of a PNF, 2) as part of the VIM, 3) virtualised as a VNF, 4) as part of an Elemental Management System (EMS) for VNF monitoring.

 © IoRL consortium 2018

In this regard, Figure 2-1 illustrates our taxonomy approach of SDN/NFV elements interpretation in the IoRL home network, while rest sub-sections perform systematic survey to identify the contemporary advancements of each considered architectural element.

 © IoRL consortium 2018

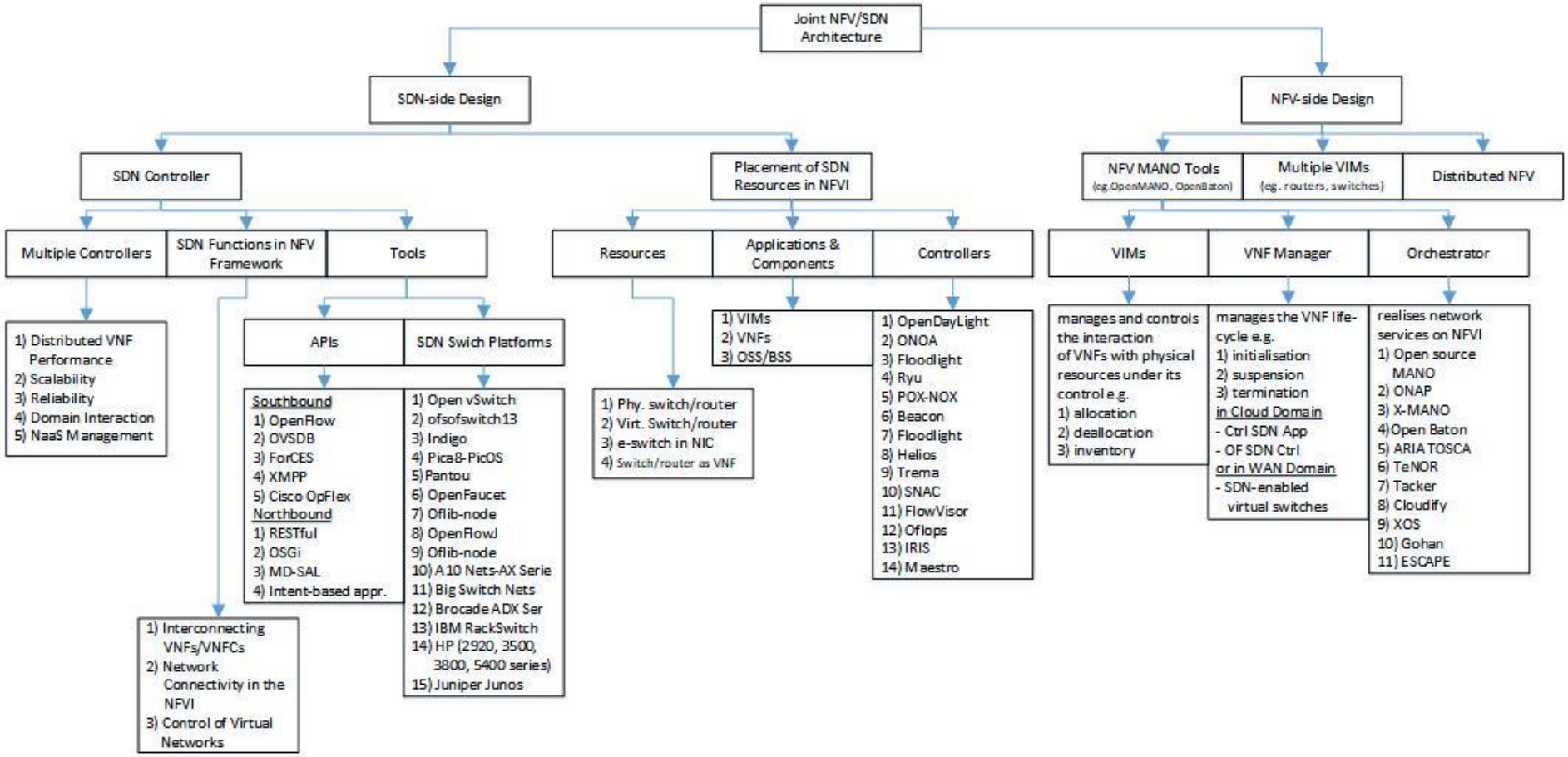


Figure 2-1 - Illustration of the proposed taxonomy approach for joint SDN and NFV structural interpretation

© IoRL consortium 2018

2.2 SDN architectural advancements

After presenting our taxonomy approach we study in the sub-section the state-of-the-art of network virtualisation including recent advancements on controllers, switchers, debugging tools and emulators to highlight the importance of SDN technology in the IoRL project.

2.2.1 Requirement for SDN research and development

The use of distributed protocols and coordination of changes in conventional networks remains incredibly complex involving the implementation of distributed protocols on the underlying network hardware to facilitate multiple services for traffic routing, switching, guaranteeing quality of service (QoS) applications, and providing authentication. Keeping track of the state of several network devices and updating policies becomes even more challenging when increasingly sophisticated policies are implemented through a constrained set of low-level configuration commands on commodity of IoRL's networking hardware. This frequently results in misconfigurations in view that changing traffic conditions requires repeated manual interventions to reconfigure the network. Therefore, the fundamental requirement of an overall framework is to fulfil a range of operational requirements such as ease of programmability, dynamic deployment, and provisioning, while facilitating innovative applications emerges. In the following, we highlight some of the technology and operational concerns that lead to development of current SDN technology.

- Automation: improving automation reduces operational expenditure and enables effective troubleshooting to improve unscheduled downtimes, ease of policy enforcement, provision of network resources/applications, etc.
- Dynamic resource management: adjusting the size of the network while updating its topology and resources dynamically can be further aided by SDN virtualization.
- Orchestration: orchestrating control of wide range of hundreds or even thousands network appliances such as in data centres or larger campus/building network environments.
- Multitenancy: SDN can improve proliferation of cloud-based services such that tenanted infrastructure can be separated from hosted services, e.g., tenants can be able to complete control over their addresses, topology, routing, and security.
- Open APIs: developers and operators can choose between numerous modular plugins to offer abstraction and define tasks by APIs with no concern about rest implementation details.
- Greater Programmability: SDN can potentially increase the ability to change device behavior and configuration in real time according to prevalent traffic conditions.
- Integrated security: SDN can lead to greater accuracy in detecting security incidents and simplifying management by integrating security devices within the network fabric.
- Effective resource management: in addition to security, multiple services such as load balancers and resource monitors, can be provisioned on demand and placed in the network fabric as and when required.
- Improved performance: SDN control frameworks can offer the ability to incorporate innovative traffic engineering solutions, capacity calculation, load balancing, and a higher level of utilisation that reduce ICT-related CO2 footprint.
- Real-time monitoring: SDN technology can improve real-time monitoring and connectivity of devices in the network.

 © IoRL consortium 2018

2.2.2 SDN architecture overview

The basic architecture of SDN utilises modularity-based abstractions, similar to formal software engineering methods. A typical SDN architecture divides processes such as configuration, resource allocation, traffic prioritisation, and traffic forwarding in the underlying hardware using a 3-tier structure consisting by the application, control, and data plane as highlighted below.

- Tier 1 - Data (forwarding) plane: Data plane is the set of network physical components (switches, routers, virtual networking equipment, firewalls, middle box appliances, etc.), almost same as in conventional networks. It aims to efficiently forwarding network traffic based on certain set of certain rules instructed by Tier 2 - Control plane. That way, SDN technology makes hardware (physical) infrastructure of the network rather flexible by removing intelligence and isolated configuration per network element, while moving these functionalities to the control plane.
- Tier 2 - Control plane: Control plane is the logic to decide on how traffic can be routed through the network from one node to another based on end user application requirements. Such control logic is incorporated into external software application elements called SDN controllers. An SDN controller handles all Tier-1 forwarding devices by translating individual application requirements and business objectives (e.g. traffic prioritising, access control, bandwidth management, QoS, etc.) into relevant programmable rules and announcing them to data plane. By introducing programmability through these rules, flow tables can be manipulated in individual elements and in real time based on network performance and service requirements.
- Tier 3 - Application plane: Application plane is the layer to include all applications and services of the network. Conceptually, the application plane situates above the control plane to enable applications communicating with data plane through requesting network functions from control plane, while performing network related tasks. Application plane uses APIs to capture the individual application parameters (delay, throughput, latency, etc.) based on which the SDN controller configures the individual network elements in the data plane for efficient traffic forwarding [1], [2].

Remark that the communication process between control-data, and control-application planes, takes place using southbound and northbound communication standards, respectively.

2.2.3 SDN communication standards

This Section presents the most prominent SDN communication standards, so-called application programming interface (APIs) and provides some examples of recent efforts to examine the evolution of these standards towards the 5G system.

2.2.3.1 Southbound communication standards

2.2.3.1.1 OpenFlow standard

The OpenFlow standard comprises three main elements, namely switches, controllers, and protocols (southbound API) as shown in Figure 2-2. The OpenFlow Switches are responsible for the data packet forwarding (i.e., data plane) according to the rules created and maintained by the SDN controller [3]. In OpenFlow, the SDN controller is the main component to support the network applications by determining the rules to be stored and

 © IoRL consortium 2018

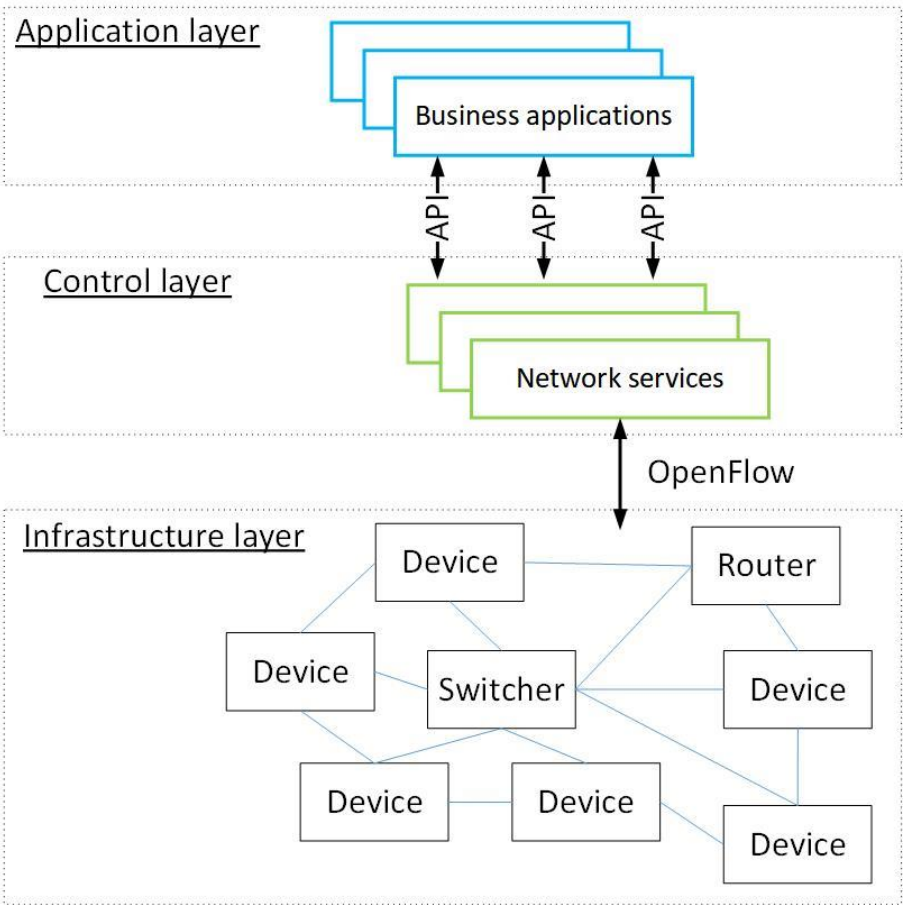


Figure 2-2 - Illustration of the elements of the OpenFlow SDN standard

applied by switches. There are several OpenFlow controllers available, such as OpenDayLight, Floodlight, ONOS, Ryu, POX, and NOX. An OpenFlow-based SDN controller can use two types of interfaces to create rules on switches, namely: the OpenFlow protocol and the Open vSwitch Database (OVSDB) management protocol.

- OpenFlow protocol: the OpenFlow protocol (ONF), maintained and updated by ONF, is the first and most prominent southbound communication API. This protocol uses a secure and encrypted channel (TCP/TLS) for performing the management of switches. It includes a set of messages for different situations: establishment and configuration of the management channel (e.g., Hello, Echo Request/Reply, Features Request/Reply, Set-Config), receiving (Packet-in) and redirecting (Packet-out) data packets, and OpenFlow rules management (Flow-mod). The OpenFlow protocol can coordinate all OpenFlow-enabled switches.
- OVSDB management protocol: Defined in RFC 7047 [4], this protocol uses a set of operations available in Open vSwitch (programmatic extension) to manage OpenFlow rules. Such operations allow insertion, updating, and deletion of forwarding rules directly into the Open vSwitch Database. As a consequence, the OVSDB Management Protocol is limited to use in virtual switches based on Open vSwitch.

As an example of using the aforementioned two OpenFlow-based protocols, the study in [5] deploys multi-tenant service function to chain edge network functions, using OpenStack. It does so, using the OVSDB management protocol via Neutron Open vSwitch Agent to provide

 © IoRL consortium 2018

connectivity to VNFs, and the OpenFlow protocol via POX controller to monitor the throughput of OpenFlow rules.

2.2.3.2.1 Forwarding and control element separation (ForCES) standard

ForCES is an SDN standard defined by the Internet Engineering Task Force (IETF) [6]. In ForCES, the plane separation occurs by dividing the Network Elements (NE) into two entities: Forwarding Elements (FE) and the Control Elements (CE). FEs represent the Data Plane and comprise both physical and virtual switches. ForCES models FEs by defining one or many Logical Functional Blocks (LFBs) classes, realised by XML-based modelling. An LFB comprises input and output ports and acts as a packet processing resource performing different functions, such as filtering, classification, and measurement. Multiple LFB instances in the same FE can be connected in a directed graph to create a network service. Each LFB provides operational parameters, capabilities, and events to a CE that acts as an SDN controller. CE uses the ForCES protocol as a southbound API to perform the per-LFB controlling.

As an example, the study in [7] proposes an NFV/SDN architecture using ForCES standard, where the NFV infrastructure (NFVI) includes LFB hypervisor to allow creating FE/LFBs as VNFs and CEs as EMS entities. Also, the NFVI provides LFBs acting as virtual switches to interconnect these VNFs. Furthermore, [8] proposes a ForCES approach to evaluate PoC architecture when applied for virtualisation of 4G Evolved Packet Core (EPC) components.

2.2.3.3.1 Extensible messaging and presence protocol (XMPP) standard

XMPP is a general communication standard offering messaging and information exchange among clients through centralised servers [9]. Similar to message transfer protocol (SMTP), each XMPP client can be identified by an ID which could be as simple as an email address. Client machines set up connections with a central server to advise their presence, which maintains contact addresses and may let other contacts know that a particular client is online. Clients communicate with each other through chat messages which are pushed as opposed to polling used in SMTP/POP emails. Every SDN object such as virtual machine, switch, and hypervisor can have an XMPP client module awaiting instructions from XMPP server for authentication and traffic forwarding, while each client updates its configuration as per server request. XMPP API is an IETF standardisation of the Jabber protocol and is defined for use with TCP connections in RFC 6121. A number of open source XMPP implementations are also available with variations being used in programs including Google, Skype, Facebook, and many games.

2.2.3.4.1 Cisco OpFlex standard

Cisco OpFlex standard [10], has gained momentum among southbound APIs because it facilitates control-data plane communication by implementing programming language across physical and virtual environments. In comparison with OpenFlow standard, which centralises the network control functions using SDN controller, Cisco OpFlex aims implementing and defining these policies with minimum controller usage. In particular, OpFlex aims enhancing policies to remove controller's scalability and directly manage channel communication from becoming the network bottleneck, pushing that way some level of intelligence to the devices. The standard allows policies to be defined within a logical, centralised repository in the SDN controller, such that OpFlex can communicate and enforce the respective policies within a subset of distributed policy elements on the switches. The standard allows bidirectional communication of policies, networking events, and statistical monitoring

 © IoRL consortium 2018

information. Real-time provision of information may in turn be used to make networking adjustments. The considered switches contain an OpFlex agent supporting the Cisco OpFlex standard. Some of the industry giants including Microsoft, IBM, F5, Citrix, and Red Hat have shown commitment to embedding OpFlex agent in their product line. OpFlex relies on traditional and distributed network control protocols to push commands to the embedded agents in switches. One of the main reasons for the early adaption of OpenFlow has been the level of control it can offer to developers for designing network control applications with minimal support from network vendors. To standardise OpFlex, Cisco submitted the protocol to IETF standardisation process and several vendors are presently working to standardise as well as increase the adoption of the standard.

2.2.3.2 Northbound communication standards

2.2.3.1.2 Representational State Transfer (RESTful) standard (1996)

RESTful follows the software architecture developed for World Wide Web consortium (W3C) encompassing all client-server communications. The standard was originally introduced by in [11]. Its main goal is to offer scalability, generality, and independence by allowing the inclusion of intermediate components between clients and servers to facilitate these necessary functionalities. Both clients and servers can be developed independently or in tandem by different vendors. In RESTful the server component is stateless, and clients keep track of their individual states to allow scalability. Server responses can be cached for a specified time. Every entity or global resource can be identified with global identifiers such as a URI and is able to respond to create, read, update, and delete (CRUD) operations. The uniform interface for each resource is GET (read), POST (Insert), PUT (write), and DELETE (remove). Data types can define network components such as controller, firewall rule, topology, configuration, switch, port, link, and hardware. RESTful is prevalent in most controller architectures as the northbound API of choice along with Java APIs. One of the major drawbacks of RESTful, however, is the lack of public subscription or live feed informing the application/service of network changes. Like HTTP, REST does not tell when a page has changed and requires frequent refresh. Application developers, therefore, periodically use loop calls to retrieve and subsequently post updates to individual switches based on pre-defined policies. RESTful support is included in almost all major SDN controllers including Ryu [12] and OpenDayLight [13] as well as several vendor proprietary platforms. According to [14], the REST API has become a prevalent choice for the NBI in SDN, because it is highly extensible and maintainable for managing services from both data and control planes. Procera [15] and Frenetic [16] would be alternatives for northbound APIs, but they run on top of a single OpenFlow Controller, and they are not as extensible as RESTful standard.

2.2.3.2.2 Open services gateway initiative (OSGi) standard (2000)

The OSGi standard includes a set of specifications for dynamic application composition using reusable Java components called bundles [17]. Bundles publish their services with OSGi services registry to find and use services of other bundles, which can be installed, started, stopped, updated, and uninstalled. Modules define how a bundle can import and export code. Security layer handles security and execution environment defines the methods and classes available in specific platform. A bundle can either get service or listen for a service to appear or disappear. Each service has properties to allow other services to select among multiple bundles that offer the same service. Services are dynamic, and a bundle can decide to withdraw its service which may cause other bundles to stop using the specific service.

 © IoRL consortium 2018

Bundles can be installed and uninstalled on the fly. Remark the OpenDayLight project [13], which is one major example of SDN controller platform built using the Java-based OSGi framework. OSGi allows the starting, stopping, loading, and unloading of Java based network (module) functionalities. In comparison, a Ryu controller [12] cannot offer OSGi support and the controller has to be stopped and restarted with the needed modules or a custom REST method is built with all required functionalities included to avoid controller restarts. A few other SDN platforms supporting OSGi include Beacon [18], Floodlight [19], and ONOS [20].

2.2.3.3.2 **Model-Driven Service Abstraction Layer (MD-SAL) standard (2014)**

The project in [21] designs Application containers, which can be built on top the OSGi framework to simplify the operational aspects of packaging and installing of applications in OpenDayLight controller. To further facilitate application development, the model-driven approach MD-SAL in [22] abstracts some services used in the OpenDayLight controller and offers to developers and network administrators the opportunity to unify the northbound and southbound APIs with the data structures of multiple services and individual components of the controller. The data structure itself is described by Yet Another Next Generation (YANG) language used to model the service and data abstractions as single system [23]. YANG can model semantics and data organisation including configuration and/or operation data as a tree. The controller northbound API utilises the self-describing data in XML, which simplifies the development of additional controller-driven functionalities as well as SDN applications. Modules providing specific network functionality can define a scheme to allow the interpretation of data structures through the MD-SAL in simple manner. MD-SAL uses APIs to connect and bind requests and services and provides an extra layer containing all the necessary logic to receive and delegate requests [24]. The SAL architecture itself comprises (i) top-level sub-system consisting of controller components or applications that use the controller SAL to communicate with other controller components and plugins (data store, validator, etc.) and (ii) nested subsystems that expose a set of functionalities and may have multiple instances attached to the overall system (network elements, virtual systems, etc.). The MD-SAL approach is relatively agnostic supporting any service model. Furthermore, the scheme stiches together horizontal modules and allows developers to use generic interfaces for service discovery and subsequent utilisation.

2.2.3.4.2 **Intent-Based Networking Approach (2015)**

Intent-based northbound interface (INBI) represents an evolution from static to dynamic network setup through simple application intent specification, programmed by the controller into physical and virtual devices. INBI allows SDN applications to describe their *intent* without necessarily having to specify the *methods* to achieve it. The SDN controller in turn translates these *intents* into low-level configuration commands in the data plane for subsequent execution. The application or user is therefore oblivious to the underlying infrastructure configuration providing added flexibility and automation for application developers and enabling agile deployment of services. With INBI's intents, a compelling case for implementation in SDN is ahead us with plenty of benefits such as application scalability, greater portability, increased coherence, contextual management. However, to permit practical deployment INBI requires a language for translating user or application intent [25], [26] hence, recent focus has drawn on developing intent-based INBI for proprietary controllers [27]. For instance, the aforementioned ONOS project [28] provides application intent at the INBI, where intents are described to allow applications to specify policies, which are compiled and installed as device flow rules. In terms of community efforts, the ONF INBI

 © IoRL consortium 2018

Working Group aims at specifying the information and architectural model for an intent-based interface to the SDN controller [29]. Within this direction, the OpenDayLight Network Intent Composition (NIC) initiative also aims assisting administrators in managing and directing network services by describing the intent for network behavior through a northbound interface by facilitating abstracted policy semantics instead of specifying data plane (flow) rules [30], [31]. This project uses existing OpenDayLight functions and southbound plugins and is designed to be protocol agnostic, that is, able to use any control protocol. Both OpenDayLight INBI Working Group and NIC initiative comprise a diverse set of projects, operators, and vendors collaborating with the open source community to bring intent-based approach to the SDN INBI.

2.2.4 SDN-based controllers

2.2.4.1 SDN controllers overview

A key abstraction of the SDN paradigm is the separation of the network control and forwarding planes. Conceptually, in SDN networks, resources are treated as a dynamic collection of arbitrarily connected forwarding devices with minimal intelligence. The control logic is implemented on top of a so-called SDN controller. Figure 2-3 illustrates the representative structure of a typical SDN controller. More precisely, the controller is a software platform to test and validate the switching/controlling behavior focusing on southbound protocol APIs. SDN controllers are seen as logically centralised entities responsible for set of tasks, including the extraction and maintenance of a global view of the network topology and state, as well as the instantiation of forwarding logic appropriate to a given application scenario. In practice the SDN controller manages connections to all substrate switches using a southbound protocol such as OpenFlow, and installs, modifies and deletes forwarding entries into the forwarding tables of the connected switches by using protocol specific control messages.

Fairly said, SDN controllers represent the focal point in the SDN system to oversee and manage the flow of traffic among southbound switches/routers (network wise) by using APIs according to each application requirements. In order an SDN controller to perform such tasks, it requires some information regarding the state of the underlying network provided by collections of pluggable modules, which perform different information gathering procedures about the data link layer devices, providing full inventory of the network below, as well as devices capabilities. In addition, when the SDN controller has full view of the network, it adds extensions and bundles to improve the controller capabilities and provide customised forwarding rules created using algorithms that analysed information and statistics gathered from the network.

The architectural structure of SDN controllers comprises of three coexisting layers:

- Southbound layer that communicates to the networking devices via different types of protocols such as OpenFlow, OVSDB, NETCONF, etc.;
- Northbound layer where the applications and services exist and developed making use of the abstraction of the network provided by the central layer and communicating to the controller via the northbound REST APIs;
- Central layer, which provides abstraction layer to the application developers allowing them to develop applications and services. Central layer contains the data stores (operational and configurational) where the required and current state of the system

 © IoRL consortium 2018

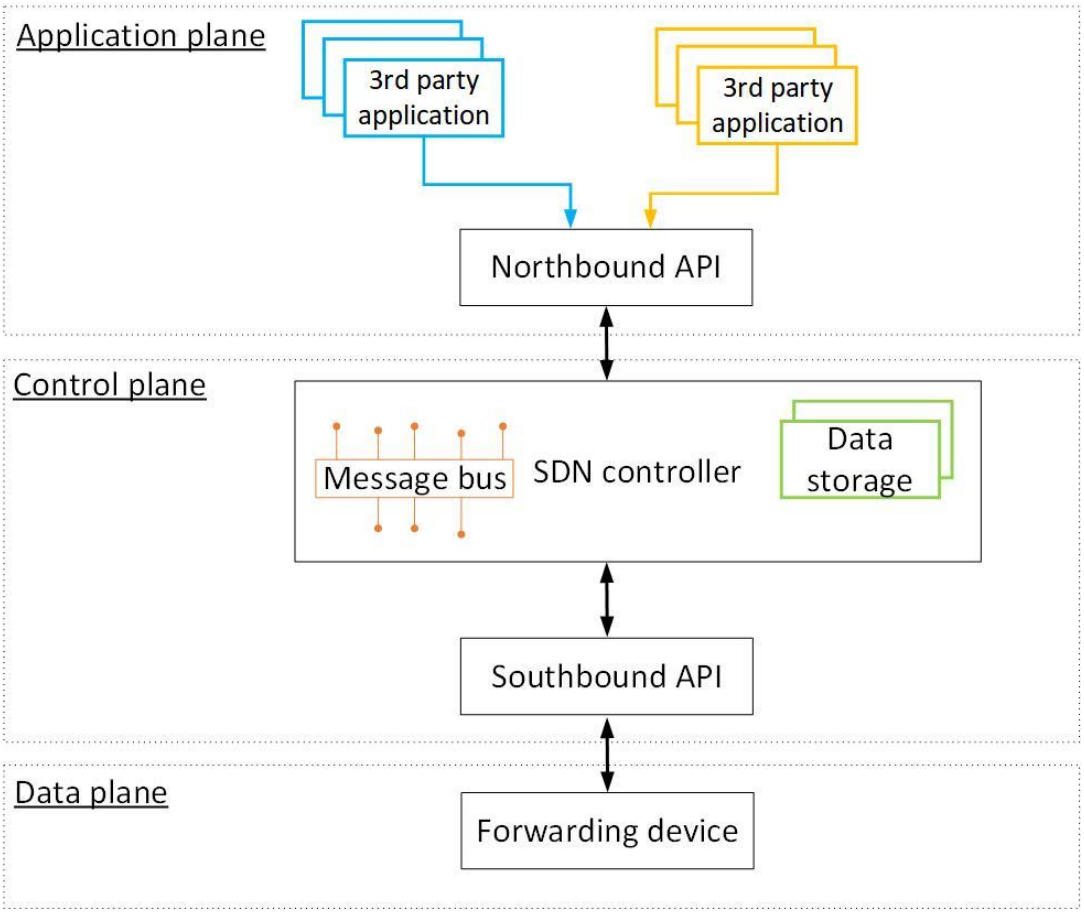


Figure 2-3 - Illustration of the structure of a typical SDN controller

stored, and also provide a message bus allows the northbound layer communicates to the southbound layer.

2.2.4.2 Types of SDN controllers

To design and utilise SDN controllers, there exist two popular approaches: (i) the vertical approach where multiple controllers are in effect managed by controller(s) at a higher layer and (ii) the horizontal approach, where controllers establish a peer-to-peer communication relationship [32], [33], [34]. In the following, we summarise most prominent of those SDN controllers.

2.2.4.1.2 The NOX and POX controller

Developed at Stanford University, NOX was the first open source framework to develop SDN controllers [35]. Due to its early availability and simplicity, NOX quickly became the de-facto reference design for OpenFlow controllers. As a result, it has been used to test new OpenFlow features, novel controller ideas and it has been employed extensively in research and feasibility studies. NOX applications - called modules - are implemented using the C programming language. NOX is event based; each module essentially consists of a collection of call-back functions, triggered by the arrival of specific OpenFlow protocol messages. The spin-off of NOX called POX [2] enables the use of Python for programming modules. POX is a Python controller, support Linux, MAC OS, and Windows operating platform, works with OF1.0 as southbound protocol, and support REST as northbound API, it can also function as

 © IoRL consortium 2018

an OpenFlow switch. While NOX/POX is extremely versatile it is not primarily aimed for production use, as it is not optimised for performance and stability and lacks resilience features. NOX is a C++ controller, support Linux operating platform, works with OF1.0 as southbound protocol, and support REST as northbound API, NOX and POX suffer from poor documentation. Also, since both NOX and POX are written in C++/Python, they support OpenFlow version 1.0 only and therefore, both projects are currently discontinued. Most recent studies using NOX and/or POX can be found in [5], [36], [37].

2.2.4.2.2 **FloodLight controller**

FloodLight is a Java-based OpenFlow cross-platform controller, support Linux, MAC Floodlight is a SDN controller written in Java and supports the OpenFlow v.1.0-v.1.4 [38] and 1.3 southbound protocol, and support REST as northbound API, it has better documentation than Beacon and RYU controllers. It is Apache-licensed and supported by engineers and developers from Big Switch Networks. In addition to OpenFlow controller, Floodlight provides a set of internal SDN applications (e.g., firewall and load balancing) and a REST API for development of external applications. *Recent SDN developments with Floodlight can be found in* [39], [40], [41], [42], [43], [44].

2.2.4.3.2 **Ryu controller**

Ryu is an open source framework (Apache 2.0 licensed) written in Python, created by NTT and supports OpenFlow southbound API v.1.0-v.1.5 [12]. Also, a REST API is available to be used for external SDN applications. Currently, Ryu is fully integrated into Neutron (OpenStack Networking Service). *Recent SDN developments with Ryu can be found in* [45], [46], [47], [48], [49], [50].

2.2.4.4.2 **Open network operating system (ONOS) controller**

ONOS comprises an open source SDN controller for SDN/NFV solutions [28]. Like ODL, ONOS was developed using Java on top of the Apache Karaf OSGi container and provides the following main features: i) a GUI for the view the network state, support different SDN southbound APIs, such as OpenFlow, NETCONF, and OpenConfig, ii) northbound abstractions to simplify creating intent-based virtualised networks, and iii) high availability and scalability support (e.g., cluster of ONOS instances). *Recent SDN developments with ONOS can be found in* [51], [52].

2.2.4.5.2 **OpenDayLight controller**

OpenDayLight is the newest and content-widest SDN controller platform. It is backed by the Linux Foundation and developed by an industrial consortium, which includes Cisco, Juniper and IBM, among many others. OpenDayLight provide enhancement and support to cloud and NFV, as well as greater integration with larger industry frameworks from OPNFV and OpenStack. OpenDayLight is a Java-based OpenFlow cross-platform controller, support Linux, MAC OS, and Windows operating platform, it works with OF 1.0, 1.3, 1.4, NETCONF, OVSDB, and many other southbound protocols, as well as supporting REST as northbound API, it is partnered with Linux foundation and over forty companies such as Cisco, IBM, and NEC, and it has the best documentation amongst all the other controllers. OpenDayLight includes numerous functional modules which are interconnected by a common service abstraction layer. ODL is a modular SDN open source platform maintained by Linux Foundation21 [53]. It is written in Java and aims to accelerate the development of SDN in conjunction with NFV solutions in production environments. ODL offers plugins that support

 © IoRL consortium 2018

different SDN southbound APIs, such as OpenFlow (1.0 and 1.3) and OVSDB. Currently, the newest version of the ODL is the Oxygen, released in March 2018. *Recent SDN developments with ODL can be found in* [36], [40], [54], [55], [56], [57], [58], [59], [60], [61], [62], [63], [64]. Further, OpenDayLight provides a flexible northbound interface using Representation State Transfer APIs (REST APIs) and includes support for the OpenStack cloud platform. Current OpenDayLight release is built upon the following four structural considerations:

- technology-specific plug-ins, for managing SDN and non-SDN devices with various network configuration protocols;
- a service abstraction layer to unify the capabilities of the underlying technology-specific plug-ins;
- a core of basic network services such as topology management, host tracking etc.;
- a set of northbound APIs (REST-based) for communicating with network management applications.

2.2.4.6.2 **Uncommon SDN-based controllers**

Besides the aforementioned well-known controller frameworks, some other less used yet known tools to design SDN controllers are:

- Beacon (Stanford University) [18]
- Floodlight (Big Switch) [65]
- Helios (NEC) [66]
- Trema (NEC) [67]
- SNAC (Nicira) [68]
- FlowVisor (Stanford University/Nicria) [69]
- Oflops (Cambridge, Berlin, Big Switch) [70]
- IRIS (IRIS Team-ETRI) [71]
- Maestro (Rice University) [72]

2.2.5 **SDN-based switches**

2.2.5.1 **SDN switches overview**

An OpenFlow switch is a software program or hardware device that forwards packets in a SDN environment. When it receives a packet, which does not have a flow for (match + exit port) the SDN switcher contacts the SDN controller (server) to ask for information of how to handle this specific packet. The controller can then download a flow to the switch, possibly including some packet manipulation as shown in Figure 2-4. Once the flow is downloaded to the switch it will switch similar packets at wire-speed. Having a central server that is aware about network's layout and can make all the switching decisions, while building the routing paths gives to SDN system the following advantages:

- the SDN controller could route non-critical/bulk traffic on longer routes that are not fully utilised;
- the SDN controller can send the initial couple of packets to a firewall, and once the firewall is happy/accepts the flow, the SDN controller can bypass the firewall thus removing load from the FW and allowing multi-gigabit data centers to be fire-walled;

 © IoRL consortium 2018

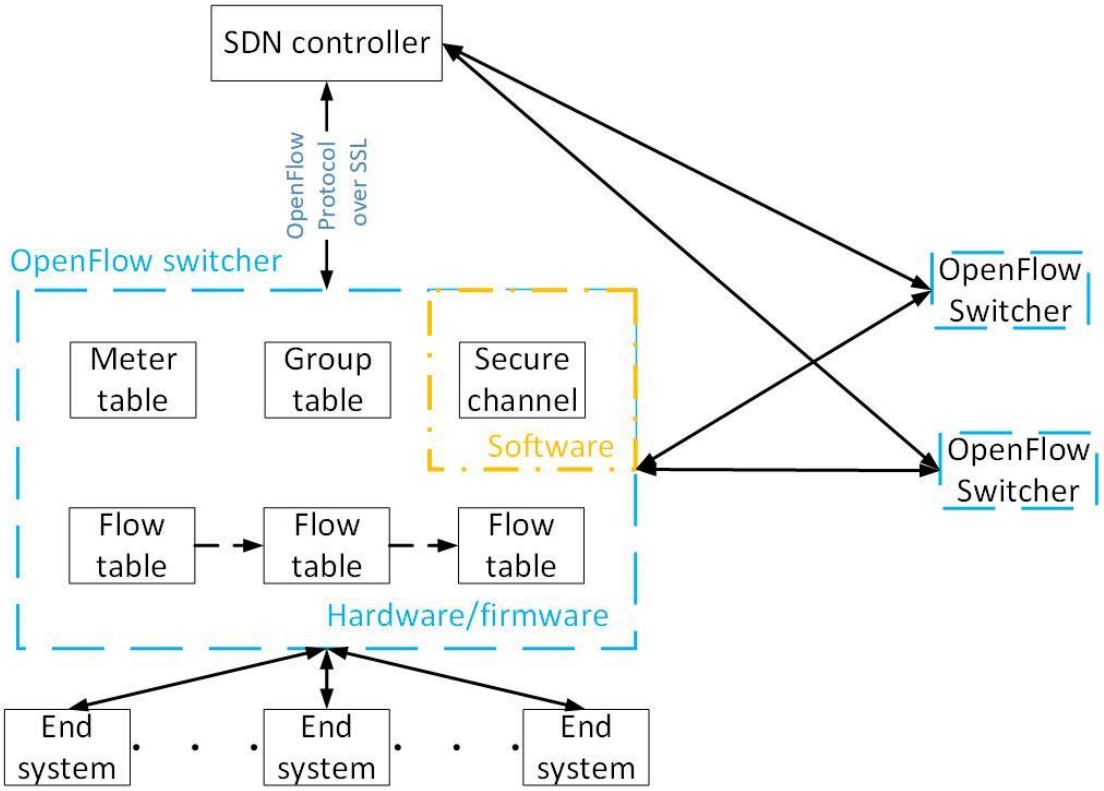


Figure 2-4 - Illustration of the structure of a typical virtual network including SDN controllers and switches

- the SDN controller can easily implement load-balancing also at high data rates by just directing different flows to different hosts, only doing the set-up of the initial flows;
- network traffic can be isolated without the need for VLANs, the SDN controller can just refuse certain connections;
- makes easier to set up a network terminal access point (TAP) for any port or specific traffic by programming the network and by sending duplicate streams to network monitoring devices;
- SDN switching allows for the development of new services and ideas all in software on the SDN controller.

2.2.5.2 Types of SDN switches

In the following, we summarise most prominent and widely-used SDN switches.

2.2.5.1.2 Open vSwitch switcher

Open vSwitch is one of the most widely deployed software switches. It employs an OpenFlow stack that both can be used as a virtual switch in virtualised network topologies and has also been ported to multiple hardware/commodity switch platforms [73]. The Open vSwitch is a part of the Linux kernel since version 3.3 [74].

2.2.5.2.2 ofsofswitch13 switcher

ofsofswitch13 operates in the user space and provides support for multiple OpenFlow versions [75]. The soft switch supports a management utility to directly control the

 © IoRL consortium 2018

OpenFlow switch, namely Data Path Control (Dpctl), which enables the addition and deletion of flows, query switch statistics, and modification of flow table configurations. Although ofsofswitch13 supports a variety of OpenFlow features, it has recently run into some compatibility issues with latest versions of Linux (i.e. Ubuntu 14.0 and beyond).

2.2.5.3.2 **Indigo switcher**

Indigo is an open source project to implement a range of physical switches by utilising hardware features of existing Ethernet application-specific integrated circuit (ASIC) switches, which operate using OpenFlow v.1.0 pipeline at line rates [76].

2.2.5.4.2 **Pica8 PicOS switcher**

PicOS by Pica8 is a network operating system to design flexible and programmable networks using white box switches with OpenFlow [77]. Its main feature is that it allows integrating OpenFlow rules in legacy layer 2 and layer 3 networks to create a new network from scratch without disrupting the existing network structure.

2.2.5.5.2 **Pantou switcher**

Pantou modifies commercial wireless routers and access points to OpenFlow enabled switches. It implements the OpenFlow standard on top of OpenWRT platform [78]. Pantou's OpenWRT platform relies on the BackFire release (Linux v2.6.32), while the corresponding OpenFlow module is based on the Stanford reference implementation in user space.

2.2.5.6.2 **Oflib-node switcher**

This is a software switcher designing tool based on OpenFlow protocol library for Node.js to offer design flexibility by converting input data between the protocol messages and JavaScript objects [79].

Some other less used yet known platforms to design SDN switches are:

- OpenFlowJ [80];
- OpenFaucet [81];
- A10 Networks - AX Series [82];
- Big Switch Networks - Big Virtual Switch [83];
- Brocade ADX Series [84];
- IBM RackSwitch G8264 [85];
- HP (2920, 3500, 3800, 5400 series) [86];
- Juniper Junos (MX, EX, QFX Series) [87].

2.2.6 **SDN-based network-as-a-service (NaaS) platforms**

In the SDN context, the NaaS approach is to design services for network transport connectivity able to optimise the allocation of the available network and computing resources jointly. In the following, we highlight the most popular platforms to apply NaaS in SDN.

2.2.6.1 **OpenNaaS platform**

The NaaS model has been instantiated in the OpenNaaS easy prototyping and proof casing of its concepts. OpenNaaS [88] is an open-source framework, which provides tools for managing the different resources present in any network infrastructure. The software

 © IoRL consortium 2018

platform was created in order to offer a neutral tool to the different stakeholders comprising an Open Access Network (OAN). It allows them to contribute and benefit from a common NaaS software-oriented stack for both applications and services. It is based on a lightweight, abstracted, operational model, which is decoupled from actual vendors' specific details, and is flexible enough to accommodate different designs and orientations. In fact, the OpenNaaS framework provides tools to implement the logic of an SDN-like control and management plane on top of the lightweight abstracted model. Some deployment examples using OpenNaaS can be found in the following list of European projects extending the OpenNaaS framework: OFERTIE [89], CONTENT and SODALES [90]. Furthermore, authors in [91] used OpenNaaS in order to build a first proof-of-concept pilot for the VNF creation and management.

2.2.6.2 **OpenStack neuron platform**

OpenStack Neutron [92], historically known as Quantum, is an OpenStack project focused on delivering Networking as a Service (NaaS). Neutron provides a way for organisations to make it easier to deliver networking as a service in the cloud and provides REST APIs to manage network connections for the resources managed by other OpenStack services. Neutron provides native multi-tenancy support (isolation, abstraction and full control over virtual networks), letting tenants create multiple private networks and control the IP addressing on them, and exposes vendor-specific network virtualisation and SDN technologies. Also, Neutron includes a growing list of plugins that enable interoperability with various commercial and open source network technologies, including routers, switches, virtual switches and SDN controllers. Starting with the Folsom release, Neutron is a core and supported part of the OpenStack platform. However, it is a standalone and autonomous service that can evolve independently to OpenStack.

2.2.6.3 **OpenDayLight virtual tenant network (VTN) platform**

OpenDayLight VTN [93] provides multi-tenant virtual network on an SDN controller. VTN provides an abstraction that enables the complete separation of the logical plane from physical plane of the network. This allows users to design and deploy virtual networks for their customers without needing to know the physical network topology or underlying operating characteristics. The VTN also allows the network designer to construct the virtual networks using common L2/L3 network semantics. Moreover, VTN allows the users to define the network with a look and feel of conventional L2/L3 network. Once the network is designed on VTN, it is automatically mapped onto the underlying physical network, and then configured on the individual switches leveraging an SDN control protocol. The definition of the logical plane makes it possible not only to hide the complexity of the underlying network but also to better manage network resources. It achieves a reduction in the reconfiguration time of network services and minimising network configuration errors.

2.2.6.4 **FlowVisor platform**

FlowVisor, [94] originally developed at Stanford University, has been widely used in experimental research and education networks to support slicing where multiple experimenters get their own isolated slice of the infrastructure and control it using their own network OS and a set of control and management applications. FlowVisor has been deployed on a Stanford production network and sponsors, such as GENI, Internet2, NEC and Ericsson, have been contributing to it and using it in their research labs. The SDN research community considers FlowVisor an experimental technology, although Stanford University has run

 © IoRL consortium 2018

FlowVisor in its production network since 2009. FlowVisor lacks some of the basic network management interfaces that would make it enterprise-grade. For example, it currently does not any CLI or Web-based administration console but requires users to make changes to the technology with configuration file updates.

In technical terms, FlowVisor consists the ON.LAB network slicer to allow multiple tenants sharing the same physical infrastructure. In particular, a tenant can be either a customer requiring his own isolated network slice; a sub-organisation that needs its own slice; or an experimenter who wants to control and manage some specific traffic from a subset of endpoints. FlowVisor acts as a transparent proxy between OpenFlow switches and various guest network operating systems. It supports network slicing and allows a tenant or an experimenter to control and manage some specific traffic from a subset of end points. This approach enables multiple experimenters to use a physical OpenFlow network without interfering with each other.

2.2.6.5 **OpenVirtex platform**

OpenVirteX [95] is a network hypervisor to create multiple virtual and programmable networks on top of a single physical infrastructure. Each tenant can use the full addressing space, specify their own topology, and deploy the network OS of their choice. This NaaS platform is actually a network hypervisor that enables operators to provide networks whose topologies, management schemes, and use cases are under the full control of their tenants. More specifically OpenVirteX builds on OpenFlow as protocol and FlowVisor for design. In this respect they share some common properties i.e. act as proxies between tenants and the underlying physical infrastructure. Unlike FlowVisor however, OpenVirteX provides each tenant with a fully virtualised network featuring a tenant-specified topology and a full header space.

2.2.6.6 **OpenContrail platform**

A popular NaaS platform that is attracting attention is led by Juniper Networks, named OpenContrail [96]. It is a modular project that provides an environment for network virtualisation and published northbound APIs. In particular, the network virtualisation is provided by means of a set of building blocks and high-level policies; it integrates an SDN controller to support network programmability and automation, and a well-defined data model to describe the desired state of the network; an analytics engine is designed for very large-scale ingestion and querying of structured and unstructured data. It also provides an extensive REST API to configure and gather operational and analytics data from the system.

2.2.7 **SDN emulators for network simulation**

Besides SDN controllers, switchers and NaaS platforms, it is important to study SDN emulation approaches, which provide network topology references for network event simulation and monitoring. Next, we present some widely-used SDN emulators.

2.2.7.1 **Mininet emulator**

Mininet emulator [97] allows OpenFlow-based networks to be emulated over either single or a cluster of machines. The distribution of Mininet nodes and links over clusters of machines can utilise the resources at each machine, adding scalability to emulate larger networks, where more computation and communication bandwidth is required than in the single server case. Also, Mininet simplifies the development and deployment of new services by

 © IoRL consortium 2018

providing a flexible software platform to create virtual machines, hosts, and network switches connected to an in-built (OVS-reference) or user defend controller for testing purposes. The latest Mininet v2.2.2 supports OpenFlow versions up to 1.3 (along with Open vSwitch v2.3) by default and can also be customised to use external user space switch such as the sofswitch13 [75].

2.2.7.2 **NS-3 emulator**

The widely-used NS-3 network emulator can test and develop networking protocols and services to support OpenFlow switches. Although the design simplicity featured, the main drawback in NS-3 is that it is limited to early versions of OpenFlow (v.0.89 and earlier) [98] thus, outdated compared to Mininet. While official work to update versions of OpenFlow in NS-3 continues, in the meantime a specialised OpenFlow v.1.3 module for NS-3, namely OFSwitch13, has been externally designed by NEC [99]. OFSwitch13 module relies on the ofsofswitch13 library and provides switch implementations by converting messages (data) from wire protocol (point-to-point) format, which makes NS-3 network emulation compatible with up to v.1.3 version of OpenFlow.

2.2.7.3 **OMNET emulator**

OMNET is an object-oriented modular discrete event network simulation framework that can be used to model wired and wireless networks, network protocols, and many other things [100]. OMNET itself is not a network simulator, rather it is a framework that provides infrastructure of components and tools called modules, which can be combined together to form the simulated network. These OMNET modules have gates, which act as interfaces. The gates are connected by predefined connection links. The modules communicate with each other by sending and receiving messages through those modules' gates. At the top of this ease-of-use hierarchy is the network, which has no gates to the outside world. In OMNET, the models structure (gates, connections, etc.) is described in files written in NED (Network description) language. The implementation of the modules behavior is written in C++, while the parameters value that customize the module behavior and define the model topology can be assigned in either the NED or “.ini” files.

 © IoRL consortium 2018

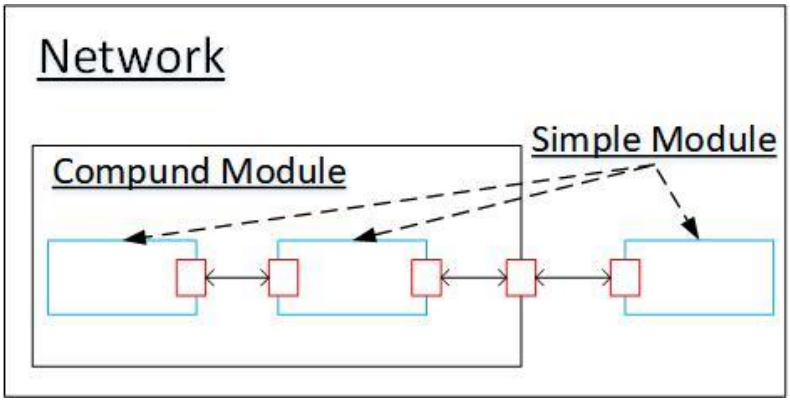


Figure 2-5 - Illustration of the basic building blocks of OMNET emulator

The active modules are written in C++, by using the class library of the OMNET, and they are called simple modules (which sit at the lowest level of the module hierarchy). Simple modules can be grouped to form compound modules. Simple modules in different compound modules can connect with each other only through the compound module's gates (the connections cannot sidestep module hierarchy) as shown in Figure 2-5. Overall, the OMNET simulation model consists of simple modules communicating via messages. The creation, deletion, modification, storage, transmission, and reception of messages is the main job of the simple modules, hence the whole OMNET model is there to accomplish this job. To create or destroy a message object you need to use the C++ New or Delete operators. The message object is an instance of a class called cMessage, or one of its subclasses. Practically, fields should be added to the cMessage to customize it upon your simulation requirements, by creating subclasses to extend the cMessage class. The same goes for the network packets, since it is an instance of the cPacket, which is sub-classed from the cMessage.

INET framework represents the most known community-based simulation package written for OMNET. The framework contains IPv4, IPv6, TCP, SCTP, UDP protocol implementations, MPLS model with RSVP-TE and LDP signaling, and a detailed physical layer model, application models and more. simuLTE and Openflow 1.3 are another simulation packages that extends INET to provide simulation platform SDN oriented mobile network.

2.2.8 SDN debugging tools

These are specialised tools set to debug SDN behavior at the switcher and controller level. Some robust approaches are the followings.

2.2.8.1 SDN troubleshooting system tool

SDN troubleshooting system (STS) simulates the network devices by allowing enough flexibility to generate and examine various test case deployments via programming [101]. With STS users can interactively visualise the network states and the real-time changes, to automatically determine the events that trigger deviant behavior and identify potential bugs. The implementation of STS is based on the POX controller platform, with feasibility to use other OpenFlow v.1.0 compliant controllers such as Ryu and Floodlight.

 © IoRL consortium 2018

2.2.8.2 **Open vSwitch specific tool**

This debugging tool relies on vSwitch OVSDB library [73]. It includes the (i) OVS-vsctl utility for configuring the switch (daemon) configuration database (known as OVS-db), (ii) the OVS-ofctl command line tool for monitoring and administering OpenFlow switches, (iii) the OVS-dpctl utility to administer Open vSwitch data paths (switches) and enable visibility and control over a single switch's flow table, and (iv) the OVS-appctl utility for querying and controlling Open vSwitch daemons.

2.2.8.3 **NICE tool**

NICE offers an automated testing tool to identify and check bugs in OpenFlow programs [102]. It can apply model checking to explore the entire state of SDN controller, switcher(s), and host(s). To address scalability issues during model checking, NICE uses symbolic execution of event handlers, which can effectively identify the packets that exercise code paths on the controller. NICE prototype tests Python applications using the NOX platform.

2.2.8.4 **OFTest tool**

OFTest is an OpenFlow-based debugging tool that includes a collection of test cases for SDN switchers [103]. To host the switch under test, OFTest uses unit test module of the standard Python distribution. Both control plane and data plane side of switch connections can be tested by sending and receiving packets to the switch as well as polling switch counters.

2.2.8.5 **Anteater tool**

Anteater attempts to check network invariants that exist in network devices, such as connectivity and consistency [104]. Its main benefit is that it is agnostic to protocols and identifies errors due to faulty firmware and/or control channel communication.

2.2.8.6 **VeriFlow tool**

VeriFlow allows real-time verification and resides between the controller and the data plane elements (switches) [105]. It is mainly used to perform pruning of flow rules that may result in anomalous network behavior.

2.2.8.7 **OFRewind tool**

OFRewind is another tool to perform debugging of network events in both the control and data plane [106]. Its main functionality is to localise troubleshooting efforts by building a log of network events based on their impact on system's operation, in order this log to be used in case of problematic network operation.

2.2.8.8 **Network debugger tool**

Network debugger (NDB) tool implements traffic breakpoints and packet-back traces for an SDN environment [107]. In NDB, users can isolate networking events that may have led to an error during traffic forwarding. The tool works using the OpenFlow API to configure switches and generate debugging events. NDB then acts as a proxy intercepting OpenFlow messages between switches and the controller. The debugger relies on OpenFaucet Python module implementing OpenFlow v1.0.

2.2.8.9 **Wireshark tool**

The popular network analyser Wireshark can be deployed on the controller or Mininet host to view OpenFlow exchange messages between the controller and individual switches [108].

 © IoRL consortium 2018

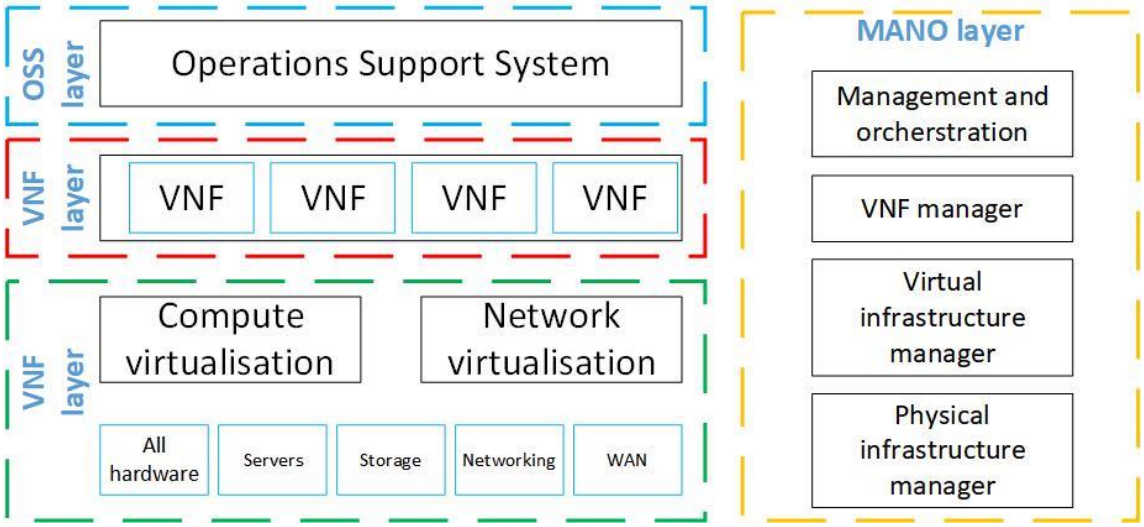


Figure 2-6 - Illustration of the generalised viewpoint of NFV infrastructure

OpenFlow control packets can be directly faltered, while capturing using the TCP control channel traffic ports (i.e. ports 6633 and 6653). The captured data packets provide a useful learning tool to understand switcher's and controller's behavior.

2.3 NFV Architectural Advancements

This Section provides an in-depth study on research advancements of virtual applications and functions to capture the evolution of NFV technology and correlate it with SDN in the context of the IoRL home network.

2.3.1 NFV background and architecture

NFV technology was initiated in 2012 by the European Telecommunications Standards Institute (ETSI) NFV Industry Specification Group (NFV ISG) to allow customers to transfer the networking functions from vendor-specific and proprietary hardware appliances to software hosted on commercial off-the-shelf (COTS) platforms [109]. The main idea is to provide the network services in virtual machines (VMs) working in Cloud infrastructures, where each VM can perform different network operations (e.g., firewall, intrusion detection, deep packet Inspection, load balancing, etc.) [110]. The main benefits of deploying network services as virtual functions are:

- Flexibility in the allocation of network functions in general-purpose hardware;
- Rapid implementation and deployment of new network services;
- Support of multiple versions of service and multi-tenancy scenarios;
- Reduction in capital expenditure (CAPEX) costs by managing energy usage efficiently;
- Automation of the operational processes, thus improving efficiency and reducing operational expenditure (OPEX) costs.

The NFV infrastructure depicted in Figure 2-6 comprises of four main functional elements [111].

- Virtual network function (VNF) layer virtualises a certain network function, which operates independently by others. A particular VNF can run on one or more VMs and it can be divided into several sub-functions called VNF Components (VNFCs). VNFCs

 © IoRL consortium 2018

monitoring is performed using Elemental Management Systems (EMSs). Automation of the operational processes, thus improving efficiency and reducing operational expenditure (OPEX) costs.

- The NFV infrastructure (NFVI) comprises of all hardware and software required to deploy, operate, and monitor VNFs. Particularly, NFVI includes a virtualisation layer necessary for abstracting the hardware resources (processing, storage, and network connectivity) to ensure independence of the VNF software from the physical resources. The virtualisation layer is usually composed of virtual server (e.g. Xen [112], Linux-KVM [113], Dell-VMware [114], etc.) and network (e.g., VXLANs [115], NVGRE [116], OpenFlow [3], etc.) hypervisors. The NFVI point of presence (NFVI-PoP) defines a location for network function deployments as one or many VNFs.
- NFV management and orchestration (MANO) comprises three components:
    i) The virtualised infrastructure manager (VIM), which manages and controls the interaction of VNFs with physical resources under its control (e.g. resource allocation, deallocation, and inventory);
    ii) The VNF Manager (VNFM), which is responsible for managing the VNF life-cycle (e.g., link initialisation, suspension, and termination)
    iii) The NFV Orchestrator (NFVO), which is responsible for realising network services on NFVI.

  Also, NFVO performs monitoring operations of the NFVI to collect information for operations and performance management.
- Operations support systems and business support systems (OSS/BSS) element comprises the legacy management systems and assists MANO in the execution of network policies. The two systems (OSS and BSS) can operate together by telecommunications service providers, either automatically or manually to support a range of telecommunication services.

2.3.2 **Research advancements on NFV applications**

This section examines the environments where NFV technology can been applied with the intension to identify the open issues NFV promises to resolve.

2.3.2.1 **NFV towards computer and network resources unification**

NFV aims to unify wireless network resources by creating an abstraction layer (AL) on both the computational and network resources in a way to provide unique and centralised view of the whole environment. Implemented together with SDN, the joint NFV/SDN solution focuses on creating the AL to deliver intelligent network services - regarding performance and reliability - for different customer profiles such as end, retail, and enterprise users, as well as over-the-top (OTT) providers, and developers [117]. AL can integrate two orchestration functions, namely resource orchestration (RO) and network service orchestration (NSO) [118]. RO utilises NFV VIM component with SDN to perform a global resource management with resource virtualisation provisioning and management. The NFVO uses NSO functions to implement the life-cycle management of network functions for VNF forwarding graphs (VNFFGs), which can coordinate groups of VNF instances as network services. These VNFFG functions use the services exposed by the VNF and RO allowing joint instantiation and configuration along with connectivity and dynamic resource management.

 © IoRL consortium 2018

In this context, different computer and network resources can achieve unification, as done in the EU-FP7 projects T-NOVA [119], UNIFY [120], SONATA [121], Project VITAL [122], 5GEx [123] and 5G-Transformer [124].

2.3.2.1.1 **Project T-NOVA**

Project T-NOVA (2014-2016) aims at implementing an SDN-based MANO framework to manage a federated network and cloud resources. Its primary objective is to deliver third-party network functions (NFs) to operator's customers in an automated and optimised manner, introducing a "network function store". For NF management, an orchestrator platform was developed on top of common open source components such as OpenStack and OpenDayLight [13]. Besides, the WICM (WAN Infrastructure and Connectivity Manager) provides network connectivity between NFVI-PoPs (Points of Presence) and manages traffic steering in virtual networks.

2.3.2.2.1 **Project UNIFY**

Project UNIFY (2013-2016) seeks to unite computer and network resources in a common management framework. The UNIFY NFV/SDN architecture aims at creating and managing the dynamic end-to-end network services from the home and enterprise networks to the operator's data centre. It provides a MANO framework that integrates both Cloud and WAN domains and includes three layers, namely the service layer (SL), the orchestration layer (OL), and the infrastructure layer (IL).

- The SL comprises business management concerned with service life-cycle, providing operation support system (OSS) and business support system (BSS) functions related to services from different tenants (e.g. enterprise users, service providers, etc.). It also executes the NFVO and VNFM functions for the NS and VNF life-cycle management.
- OL acts as a policy enforcement and VIM component to provide resource orchestration (RO module) and deliver virtual resources views to SL. It also includes the controller adapter (CA), and a multi-domain, multi-technology, and multi-vendor controller. CA provides computing and networking abstraction by collecting virtualised resources from lower layer domain-specific controllers and organising them into a global virtualised resource view. CA offers an independent technology control to the RO module. In the OL, SDN is also used for the creation and integration of virtual networks in both domains.
- IL manages all IT and network resources (physical and virtual) needed for the VNF execution. For this, it uses two types of domain-specific controllers, (i) the compute controller (CC) to manage computational resources, and (ii) the network controller (NC) to manage network resources. IL considers different kinds of resources such as SDN enabled network nodes (e.g. OpenFlow switches), and cloud-enabled data centres (e.g. OpenStack).

In general, the proposed UNIFY architecture is currently used as basis for the implementation of several NFV/SDN solutions to different problems, such as Middleboxes virtualisation and virtualised customers premises equipment (vCPE), which we discuss in next Section.

 © IoRL consortium 2018

2.3.2.3.1 **Project SONATA**

Project SONATA [121] (2015-2017) targets to address two main technological challenges envisioned for the 5G system: (i) flexible programmability and (ii) deployment optimisation of software networks for complex services/applications. SONATA provides an integrated development and operational process for supporting network function chaining and orchestration [125]. The major components in SONATA solution consist of two parts: (i) the SONATA software development kit (SDK) that supports functionalities and tools for the development and validation of VNFs and NS, and (ii) the SONATA service platform, which offers the functionalities to orchestrate and manage network services during their lifecycles using MANO framework, as well as interact with the underlying virtual infrastructure through virtual infrastructure managers (VIM) and wireless access network (WAN) infrastructure managers (WIM) [126]. The project describes the use cases envisioned for the SONATA framework and the requirements extracted from them. These use cases encompass a wide range of network services including NFVI-as-a-service (NFVIaas), VNF-as-a-service (VNFaaS), v-content delivery network (CDN), and personal security. One of the use cases consists of hierarchical service providers simulating one multi-domain scenario. In this scenario, service programming and orchestration for virtualised software networks (SONATA) does not address the business aspects only the technical approaches are in scope. SONATA intends to cover aspects in the cloud, SDN and NFV domains [127]. Moreover, the project proposes to interact and manage with not only VNFs also support legacy [128]. Besides, it describes technical requirements for integrating network slicing in the SONATA platform. SONATA framework complies with ETSI NFV-MANO reference architecture [128].

2.3.2.4.1 **Project VITAL**

Project VITAL [122] (2015-2017) is to address the integration of terrestrial and satellite networks through the applicability of SDN and NFV technologies. Its main outcomes are (i) the virtualisation and abstraction of satellite network functions, and (ii) the support of multi-domain service and resource orchestration capabilities for a hybrid combination of satellite and terrestrial networks [129]. The architecture followed by VITAL is in line with the main directions established by ETSI ISG NFV [130], with additional concepts extended to the satellite communication domains and network service orchestration deployed across different administrative domains. This architecture includes functional entities (i.e. NFVO, VNFM, SO, Federation Layer) for the provision and management of the NS lifecycle. In addition, a physical network infrastructure block with virtualisation support includes SDN and non-SDN (legacy) based network elements for flexible and scalable infrastructure management. Implementing the relevant parts of the VITAL architecture, X-MANO (2017) [131] achieves a promising cross-domain network service orchestration framework. It supports different orchestration architectures such as hierarchical, cascading and peer-to-peer. It also introduces an information model and programmable network service in order to enable confidentiality and network service lifecycle programmability, respectively.

2.3.2.5.1 **Project 5GEx**

Project 5GEx (2015-2018) investigates end-to-end network and service elements to correlate multi-vendor, heterogeneous technology and resource environments. Its target is to evolve business relationships among administrative domains, including possible external service providers without physical infrastructure resources. Architecturally, 5GEx addresses business-to-business (B2B) and business-to-customer (B2C) relationships across multi-

 © IoRL consortium 2018

administrative domain orchestrator that might interface different technological domains. 5GEx extends ETSI NFV-MANO reference architecture with new functional components and interfaces as well as by defining modules for topology abstraction, topology distribution, resource repository, service level agreement (SLA) management, policy database, resource monitoring, service catalog, inter-provider NFVO. 5GEx rely on the outcomes and open source components of projects Unify and T-NOVA.

2.3.2.6.1 **Project 5G-Transformer**

Project 5G-Transformer [124] (2017-2019) is to transform current mobile transport network into a type of mobile transport and computing platform (MTP) using SDN, NFV, orchestration, and analytics, which brings the network slicing paradigm into mobile transport networks. The project aims to support a variety of vertical industries such as automotive, healthcare and media/entertainment. 5G-Transformer defines three new components to the proposed architecture: (i) vertical slicer as a logical entry point to create network slices, (ii) service orchestrator (SO) for end-to-end service orchestration and computing resources, and (iii) mobile transport and computing platform for integrate fronthaul and backhaul networks. The main decision point of the proposed system is the service orchestrator, which interacts with other SOs to provide the end-to-end service (de)composition of virtual resources and orchestrate the resources across multiple administrative domains. The project architecture is still ongoing.

2.3.2.2 **NFV middleboxes virtualisation**

Middleboxes are network devices used for traffic manipulation via packet forwarding to increase system performance (e.g. traffic shaping, load balancing, TCP optimisation) and provide security functionalities (e.g. firewalls, intrusion detection, prevention systems (IDPS), deep packet inspection (DPI)). They can be categorised into hardware- and software-based middleboxes.

- Hardware-based NFV middleboxes [55] incur high OPEX costs due to the management complexity. They also come from different manufacturers and must be deployed, configured, and managed individually. Furthermore, they incur high CAPEX costs due to systematic hardware maintenance needed. When new network functions are necessary, enterprises must purchase one or more middleboxes due to the inflexibility in proprietary hardware that creates vendor lock-in and limits innovation.
- Software-based NFV middleboxes use NFV/SDN architecture to deal with the aforementioned challenges shown in Figure 2-7. The primary objectives are to reduce both CAPEX and OPEX and provide fast delivery of network function, elasticity, and dynamic service chaining. In particular, NFV can effectively manage virtual middleboxes, while SDN can provide interconnection between VNFs to deliver network services by means of service function chaining (SFC). In this context, [132] considers two approaches to redirect network traffic to virtualised middleboxes for further processing, namely Bounce and IP redirections. In the case of Bounce Redirections, a certain enterprise gateway uses tunneling techniques to redirect both ingress and egress traffic to the virtual middleboxes (grouped into SFC). This approach requires minimal configuration (i.e. few static rules) at the gateway since it only redirects traffic to a cloud provider hosting the middleboxes. However, Bounce Redirections might increase the end-to-end delay due to these redirections for each

 © IoRL consortium 2018

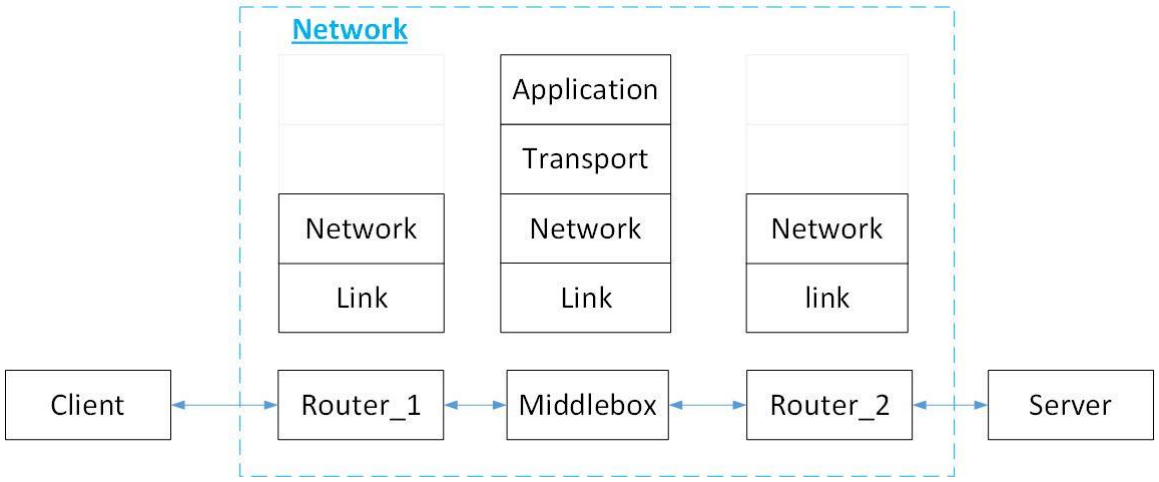


Figure 2-7 - Illustration of the setting of NFV middlebox in the end-to-end virtual network

packet. On the other hand, to avoid the extra round-trips of the Bounce Redirection, IP Redirection allows routing traffic directly to/from the cloud provider. In this case, the cloud can be located in the middle of the communication between enterprise and external sites. However, the provider must announce naming and addressing on the company's behalf (e.g. DNS redirection).

In this regard, NFV/SDN approaches have been proposed to deal with Middleboxes virtualisation. For example, authors in [55] (2015) propose an NFV/SDN framework, so-called Glasgow network functions (GNF), to deploy and manage container-based network services in public [133] and private [55] cloud environments. This framework aims at overcoming the limited network re-configurability in these scenarios, delivering network programmability and fast deployment of new network services. GNF comprises three planes. (i) The infrastructure plane to encompass the physical resources of network and computations, and incorporate edge devices (e.g. virtual routers, IoT gateways, etc.) [56]. (ii)-(iii) The VIM and Orchestration planes, which are responsible for resource orchestration, where operator deploys the GNF agent on all cloud servers and all edge devices by applying (a) local VNF instantiation with Docker Engine [48] for fast deployments and low resource utilisation, and (b) OpenFlow rules (via OVSDB) and virtual switches for local traffic steering management, respectively. The GNF Manager is responsible for receiving NFV service requests (service plane) and performing the necessary operations using the OpenDayLight and GNF agent instances.

Moreover, the study in [36] (2016) focus on the UNIFY project to extend its architecture by creating a service function chaining control plane solution. The goal is to support SFC in distributed cloud scenarios, where VNFs from the same SFC can run in different NFVI-PoPs. To do so, authors built prototype framework, called extensible service chain prototyping environment (ESCAPE) implemented in Python, on top of an OpenFlow POX controller. This prototype can work with two protocols in the infrastructure layer (IL): Cloud and OpenFlow protocol. The OpenStack cloud platform [134] and the OpenDayLight controller [13] perform the management of cloud domains, while VNFs are deployed as kernel-based virtual machines (KVMs). The OpenFlow protocol handles transport networks with Linux nodes running Open vSwitches (OpenFlow support). The POX controller [35] (network management) and the NETCONF/YANG (VNF management) manage these protocols, while

 © IoRL consortium 2018

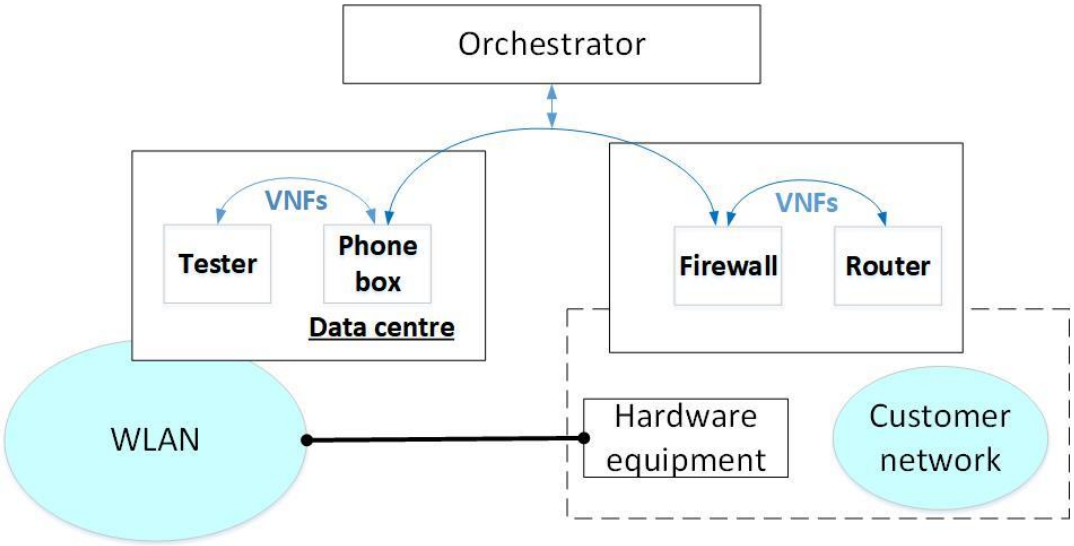


Figure 2-8 - Illustration of the vCPE functionality in the network and customer site

VNFs are deployed as distinct processes (Linux groups) and run network functions implemented in modular router.

Authors in [135] (2015) develop the VNGuard framework, which uses NFV to provide fast and dynamic virtual firewalls in a cloud environment, for the protection of VNs. To address VN's changeable topology (i.e. VMs dispersion and migration), VNGuard framework uses SDN to provide fast and flexible traffic steering to the virtual firewalls. It operates VIM using OpenStack and VNF using CloudLab, which is a testbed that provides an infrastructure-as-a-service (IaaS) for cloud-based experiments [136].

2.3.2.3 NFV customers premises equipment

Traditionally, customers premises equipment (CPE) includes all equipment within the customer domain that receives a communication service (e.g. router, modem, etc.). CPEs have been a barrier to the current goals of both telecommunications companies and service providers, due to the high cost of maintenance, management difficulties, and the impossibility of remote upgrades. A promising solution to this issue is to apply virtualisation in traditional CPE using NFV architecture (vCPE) shown in Figure 2-8.

In particular, vCPE is a service in which some or all of the functions associated with CPE are virtualized [137]. Its main difficulty is how to instantiate network services in distributed infrastructures (using multiple NFVI-PoPs, so-called distributed NFV) [138], where VNFs are placed either in the service provider cloud platform (cloud CPE) or the on-premise CPE, depending on where they are most efficient regarding latency, available resources, etc. A solution to this problem is to apply the UNIFY architecture, which includes three layers. The service layer application (SLApp), which represents the UNIFY SL and it main function is to enable different players (operators and end users) to select their network services by including an authentication mechanism and providing a high-level data model for defining flexible network services, called service graph (SG). The global orchestrator (GO), which represents the UNIFY orchestration layer (OL) and is to manage the forwarding graph (FG) received from SLApp to enable the network service deployment according to the VNF requirements and infrastructure capabilities. Note that in order to allow distributed NFV, GO

 © IoRL consortium 2018

includes multiple control adaptors for coordination of different infrastructures, and an orchestrator component, which is responsible for the centralised coordination of multiple control adaptors. On this basis, the GO can (i) select one of the infrastructures to implement all the network service requested and (ii) host network services using either the integrated or the OpenStack protocol. In particular, the integrated protocol represents the CPE (home gateway) and receives an FG from the GO through the node resource manager (NRM) via REST API. The NRM will instantiate all VNFs using Docker containers and DPDK process. For traffic steering, NRM uses an extensible data-path daemon (xDPd) to create an OpenFlow switch (and its correspondent controller) for each FG. Separately, the OpenStack node represents a data centre and uses the OpenStack cloud platform for network service deployment such that the KVM hypervisor can create the VNFs, and the OpenDayLight and Open vSwitch can control the traffic steering. In this research direction, the works in [139] and [140] propose the Cloud4NFV platform, which consists an NFV/SDN framework for Telco network virtualisation. Cloud4NFV platform considers multiple NFVI-PoPs and WAN domains when deploying new service function chaining along with a topology with multiples customer sites (NFVI-PoPs). All NFVI-PoPs include an OpenStack distribution working as a Cloud VIM and an OpenDayLight controller to provide VNF connectivity. The VNFs are CPE functions, and can be deployed as VMs in the NFVI-PoP closest to the customer.

2.3.2.4 **On-demand and application-specific traffic steering**

Traffic steering is the ability to direct users' requests to the appropriate service/content sources, and it can be dependent either on the available networking resources and capabilities on the client and server side or user's permissions and location. For example, for a user who requests video streaming service that has stringent application performance requirements, on-demand and application-specific traffic steering can guarantee the efficient network resource usage and required quality-of-experience (QoE) for the user.

In the NFV framework context, SDN can enhance traffic steering between VNFs, providing dynamic service chaining as shown in Figure 2-9. This is because, with the separation of control and data planes, SDN can enable the exchange of information between the application and network layers, allowing users' services to have an overview of the general state of the network, and to make intelligent decisions (to meet service requirements) on how to steer traffic through VNFs in optimal manner. In this research direction, authors in [141] (2015) propose a joint NFV/SDN design of cross-layer interface orchestration (CLO) between the application and network layers that allows deploying network services with on-demand and application-specific traffic steering. The proposed approach comprises by application, control, and infrastructure layers. The application layer consists of network services, which interface together with the API at the control plane in order to communicate their network requirements (e.g., bandwidth, maximum latency). The control plane has a global view of computer and network resources and provides the traffic steering capabilities to the application layer, with its main component to be the cross-layer orchestrator (CLO) that acts as an NFVO and VNFM manager of the lifecycle of services over the cloud (OpenStack-based) and WAN protocols (OpenFlow-based). The CLO solution can be implemented using the OpenSDNCore orchestrator [142] and by using Java programming language.

 © IoRL consortium 2018

2.3.3 Research advancements on NFV wireless networking

Modern wireless communication demands arise new network system requirements such as mobility support, programmability, fast delivery of network services, performance, security, etc. However, the management and configuration of today's large WiFi networks are complex and inflexible, while they do not include some application requirements or user needs. In the following, we present the main problems addressed by NFV together with SDN architectures designed for different wireless networks scenarios.

2.3.3.1 Wireless LAN (WiFi)

Most recent studies related to WLANs leverage abstraction of the so-called virtual access point (VAP) by moving the MAC layer or middleboxes processing to the cloud. When associated with the wireless network, each client acquires a VAP that is dedicated to this client, e.g., each VAP is independent of the client migrating from one access point to another (handover). Authors in [52] (2017) proposed an NFV/SDN approach, namely OpenSDWN, to implement per client access points and virtual middleboxes. In particular, for the access point case, the authors created an extension to Odin SDN framework [143], called light virtual access point (LVAP), which uses SDN applications to abstract some functionality of the 802.11 access point (AP), such as authentication, handoff, and client associations. A physical AP can support multiple LVAPs (i.e. one for each client) meaning that a certain

LVAP can serve as dedicated link between its client and infrastructure. The authors in [52] also implemented virtual middleboxes (e.g. firewall), which can be deployed either on a middlebox server or at the access point itself and can be integrated using LVAPs with virtual networks. A service differentiation mechanism classifies the data flows to redirect traffic to the correct v-middleboxes (vMB). The OpenSDWN controller performs all the above functionalities, using the Floodlight and ONOS [140] as SDN controllers. Such an abstraction allows the seamless mobility with the migration of both LVAPs and vMBs among APs.

 © IoRL consortium 2018

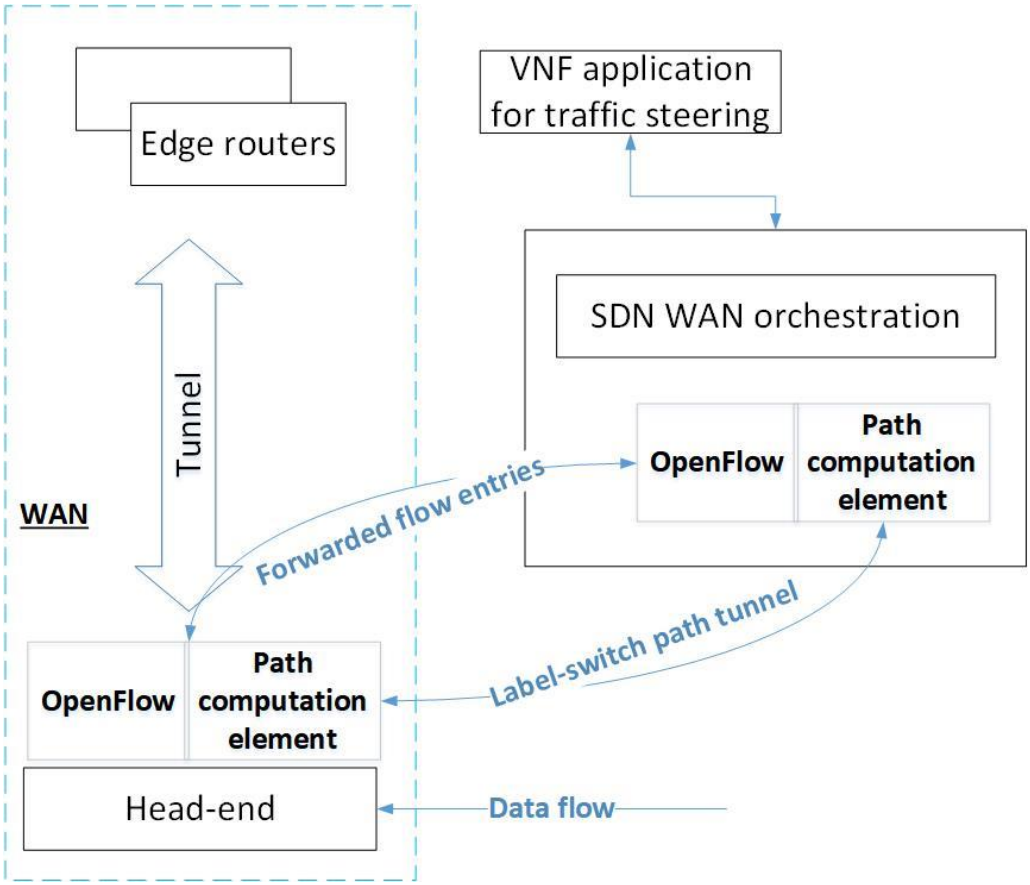


Figure 2-9 - Illustration of the OpenFlow mechanism for traffic steering

Furthermore, authors in [144] extended CloudMAC OpenFlow framework [145] to provide QoS for VAPs, by setting each VAP as a VM in cloud environment, which is responsible for the MAC layer management frames (e.g. beacons, probes request/response). In this study, the physical AP redirects these frames to the destination VAP, using the OpenFlow rules, while the QoS mechanism implements VAP traffic prioritisation using different strategies of queue management (e.g. stochastic fair queueing) for all the considered open vSwitches between the APs and VAPs. The OpenDayLight controller is used to manage both traffic redirection, prioritisation and handover rules.

2.3.3.2 Wireless MESH

To the best of our knowledge, the study in [146] (2016) is the only effort to examine NFV/SDN solutions for wireless MESH networks (WMNs). In particular, the authors propose the UrbanX, which is a multi-radio cognitive mechanism to dynamically self-adapt when there are variations in the interference conditions on the WiFi channels. For this, each mesh node includes an OpenFlow agent component that allows the instantiation of VNFs, while the VIM component controls all these components. It uses an OpenFlow controller to establish a path for end-to-end connectivity, including one or more mesh nodes, i.e., the VIM uses the OpenFlow agent components to monitor link status and configure the path with minimum latency. At the same time, the VIM instantiates TCP accelerators as VNFs at each mesh node included in the path. In that way, the Urban-X improves TCP throughput to the mesh clients.

 © IoRL consortium 2018

2.3.3.3 **Network slicing**

Network slicing is the partitioning of a certain physical infrastructure, composed of both network and computational resources, into multiple logical networks, called network slices, where each slice is a self-contained network with its own virtual resources created on top of the underlying infrastructure. Slices can be designed and optimised for a particular mobile operator or service provider providing (i) better customisation of logical networks according to service requirements; (ii) on-demand provisioning to scale resources up or down as conditions change, and iii) network resource isolation for improved security and reliability, compared to traditional physical networks. NFV and SDN technologies are capable of providing the flexibility required for providing efficient resource sharing, traffic differentiation per slice, and management and protection tools. For instance, a use case for the use of NFV/SDN architectures in network slicing is for the creation of SDNenabled Virtual Tenant Networks (VTNs). VTNs are virtual networks deployed to different tenants in an isolated way (independent of underlying physical network resources) to support specific

QoS and service level agreement (SLA) requirements. By enabling network programmability, SDN renders the abstraction necessary for its use as a network hypervisor. In the case of SDN-enabled VTNs, one or more SDN controllers create a VTN (called an Infrastructure SDN controller), while a new SDN controller is instantiated to manage this VTN (called a tenant SDN controller). When an SDN-enabled VTN deployment takes place, the respective tenant SDN controller can be manually installed and configured on a dedicated server, which is time-consuming and complex process. However, using NFV/SDN, such tenant SDN controllers can be virtualised towards faster and more dynamic VTN provisioning. In this regard, authors in [41] and [57] (2016) propose an NFV/SDN solution for fast and dynamic deployment of SDN-enabled virtual tenant networks over multiple data centres and WAN domains. The proposed solution aims at providing geographically distributed cloud services with specific QoS and SLAs. Particularly, NFV and cloud have been used to virtualise tenant SDN controllers (either via OpenDayLight or Floodlight) and control the underlying SDN-enabled VTNs, providing fast and dynamic VTN provisioning. Also, for each data centre, OpenStack have been utilised as VIM along with an OpenDayLight controller to interconnect a virtual tenant SDN controller with its respective VTN. To create the VTNs, multidomain SDN orchestrator (MSO) mechanism is used as a network operating system (NOS). The MSO is to create abstractions over multiple domains including different transport network technologies, enabling thereby the composition of end-to-end services over heterogeneous WAN networks (HetNets). Multidomain network hypervisor (MNH) is also used to create end-to-end SDN-enabled VTNs, over the abstraction provided by MSO. Based on the developed global cloud and network orchestrator, the architecture can integrate geographically distributed data centres and multiple WAN domains, providing a unified cloud and network operating system for the creation of end-to-end NFV services over VTNs.

Furthermore several other research attempts focus on providing network slicing for the new generation of mobile network such as [147], [148], [149].The 3GPP has identified network slicing as the key technology to achieve the goals in the 5G system [148] due to its potential to enable suitable flexibility addressing specific requirements of different use cases. In this scenario, mobile operators can share the same physical network substrate, adding their virtual networks with their services (e.g. 3G, 4G services) and a centralised management plane, creating the so-called mobile virtual network operators (MVNO). As a way to maintain privacy between operators, these logical networks are isolated from each other, which

 © IoRL consortium 2018

makes NFV to provide the mobile network services per operator as VNFs and, in turn, SDN creates the slice as well as establishes network functions interconnectivity.

Authors in [150] (2016) design an NFV/SDN prototype architecture to support MVNOs in cloud environment using non-open source FOKUS OpenSDNCore orchestrator [142] to coordinate the network services between MVNOs. The proposed orchestrator uses OpenStack as VIM to create the virtual tenant networks and to instantiate VNFs per mobile operator such as evolved packet core (EPC) and IP multimedia subsystem (IMS). Furthermore, [148] proposes a three-layer network slicing framework model for 5G networks considering NFV and SDN technologies. The bottom layer is the 5G software defined infrastructure (5G-SDI), which comprises multiple administrative and physical domains (e.g. RAN, transport, core networks, etc.) with SDN-based control and management. Their SDN-based approach uses hierarchically organised SDN controllers to provide abstraction and distributed dynamic allocation of resources. In addition, RAN and MEC can be deployed to enable a cloud-based infrastructure, while the virtual resource layer can create network slices with virtual resources (radio, computing, and network) and VNFs that are customised to meet the requirements of different types of services. The application and service layer include the per-tenant services (e.g. connected vehicles, virtual reality (VR), etc.) that will use these slices to perform their functionalities. Also, the life cycle of network slices is managed and orchestrated by the slicing MANO that acts as (i) VIM, (ii) VNF manager, and (ii) slice orchestrator. The NFV/SDN architecture proposed in [148] has been adopted by many recent studies such as [58], [59], [60], [61], [62], [151] to implement 5G scenarios, including the entire mobile network implementation approach (e.g. radio access network) for fast and dynamic deployment of MVNOs. Besides, authors in [62], [152] propose the ADRENALINE testbed, which can be considered on top of an NFV/SDN platform. More precisely, ADRENALINE is an SDN/NFV packet/optical transport network and edge/core cloud platform, which encompasses multiple interrelated although independent components and prototypes, to offer end-to-end services, interconnecting users and applications across a wide range of heterogeneous network and cloud technologies for the development and test of 5G and IoT services in conditions close to production systems.

2.3.3.4 **Mobile edge computing**

Mobile edge computing (MEC) or multi-access edge computing has been a trend in mobile networks. Like NFV, the MEC architecture has been standardised by ETSI through Group specification (GS) MEC [153], [154] since 2016. MEC provides IT and cloud computing capabilities within the radio access network (RAN). For this, a set of computer and storage resources (e.g. data centres, clusters, etc.) are deployed at the edges of the mobile operator's network to assist the core data centre in supporting computing and communication [155]. MEC focuses on delivering the services closest to the user, as a way to meet certain critical application (e.g. video analytics, Internet-of-Things, augmented reality, and data caching) requirements that are not supported only by cloud computing, such as high bandwidth, low latency and jitter, context awareness, and mobility support. According to 5G-PPP, MEC is vital technological component to enable 5G networks [156]. NFV/SDN architectures are in line with current trends for MEC solutions, which however, is a new technology thus, only a few studies have been found in this SLR. As an example, the EU H2020 SELFNET project [157] (2016) investigates the design and implementation of the so-called autonomic management framework for 5G networks, using technologies such as SDN, NFV, self-organising network (SON), cloud computing, and artificial intelligence. This

 © IoRL consortium 2018

framework aims at reducing OPEX and improving QoE of the end users, addressing (i) self-protection against distributed cyber-attacks, ii) self-healing against network failures, and iii) self-optimisation of the network traffic. In this context, authors in [158] (2017) propose a SELFNET approach to support SFC in MEC scenarios in order to meet 5G requirements defined by the 5G-PPP initiative [159]. SELFNET relies on a federated cloud infrastructure (i.e. multiples edge NFVI-PoP and a core NFVI-PoP) to provide IT and network resources and execute VNFs that support some management elements and network services. The WAN infrastructure management (WIN) uses SDN controllers to provide connectivity between edge NFVI-PoPs and the core NFVI-PoP through the creation of virtual tenant networks.

2.3.4 **Orchestrators**

NFV orchestration is used to coordinate the resources and networks needed to set up cloud-based services and applications. This process uses a variety of virtualisation software and industry standard hardware. The big advantage of NFV is that it uses industry-standard commercial off-the-shelf (COTS) hardware to deliver a service via software. Prior to the advent of the NFV, operators build application-specific networks using proprietary hardware. Now, these services can be deployed as VNFs on a NFV platform. This includes popular software services such as a virtual firewall, virtual load-balancing, or other software-defined wide area network (SD-WAN) service. Because NFV requires lots of virtualized resources, it requires a high degree of software management, referred to as orchestration. Orchestration coordinates, connects, monitors, and manages the needed resources from the platform for the NFV services. Orchestration may need to coordinate with many network and software elements, including inventory systems, billing systems, provisioning tools, and operating support systems (OSSs). However, some of the existing orchestrating solutions are just tied to a specific networking environment, and moreover, some of them can orchestrate an only limited number of services [160]. This section presents an overview of main orchestration frameworks, including open source, proposed and commercial solutions.

2.3.4.1 **Open source MANO orchestrator**

Open source MANO [161] (2017) is an ETSI-hosted project to develop an open source NFVMANO platform aligned with ETSI NFV information models and that meets the requirements of production NFV networks. The project launched its third release [162] in October 2017 and presented improvements in security, service assurance, resilience, and Interoperability. One of its main goals is to promote the integration between standardisation and open source initiatives. The OSM architecture has a clear split of orchestration function between resource and service orchestrators. It integrates open source software initiatives such as Riftware as network service orchestrator and GUI, OpenMANO as resource orchestrator (NFVO), and Juju 5 server as configuration manager (G-VNFM). The resource orchestrator supports both cloud and SDN environments. The service orchestrator can provide VNF and NS lifecycle management and consumes open information and/or data models, such as YANG. MANO architecture covers only single administrative domain.

2.3.4.2 **ONAP orchestrator**

ONAP [163] (2017) is based on the union of two open source MANO initiatives namely OPEN-O [164] and OpenECOMP [165] frameworks. ONAP software platform deploys a unified architecture and implementation, with robust capabilities for the design, creation, orchestration, monitoring and lifecycle management of physical and virtual network functions. ONAP's functionalities are expected to address automated deployment and

 © IoRL consortium 2018

management and policies optimisation through achieving intelligent operation of network resource using big data and artificial intelligence [166]. Authors in [167] identify two of the biggest challenges to merge two large sets of code using ONAP. First, to define a higher-level common information model unifying the predominant data models used by OPEN-O (TOSCA) and OpenECOMP (YANG). Second, to create a standard process so that end users can introduce onboarding and lifecycle management of VNFs using an automated process.

2.3.4.3 X-MANO orchestrator

X-MANO [168] (2017) is an orchestration framework to coordinate end-to-end network service delivery across different administrative domains. X-MANO introduces components and interfaces to address several challenges and requirements for cross-domain network service orchestration such as business aspects with architectural considerations, confidentiality, and lifecycle management. In the business aspects case, X-MANO supports hierarchical, cascading and peer-to-peer architectural solutions by introducing a flexible, deployment-agnostic federation interface between different administrative and technological domains. The confidentiality requirement is addressed by the introduction of a set of abstractions (backed by a consistent information model) so that each domain advertises capabilities, resources, and VNFs without exposing details of implementation to external entities. To address the multi-domain lifecycle management requirement, X-MANO introduces the concept of programmable network service based on a domain specific scripting language allowing network service developers to use a flexible programmable Multi-Domain Network Service Descriptor (MDNS), so that network services are deployed and managed in a customised way.

2.3.4.4 Open Baton orchestrator

Open Baton [169] (2017) is an open source reference implementation of the NFVO based on the ETSI NFV MANO specification and the TOSCA standard. It comprises a vendor-independent platform (i.e. interoperable with different vendor solutions), which is easily extensible for supporting new functionalities and existing platforms. Current Open Baton release 4 includes many different features and components for building a complete environment fully compliant with the NFV specification. Among the most important are a NFVO following ETSI MANO specification, a generic VNFM to deploy Juju charms or Open Baton VNF packages, a marketplace integrated within the Open Baton dashboard, a driver mechanism supporting different type of VIMs without having to re-write anything in the orchestration logic, and a powerful event engine for the dispatching of lifecycle events execution. Finally, Open Baton is included as a supporting project in the project Orchestra6. This OPNFV initiative seeks to integrate the Open Baton orchestration functionalities with existing OPNFV projects in order to execute testing scenarios (and provide feedbacks) without requiring any modifications in their projects.

2.3.4.5 Agile reference Implementation of automation orchestrator

Agile Reference Implementation of Automation (ARIA) TOSCA [170] (2017) is a framework for building TOSCA-based orchestration solutions to support multi-cloud and multi-VIM environments, while offering a command line interface (CLI) to develop and execute TOSCA templates with an easily consumable software development kit (SDK) for building TOSCA enabled software. By taking advantage of its programmable interface libraries, ARIA can be embedded into collaborative projects that want to implement TOSCA-based orchestration.

 © IoRL consortium 2018

For example, the Linux-based Open-O orchestrator [164] uses the ARIA TOSCA codebase to create its SDN/NFV orchestration tool in Cloudify framework [171].

2.3.4.6 **TeNOR orchestrator**

TeNOR [172] (2016) is a multitenant multi NFVI-PoP orchestration platform developed by project T-NOVA [119], with main focus to manage the entire NS lifecycle service, optimising the networking and IT resources usage. TeNOR approaches an architecture based on a collection of loosely coupled, collaborating services, known as micro-service architecture, which can ensure a modular operation of the system. Micro-services are responsible for managing, providing and monitoring NS/VNFs, in addition to forcing SLA agreements and determining required infrastructure resources to support a NS instance. TeNOR architecture is split into two main components namely (i) network service orchestrator, which is responsible for NS lifecycle and associated tasks, and (ii) virtualized resource orchestrator, which is responsible for the management of the underlying physical resources. To map the best available location in the infrastructure, TeNOR implements service mapping algorithms using NS and VNF descriptors. Both descriptors follow the TeNOR's data model specifications that are a derived and extended version of the ETSI name server daemon (NSD) and VNF descriptor (VNFD) data model.

2.3.4.7 **Tacker orchestrator**

Tacker [173] (2016) is an official OpenStack project building a generic VNFM and a NFVO to deploy and operate network services and VNFs on a cloud/NFV infrastructure platform such as OpenStack. It is based on ETSI MANO architectural framework and provides a functional stack to orchestrate end-to-end network services using VNFs. The NFVO is responsible for the high-level management of VNFs and managing resources in the VIM. The VNFM manages components that belongs to the same VNF instance controlling the VNF lifecycle. Tacker also performs mapping to service function chain (SFC) and supports auto scaling and TOSCA NFV profile using heat-translator. Tacker's components are directly integrated into OpenStack and thus, provide limited interoperability with other VIMs. However, Taker combines the NFVO and VNFM into a single element, which means that its functionalities are limited and divided, while it works in single domain environments offering limited design flexibility.

2.3.4.8 **Cloudify orchestrator**

Cloudify [174] (2015) is an orchestration-centric framework for cloud orchestration focusing on optimization NFV orchestration and management. It provides a NFVO and generic-VNFM in the context of the ETSI NFV, and can interact with different VIMs, containers, and non-virtualised devices and infrastructures. Cloudify is aligned with MANO architecture but not fully compliant. Besides, Cloudify provides full end-to-end lifecycle of NFV orchestration through a simple TOSCA-based blueprint following a model-driven and application-centric approach. It includes agile reference Implementation of automation (ARIA) as its core orchestration engine providing advanced management and ongoing automation. In order to help contribute to open source NFV-MANO adoption, Cloudify sponsors diverse NFV projects and standard organisations, such as TOSCA specification, ARIA and ONAP.

2.3.4.9 **XOS orchestrator**

XOS [175] (2015) is a single uniform programming environment to unify SDN, NFV, and cloud services (all running on commodity servers) towards the Everything-as-a-Service (XaaS) approach. XOS software is organised around three main layers namely (i) data

 © IoRL consortium 2018

model, which can record the logically centralised state of the system, (ii) set of views (running on top of data model) for customising access to the XOS services, and controller framework, which is responsible for distributed state management. XOS runs on the top of a mix of service controllers such as data centre cloud management systems (e.g. OpenStack), SDN-based network controllers (e.g. ONOS), network hypervisors (e.g. OpenVirtex), virtualised access services (e.g. CORD), etc. This collection of services controllers allows the mapping to XOS onto the NFV architecture playing the role of a VNFM. Using XOS as VNFM facilitates controlling both sets of EMs and VIM [176].

2.3.4.10 **Gohan orchestrator**

Gohan [177] (2015) is a MANO-based initiative for SDN and NFV orchestration, which keeps the system architecture and deployment model simple by relaying on micro-services within a single unified process. Particularly, Gohan services use JSON schema based on which the orchestrator can deliver the so-called schema-driven service deployment that includes REST-based API server, database backend, command line interface (CLI), and web user-interface (WebUI). Some applicable use cases for the NTT's Gohan approach are to use such orchestration (i) in the service catalog on top of cloud services, and (ii) as a kind of NFV MANO in order to manage both Cloud VIM and legacy network devices

2.3.4.11 **ESCAPE orchestrator**

Extensible Service ChAin Prototyping Environment (ESCAPE) [178] (2013) is a NFV framework, which supports the three main layers of the project UNIFY architecture, i.e., (i) service layer, (ii) orchestrator layer and, (iii) infrastructure layer [178]. It operates as a multi-domain orchestrator for different technological domains, as well as different administrative domains. ESCAPE can support remote domain management (recursive orchestration) and operate based on joint resource abstraction models for networks and clouds [179]. Upon receiving a specific service request on its REST API of the service layer, ESCAPE sends the requested service function chains to the orchestration layer and maps the service components to its global resource view. The calculated service parts are then sent to the corresponding local orchestrators towards instantiating the service.

2.3.5 **NFV cloud computing platforms**

Cloud management platforms are integrated tools that provide management of cloud environments. These tools incorporate self-service interfaces, provisioning of system images, enabling metering and billing, and providing some degree of workload optimisation through established policies. Through the self-service interface (e.g. based on OCCI) the user can request virtual infrastructure. This request is issued to a Cloud Controller, which provisions this virtual infrastructure somewhere on available resources within the DC. The Cloud Controller provides the central management system for cloud deployments.

The most popular cloud management platforms include open source solutions such as OpenStack, CloudStack and Eucalyptus and commercial solutions from Microsoft and VMware. This section provides an overview of some of these solutions, on the Cloud Controller component. In the IoRL context, the Cloud Controller is a key component that can deliver end-to-end provisioning of virtual infrastructure, to enable full control over it and also to provide a detailed and real-time view of the infrastructure load.

 © IoRL consortium 2018

2.3.5.1 OpenStack NFV platform

OpenStack is a cloud OS that controls large pools of compute, storage and networking resources throughout a DC, all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface. As an open source solution, OpenStack is developed and supported by a global collaboration of developers and cloud computing technologists. The project seeks to deliver solutions for all types of clouds by being simple to implement, scalable and feature rich. The technology consists of a series of interrelated projects delivering various components for a cloud infrastructure solution. All OpenStack source code is available under an Apache 2.0 license.

OpenStack has a modular design that enables integration with legacy and third-party technologies. It is built on a shared-nothing, messaging-based architecture with modular components, each of which manages a different service; these services, together to instantiate an IaaS Cloud. The primary component of the cloud operating environment is the Nova compute service. Nova compute orchestrates the creation and deletion of compute/VM instances. Nova is designed to operate as much as possible as hypervisor-agnostic. It works with open source libraries such as libvirt.

2.3.5.2 Nova NFV platform

Similar to OpenStack components, Nova is based on a modular architectural design where services can be co-resident on a single host or, more commonly, on multiple hosts. The core components of Nova include the following:

- The nova-api accepts and responds to end-user compute API calls. It also initiates most of the orchestration activities as well as enforcing some policies;
- The nova-compute process is primarily a worker daemon that creates and terminates VM instances via hypervisor APIs (XenAPI for XenServer/XCP, libvirt for KVM or QEMU, VMwareAPI for vSphere, etc);
- The nova-scheduler process keeps a queue of VM instance requests and for each request it determines where the VM instance should run (specifically, which compute node it should run on).

Nova service itself does not come with a hypervisor, but manages multiple hypervisors, such as KVM or ESXi. Nova orchestrates these hypervisors via APIs and drivers. For example, Hyper-V is managed directly by Nova and KVM is managed via libvirt, while Xen and vSphere can be managed directly or through management tools such as libvirt and vCenter.

2.3.5.3 Eucalyptus NFV platform

Elastic utility computing architecture linking our programs to useful systems (Eucalyptus) is an open-source Cloud that provides on-demand computing instances and shares the same APIs as Amazon's EC2 cloud. Eucalyptus was designed as a highly-modular framework in order to enable extensibility with minimal effort (Eucalyptus Systems, Inc, 2014). The Cloud Controller (CLC) in Eucalyptus acts as the Cloud entry-point by exposing and managing the virtualised resources. The CLC offers a series of web services oriented towards resources, data and interfaces (EC2-compatible and Query interfaces). In addition to handling incoming requests, the CLC acts as the administrative interface for cloud management and performs high-level resource scheduling and system accounting. The CLC accepts user API requests from command-line interfaces like euca2ools or GUI-based tools and the Eucalyptus management console manages the underlying compute, storage, and network resources.

 © IoRL consortium 2018

2.3.5.4 Cloudstack NFV platform

Apache CloudStack is open source software designed to deploy and manage large networks of virtual machines, as a highly available, highly scalable Infrastructure as a Service (IaaS) cloud computing platform. CloudStack is used by a number of service providers (e.g. BT) to offer public cloud services, and by many companies to provide an on-premises (private) cloud offering, or as part of a hybrid cloud solution. CloudStack is a turnkey solution that includes the entire "stack" of features most organisations want with an IaaS cloud: compute orchestration, Network-as-a-Service, user and account management, a full and open native API, resource accounting and a first-class User Interface (UI).

CloudStack is a framework that allows pooling of computing resources in order to IaaS cloud services that can be used to provide IT infrastructure such as compute nodes (hosts), networks and storage as a service to the end users on demand. CloudStack Management Server is the main component of the framework, consisting of managing resources such as hosts, storage devices and IP addresses. The Management Server runs on a dedicated host in a Tomcat container and requires a MySQL database for persistence. The Management Server controls allocation of VMs to hosts and assigns storage and IP addresses to VM instances. This component also controls or collaborates with the hypervisor layers on the physical hosts over the management network and thus controls the IT infrastructure.

2.3.5.5 VMware vCloud suite NFV platform

VMware's vCloud Suite is a comprehensive, integrated cloud platform for building and managing cloud environments. Tools for cloud management are delivered through VMware vCenter Server, a centralised and extensible platform for managing virtual infrastructure. The tools included in the vCenter Server framework support: configuration of ESX servers and VMs, performance monitoring throughout the entire infrastructure, using events and alerts. The objects in the virtual infrastructure can be securely managed with roles and permissions.

2.4 IoRL-specific taxonomy approach of joint SDN and NFV structural elements

Having identified the architectural elements of joint SDN/NFV interpretation in contemporary networking, we summarise the elements to be used in the IoRL home network. In particular, the IoRL home network integrates:

- multiple controllers for distributed VNF performance, scalability, reliability, domain interaction and NaaS management;
- VNF/VNFC, network connectivity and virtual network management;
- OpenFlow, OVSDB and ForCES southbound APIs;
- RESTful northbound API;
- Open vSwitch SDN switch platform;
- virtualization of switchers and routers in NFV;
- SDN-based VIM for NFV-based allocation, deallocation and inventory management;
- SDN-based VNFs and OSS/BSS with initialisation, suspension and termination lifecycle;
- OpenDayLight controller;
- Open source MANO orchestrator.

 © IoRL consortium 2018

Recalling our generalised SDN/NFV taxonomy approach illustrated in Figure 2-1, we highlight in Figure 2-10 the key IoRL-specific SDN/NFV components, applications and services that will be utilised, assessed and integrated into the IoRL home network design.

 © IoRL consortium 2018

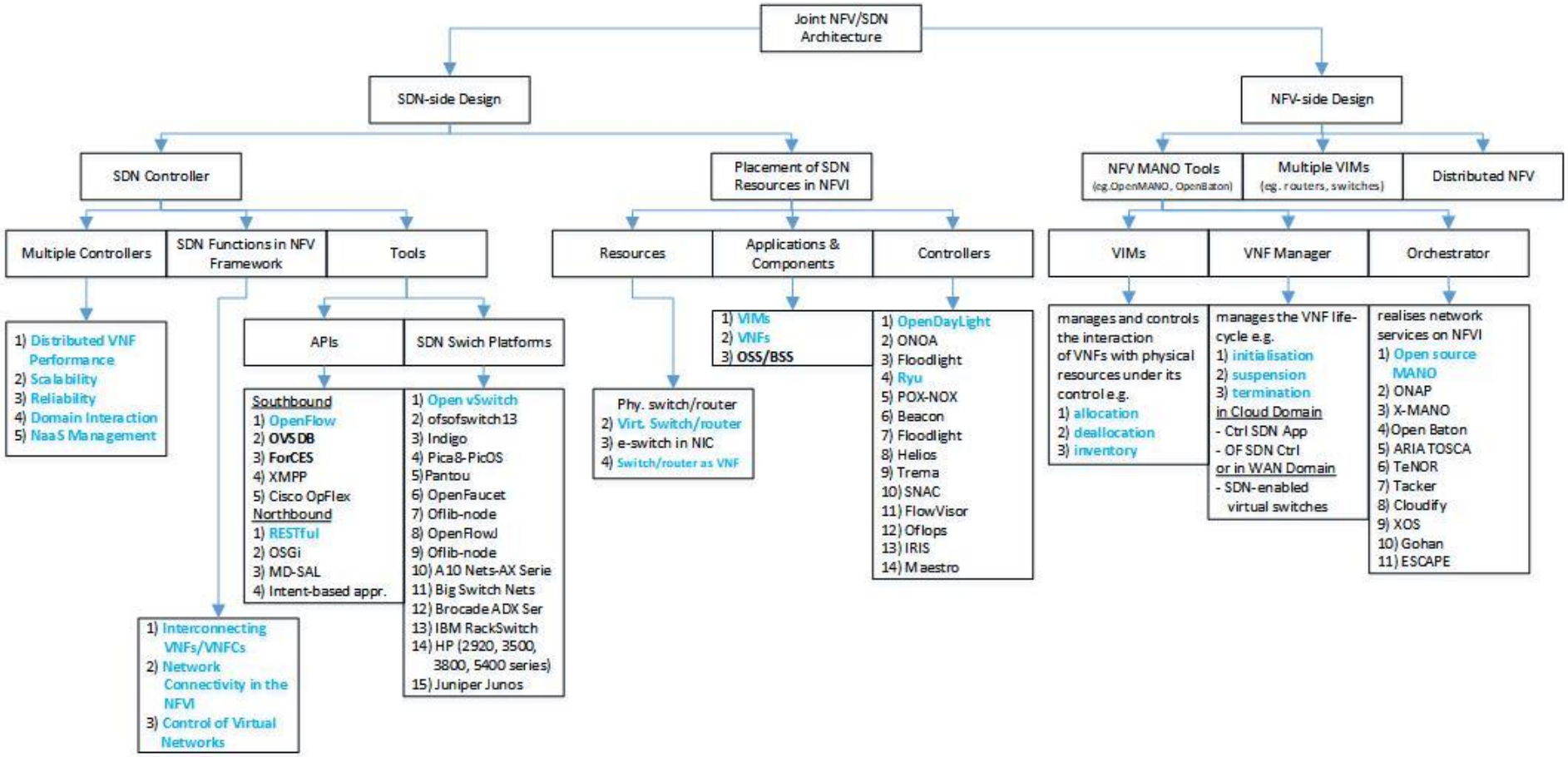


Figure 2-10 - Illustration of the IoRL-specific taxonomy approach of joint SDN and NFV structural elements

[end of section 2]

© IoRL consortium 2018

3 Integration of SDN/NFV approach in the IoRL home network

The IoRL approach includes designing and controlling a radio-light communication system that combines WLAN WiFi, mmWave and VLC access points (situated into RRLHs) at end-user position e.g. inside a house) to enable the user access in the network. In such system, the scope of joint SDN/NFV is to enable the intelligent data management and routing from cloud computers variously located remotely from the radio-light access points (e.g. in the home cell site or in external cloud network) to the different parts of the network. This can be achieved by developing an open source environment of network operations for homes, able to (i) effectively combine mmWave and VLC modules, while (ii) managing the requested network services via intelligent home IP gateway (IHIPG). The open source environment includes an integrated network management and operations plane API for buildings to abstract the home network infrastructure softwarisation, control plane and forwarding/data planes, and create customised network services for multi-network operators.

3.1 IoRL-specific design and perspectives with joint SDN/NFV integration

Having specified the role of joint SDN/NFV in the IoRL system, the considered software defined home network (SDHN) architecture can be summarised in Figure 3-1 and Figure 3-2.

Specifically, from Figure 3-1 we see that IoRL architecture relies on an intelligent home IP gateway enhanced with a SDN switch so that IP packet flow can be routed to RRLHs or WLAN. Such architectural approach can allow network service developers to write VNFs for location sensing, multiple-source streaming intra- and inter- handover and security monitoring, while providing the means to locate network operations and management functions between the Intelligent HIPG and the CHDC server in a configurable way to meet the different OPEX and CAPEX needs of different 4G/5G access points as shown in Figure 3-2. The concept of NFV can be applied to the VLC and mmWave RANs to off-load the complexity of the electronic systems required in the RRLH onto cloud home data centre (CHDC) servers or Intelligent HIPGs because of the very confined space available in Light Rose housing. The design of strategies for configuring the video streams to be transported with different percentage proportions over the different available home networks and paths (WLAN WiFi, VLC, mmWave) is also important for maintaining continued service connectivity in the presence of line-of-site (LoS) radio transmission systems. The SDHN is to be realised from OpenFlow-enabled network elements, such as HIPG and RRLH, by enhancing their dynamics using OpenFlow capabilities.

High speed gigabit IEEE Ethernet LAN technology will be used as the technical solution for wireline home networking since current fronthaul interfaces (e.g., common public radio interface (CPRI), open radio interface (ORI), open base station architecture initiative (OBSAI)) can be used to encounter capacity bottlenecks when confronted with the considered RRLH scenario that needs more transport capacity because it shifts more functional processing into the Intelligent HIPG and/or CHDC server processor. Our architecture could also support WiGig wireless LAN IEEE 802.11ad technology which can support the increasing demand for high data rates, with the standard providing 6.7 Gb/s using GHz of bandwidth at 60 GHz mmWave frequencies. For example, WiGig can be used to manage handovers between mobile network and the home network and between the different rooms. On the other hand, LTE communication protocols can be also considered for providing a radio-light broadband indoor femto cell using mmWave physical layer modelling with a 6.67ms OFDM symbol duration, 200

 © IoRL consortium 2018

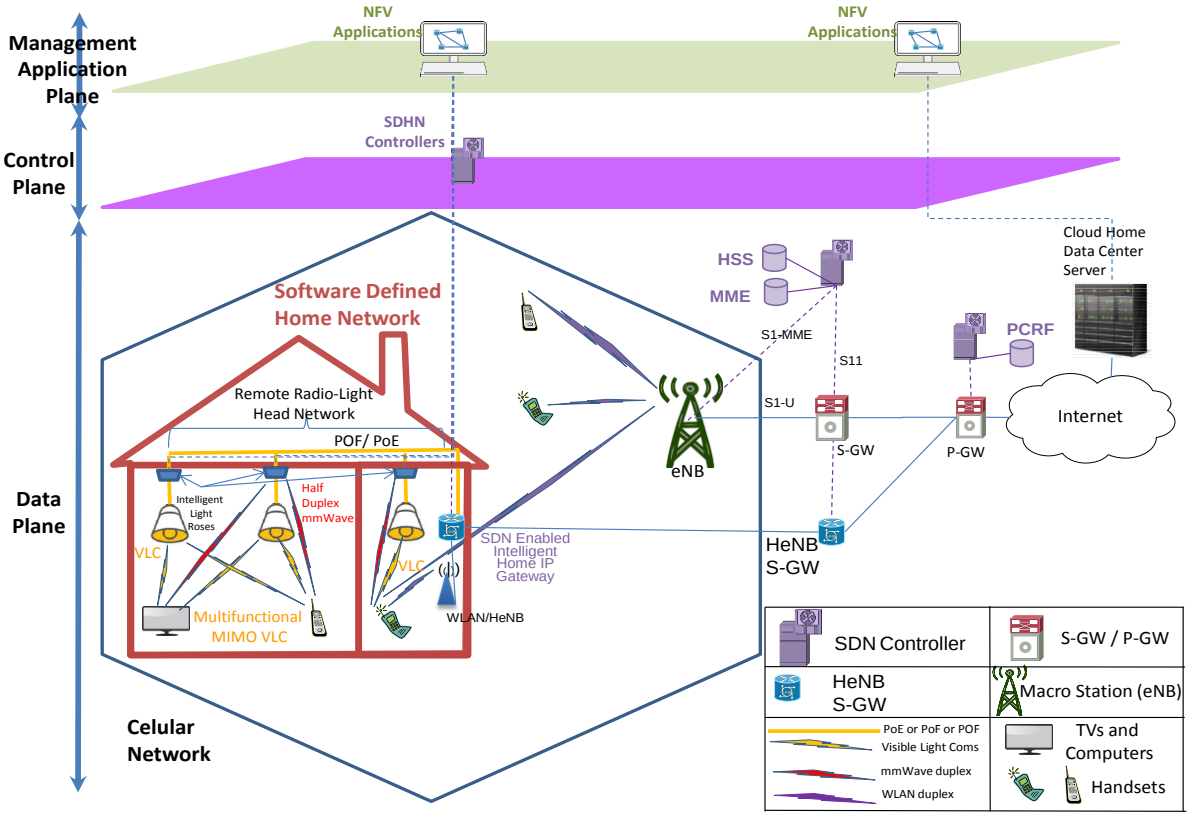


Figure 3-1 - Illustration of the SDN/NFV-based IoRl home network architecture

MHz channel bandwidth per component carrier and bit rate of 0.96 Gbits/sec under 64-QAM modulation. In addition, handover procedures based on 3GPP LTE specification consisting of hand-in, hand-out and inter-femto handovers, can be applied for handover between radio-light broadband indoor femto cells in each room of a building and combined with 3GPP 5G approach to support broadband, IOT and ultra-reliable and low-latency communications (URLLC) with real support of networking, NFV etc.

Since RRLH home communications does not interfere with the main transmitted mobile signal, their deployment will no longer require the permission of MNOs, so MNOs will be able to quickly provide indoor mobile service to all homes and businesses as opposed to only their larger business clients as is currently the case.

The following sub-sections study the interconnection between the SDN/NFV-based VNF applications that will be deployed in the IoRL home network by describing their structural service mechanisms, descriptors and lifecycle management. Our purpose is to provide insights on the appropriate NFVI resources needed to support services requested by users by examining how effectively VNFs interact to each other, which can be useful for refining the identified VNF interactions towards improving the IoRL virtual network. For the sake of clarification, at the end of this section we provide an updated version of Figure 3-2 to include all the identified mechanisms of the considered IoRL-specific VNFs.

3.1.1 **VNF deployment and configuration in the IoRL home network**

The combination, implementation and evaluation of the identified perspectives rely on a process to deploy and configure the identified VNF in the IoRL network. Such process includes

 © IoRL consortium 2018

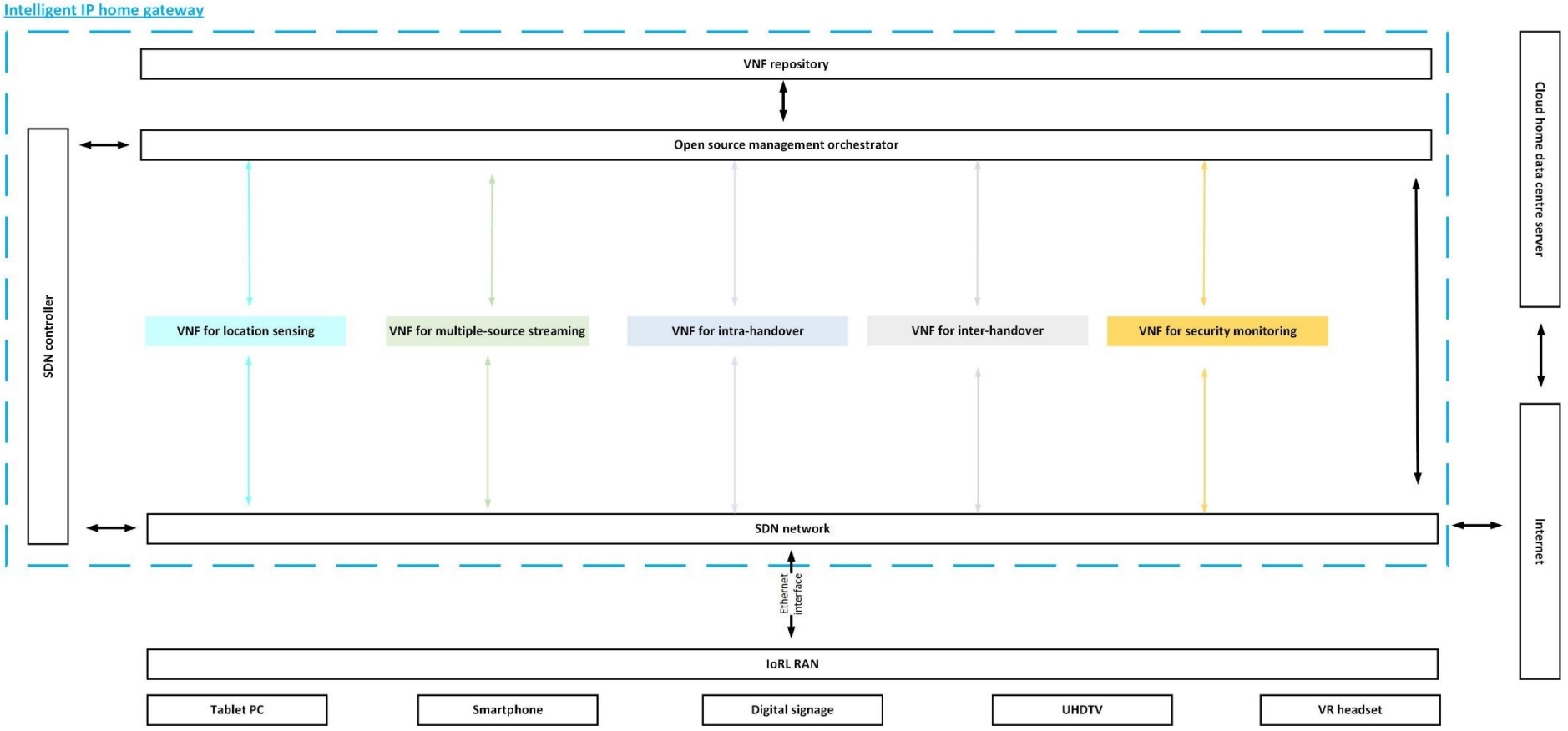


Figure 3-2 - Illustration of the IoRL Home Network with deployment of the identified VNF modellings in conjunction with SDN/NFV recent findings

© IoRL consortium 2018

the management of plane applications to adapt the operation of the intelligent HIPG and NFV servers, and the development of the use of meter, flow and group tables data to tailor a range of resource management services for the needs of network users.

To initiate such VNF configuration/identification process the client should notify the orchestrator to instantiate and deploy a registered VNF. This can be done by developing a VNF selection module which allows each IoRL user to select one of the available VNFs and decide when to start using it. Through the VNF selection module, the specific parameters of the service are configured and passed to the orchestrator, along with the VNF to instantiate. As result of the instantiation process, the orchestrator returns the IDs of the newly instantiated VNFs.

A successful VNF Instance creation returns the unique: nsi_id (note the 'i' for 'instance'), generated by the orchestrator, which can be used by IoRL for future requests. A successfully created VNF instance is left in the 'Started' state. It is assumed that only VNF instances in the 'Stopped' state can be deleted.

New VNF instances can be created based on most recent VNF registered version. Since the VNF instance is a resource created within the orchestrator, this creation process returns a VNF instance id that has to be used in further interactions with the orchestrator with respect to a specific VNF instance.

Similar to deleting a VNF (above), deleting a running VNF instance might either:

- Fail if the VNF Instance is still running;
- Imply immediately stopping the VNF instance then be deleted;
- Be tagged as 'to-be-deleted' and wait for the VNF to be stopped and then be deleted

Possible states of a VNF instance are:

- Requested
- Started
- Stopped

The VNF instance is in the requested state, while it is being provisioned in the VIM. It is expected that the newly and successfully instantiated VNF starts its lifecycle in the 'started' state.

Whenever, an IoRL user needs to change any of its parameters in current VNF configuration (e.g. the connection points, etc.), the orchestrator is notified about this change in the configuration parameters for an already deployed VNF instance. The VNF is modified and the result is returned to the orchestrator.

Below we highlight the main VNF applications to be integrated in the IoRL home network.

3.1.2 **Introduction to VNF structure for inter- and intra-handover**

The handover VNF is to support mobile services not only over 3GPP access but also over non-3GPP defined radio access. To effectively implement the handover between IoRL access and 3GPP access, a SDN-based network architecture is designed, which cooperates 5G core network architecture that decouples data forwarding function and control function, as shown in Figure 3-3. Form the figure we see the IoRL RAN as the access network for IoRL. Note the SDN controller, which is the network's core component and hosts two lower Layer 1 processors. The controller first generates an IF signal to drive up to 8 VLC MISO modules, and

 © IoRL consortium 2018

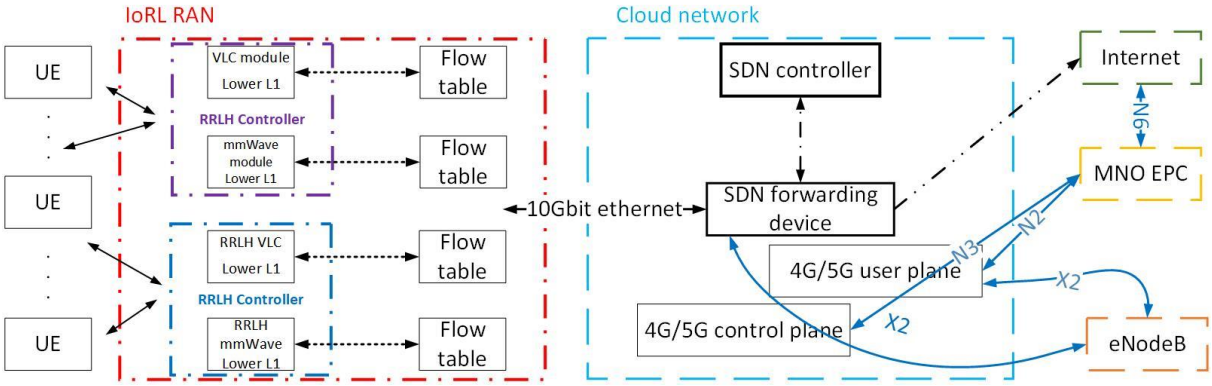


Figure 3-3 - Illustration of the IoRL SDN-based network architecture

then it generates an IF signal to drive up to 8 mmWave RF duplex modules. The IoRL RRLH controllers are interconnected by 10G ethernet ring with common public radio interface. Furthermore, intelligent home IP gateway is responsible for routing IP packets to 5G L2/L3 processor core or Internet. It contains SDN forwarding devices, the SDN controller and several VNFs. The SDN controller is in charge of computing forwarding paths, generating forwarding rules, and installing them into SDN forwarding devices. An SDN forwarding device is used to route IP packets according to the forwarding rules. Also, VNFs are used to offload the complexity of upper layer protocol processing of the communication systems in RRLH onto the intelligent gateway, including network functions such as location, security and transcoding.

Especially, to cooperate with 4G/5G technology, 4G/5G user plane VNF and 4G/5G control plane VNF are introduced.

- 4G/5G user plane VNF: it is used to route IP packets between IoRL access and 4G/5G access during the handover.
- 4G/5G control plane VNF: it can be used to manage the UE mobility such as handover decision.

Meanwhile, it introduces the following reference points:

- N2: Interface between 4G/5G control plane VNF and the user plane in mobile network operator EPC.
- N3: Interface between 4G/5G user plane VNF and the user plane in mobile network operator EPC.
- N6: Interface between the user plane in mobile network operator EPC and data network like Internet.
- X2: Interface between eNodeB and 4G/5G user plane VNF or 4G/5G control plane VNF.

3.1.3 Modelling of VNF application for intra-handover for non 5G RANs

In IoRL project Intra-building handover is performed by the 5G Layer 2, which is described in deliverable 5.1 “Technical Specification of Radio-Light Receiver”. Intra-handover mechanism is applied for UEs changing location from one RRLH/room to another within the same scenario in the special case when non 5G RANs are used and there is no Layer 2 RRC to perform the handover. In this case intra building handover is required to be performed at network layer 3, where the mechanism involves UE application interacting with SDN application via Ryu

 © IoRL consortium 2018

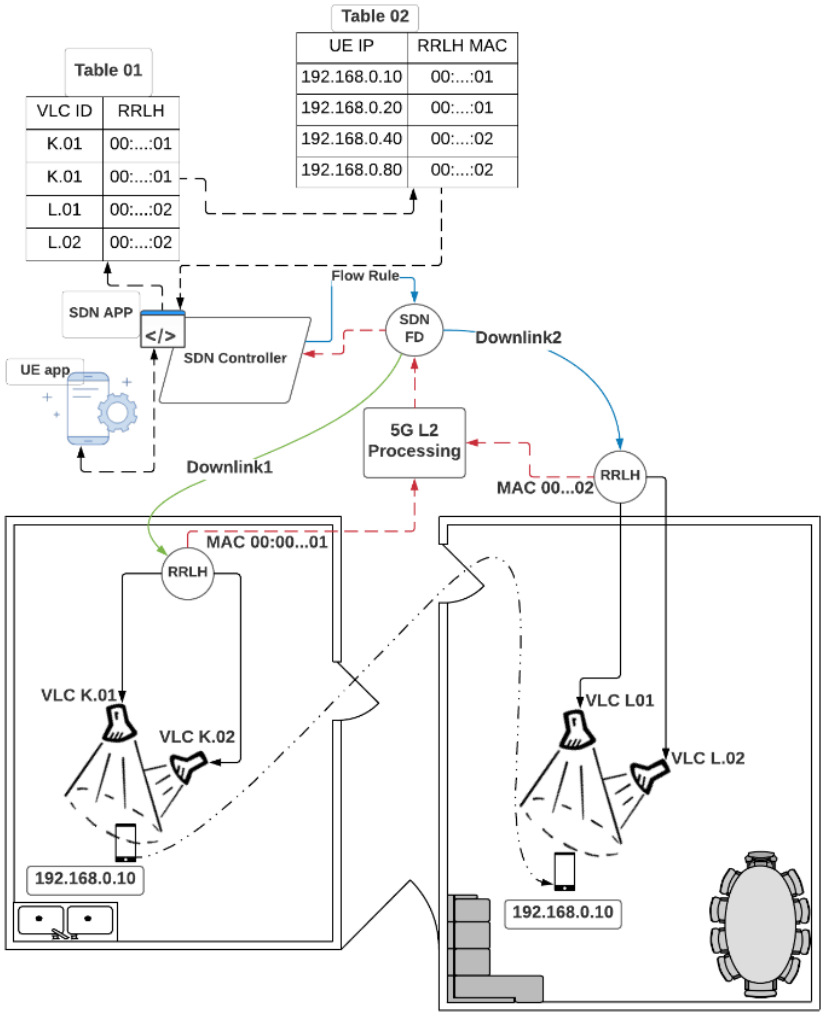


Figure 3-4 - Illustration of the intra-handover scenario in the IoRL home network

controller. This section presents the considered mechanism by adopting a scenario, when UE1 (IP 192.168.0.10) changes location from kitchen to the living room as shown in Figure 3-4. For example, let us consider that UE1 is situated in the kitchen and sends his VLC ID K.01 constantly to update the user application data base with his serving VLC ID. Therefore, its downlink data stream will always get to its updated location, where it receives downlink1 via RRLH 00:00:00:00:00:01/VLC K.01. Then UE1 changes its location to the living room and sends an update of the serving VLC ID (L.01). The SDN application will be notified of the change of UE1 location then access the user application database and retrieve the current VLC ID. Then, the SDN application uses the information illustrated in Table 01 and Table 02 of Figure 3-4. (which are included in the SDN application), to augment the previous flow rule for this destination. It does so by (i) adding the UE IP address, (ii) pushing the flow rule to OVS through the controller and (iii) updating downlink path (dowinlink2) for RRLH MAC 00:00:00:00:00:02/VLC L.2. The SDN application augments the flows frequently by updating UE IP addresses list in flow rules, based on the two-step mapping process, VLC ID - RRLH MAC, and RRLH MAC - UE IP address. For the downlink packets destined for specific UE, the switch sends the flow to the appropriate RRLH that covers that UE. The updating procedure is illustrated in Figure 3-5.

 © IoRL consortium 2018

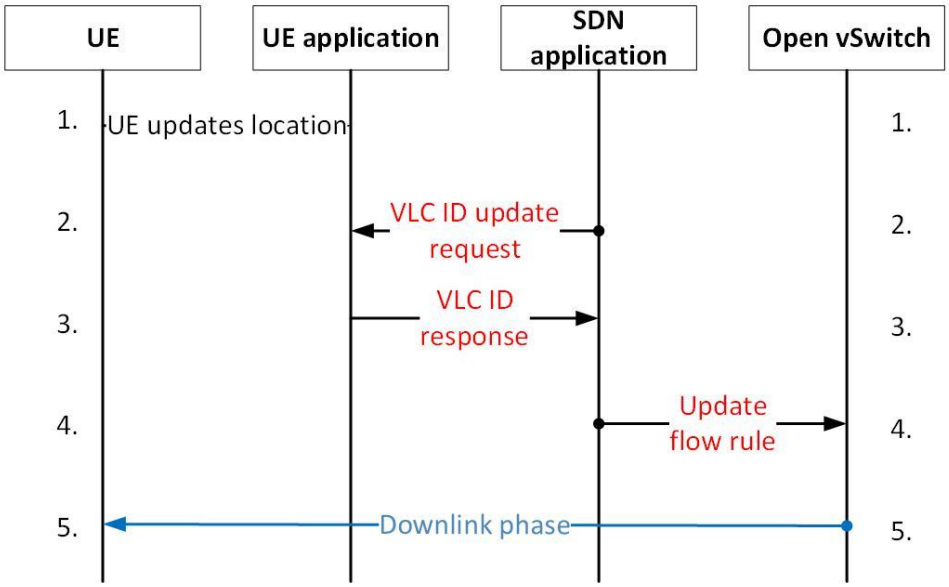


Figure 3-5 - Illustration of the intra-handover procedure in the IoRL home network

The uplink traffic is simply exploited to store the MAC header of the RRLH in the database to forward the downlink traffic to the RRLH covers the room within which the UE resides. The RRLH will flood the packets to all 8 mmWave modules and VLC modules to get the destined UE. To insure a valid destination RRLH, we develop a SDN application to send frequent uplink traffic from each UE within our coverage area to extract the RRLH MAC address and update a database table with UE-IP/RRLH-MAC information. Note that the forwarding rule for each UE has idle time-out and hard-timeout and therefore, if the SDN application will not receive an updated MAC address within the Idle-timeout period, the controller will forward the downlink traffic to all RRLH controllers to ensure the downlink transmissions to all possible locations within the coverage area. In case the hard-timeout timer expires without any MAC update, then the UE will be considered out of the coverage area and deletes its entries.

3.1.3.1 Modelling of VNF application for inter-handover

Inter-handover mechanism refers to UEs which change between the gNodeB (inside the IoRL home network) and the e/gnodeB (outside the IoRL home network). Inter-handover process integrates by switching the routing of packets from the Internet MAC address to the service gateway MAC address, and vice versa. The main steps to describe the architectural structure of the inter-handover process are given as follows.

- The PGW is responsible for the allocation of the UE local IP address and the advertisement of the UE public IP address to the Internet.
- Data traffic is forwarded by using tunnels between the UE and the PGW.
- The internet sees UE traffic coming from the public IP address advertised by the UE's PGW.
- The Internet has no knowledge for UE's movement and always uses the PGW address to reach the UE.
- The UE, eNodeB, MMEs, SGWs and possibly the PGW exchange signaling messages to modify the tunnel to redirect the traffic to the UE location, as shown in Figure 3-6.

 © IoRL consortium 2018

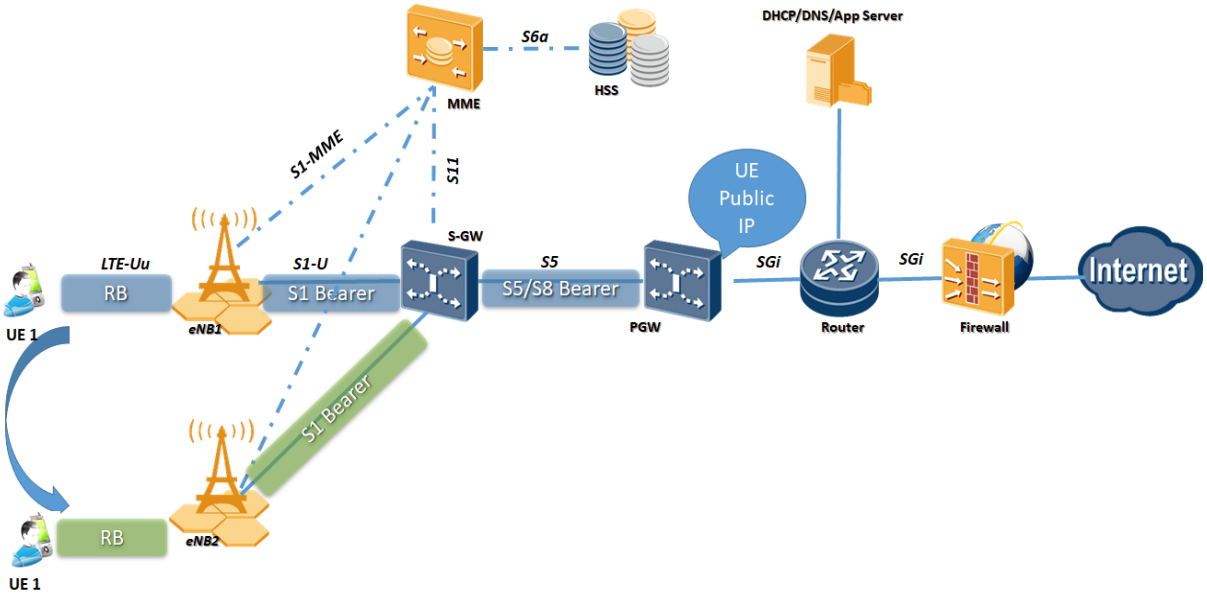


Figure 3-6 - Illustration of LTE based inter-handover mechanism

In the IoRL network, data traffic is forwarded using the SDN sub-network directly through the local interface to the Internet, after changing the source IP address to allow the Internet to reach the UE. Therefore, to design a network that contains LTE and IoRL deployment several issues and obstacles need to be addressed. This includes:

- IoRL network does not use bearers, while LTE uses bearers
- UEs served by the IoRL can access the Internet through the IPHGW public IP address and not via the 4G/5G PGW public IP address

To address these obstacles, three key architectures are considered, namely: standalone mode and two types of inter-handover scenarios (1) integrated with mobile network, (2) integrated with the mobile network through cloud gateway.

To realise each architecture separately, let us firstly focus on the basic structure of our system shown in Figure 3-7, where the RAN is the access network for IoRL. The core components of such system are the RRLH controllers, where each one of them hosts two lower layer (i.e. L1) processors and aims to generate (i) an IF signal that drives up to 8 VLC MISO modules, and (ii) another IF signal to drive up to 8 mmWave RF duplex modules. The RRLH controllers in IoRL are interconnected through a 10Gbit ethernet ring with common public radio interface. Furthermore, the intelligent HIPG is responsible for routing IP packets to higher layers' (i.e. L2/L3) processor core or the Internet, and consists of SDN FDs, the SDN controller and several VNFs. Note the SDN controller, which is in charge of computing forwarding paths, generating forwarding rules, and installing them into SDN FD that in turn, can route IP packets according to forwarding rules. The VNFs are used to offload the complexity of upper layer protocol processing of the communication systems in RRLH onto the intelligent HIPG, including network functions such as location sensing, security monitoring and transcoding. In conclusion, the IoRL system uses three different operation modes with slightly different structures (i.e. RRLH → HIPG → VNFs) that can be flexibly orchestrated through SDN/NFV deployment by means that they can either added or removed from the system at any time instance.

 © IoRL consortium 2018

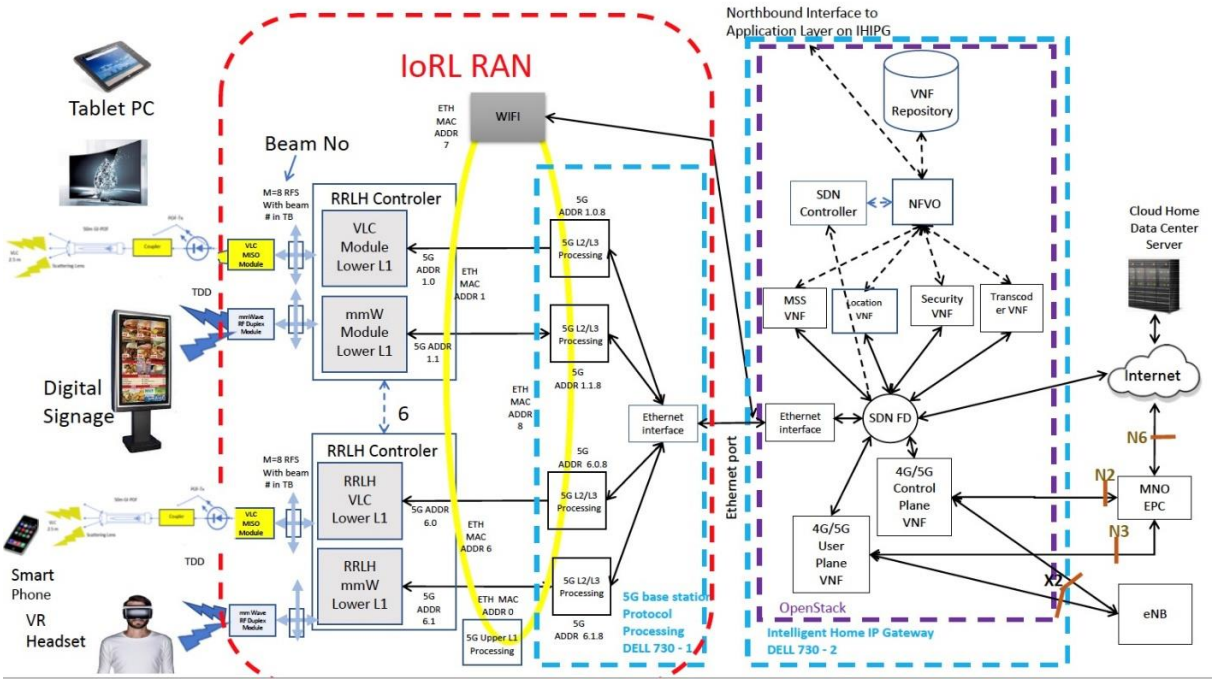


Figure 3-7 - The basic structure of IoRL system

Having realised the basic structure of the IoRL system, we can now proceed realising each of the considered architecture separately.

3.1.3.2 Architecture of the inter-handover standalone mode

This mode the IoRL home network can be realised by observing Figure 3-7; from the figure we see that the IoRL network operates independently from the mobile core network, e.g., there is no connection between the two networks. Therefore, in standalone mode, (i) 5G/4G control and data plane VNFs are not included, while (ii) the IoRL network is responsible for the IP address allocation and the air interface security, which is provided by specialised VNFs.

Intra-handover process between different lights can be supported in standalone mode, and seamlessly and efficiently handled by the Layer 2 processor. On the other hand, inter-handover process cannot be supported because there is no link between the IoRL and the core network. Instead software can be used at the UE side to prioritise the IoRL links over the 4G connections. This way, UE data traffic can be always handled by the IoRL, whenever IoRL coverage is available. Also, in standalone mode IoRL network can forward user requests to either the local services or the Internet. For example, in the museum scenario, the IoRL network can use the user location to intelligently provide to each user information about exhibits these users currently observe. At the same time, UEs can have high speed connection to the Internet through the HIPGW without consuming their subscription data bundle.

3.1.3.3 Architecture of the inter-handover process integrated with mobile network

In this mode the IoRL deployed as part of the MNO with tight integration with the MNO core network, as shown in Figure 3-7. It means, UE authentication, air interface security parameters and IP address allocation are provided by the MNO's EPC. In this mode, the IoRL has specialised VNFs known as 5G/4G control plane VNF and 5G/4G data plane VNF. The 5G/4G control plane VNF is responsible for the transportation of the control signaling messages between the UE and the MNO's EPC, while the 5G/4G data plane VNF is responsible

 © IoRL consortium 2018

for the data traffic forwarding over GTP tunnels. Given that SDN/NFV deployment consists major part of IoRL device design, the 5G/4G control plane VNF can work with the SDN controller to build a map between the GTP TEID and VLAN tags to provide traffic separation and differentiation.

On this basis, two types of inter-handover processes can be considered. The first type is the handover from IoRL access to 4G/5G access, and the second type is the handover from 4G/5G access to IoRL access. Below we provide a detailed description on how to perform the SDN-based handover in the considered two handover scenarios. Recall that the 4G/5G control plane VNF is responsible for the setup and maintenance of the GTP tunnel with the MNO's EPC while the 4G/5G data plane VNF for GTP encapsulation and decapsulation utilising information provided by the 4G/5G control plane VNF. Communication messages between the 4G/5G control plane VNF and the SDN controller are tagged with specific VLAN (control VLAN tag). The SDN-FD is pre-configured to send the traffic tagged with the control VLAN tag to the controller.

• IoRL to 4G/5G inter-handover process

1. The UE uses IoRL access system and IP packets are routed to IoRL RAN or Internet according to the forwarding rules in SDN-FD. The forwarding rules are configured by SDN controller;
2. Based on the measurement reports sent by the UE, the RRC makes the handover decision, the request is sent to SDN-FD through IoRL RAN. The SDN-FD forwards the handover request to the 4G/5G control plane VNF;
3. 4G/5G control plane VNF, notifies the SDN controller about the handover procedure to modify the flow rules related to the UE in question. The controller updates the flow rules to send the data traffic to the 4G/5G data plane VNF to be buffered;
4. 4G/5G control plane VNF, passes the handover request to the MNO's EPC, the EPC exchanges control signaling messages with the eNodeB to establish the radio bearer between UE and the eNodeB, and also establishes a communication link between the eNodeB and the EPC;
5. Upon receiving handover response from the EPC, the data related to UE can be forwarded between 4G/5G user plane VNF and EPC via data plane link. At the same time the SDN controller is notified to remove the flow rule entries;
6. The SDN controller generates flow delete message to delete the forwarding rules in SDN-FD. These rules are used for data forwarding related to UE.

• 4G/5G to IoRL access inter-handover process

1. The UE is connected with 4G/5G access system. The uplink data are routed from eNodeB to Internet through MNO EPC. The downlink data are transmitted along the reverse direction;
2. Based on the measurement report sent by the UE the eNodeB makes handover decision and forwards handover request to the EPC. The latter sends the handover request to 4G/5G control plane VNF;
3. Upon receiving the Handover Request, the 4G/5G control plane VNF notifies the SDN controller using the control signaling VLAN tag;
4. After receiving the message, the SDN controller installs the forwarding rules in SDN-FD by sending flow-mod message. The rules include routing the downlink and uplink data related to UE to IoRL RAN and Internet respectively. Meanwhile, the rules also match

 © IoRL consortium 2018

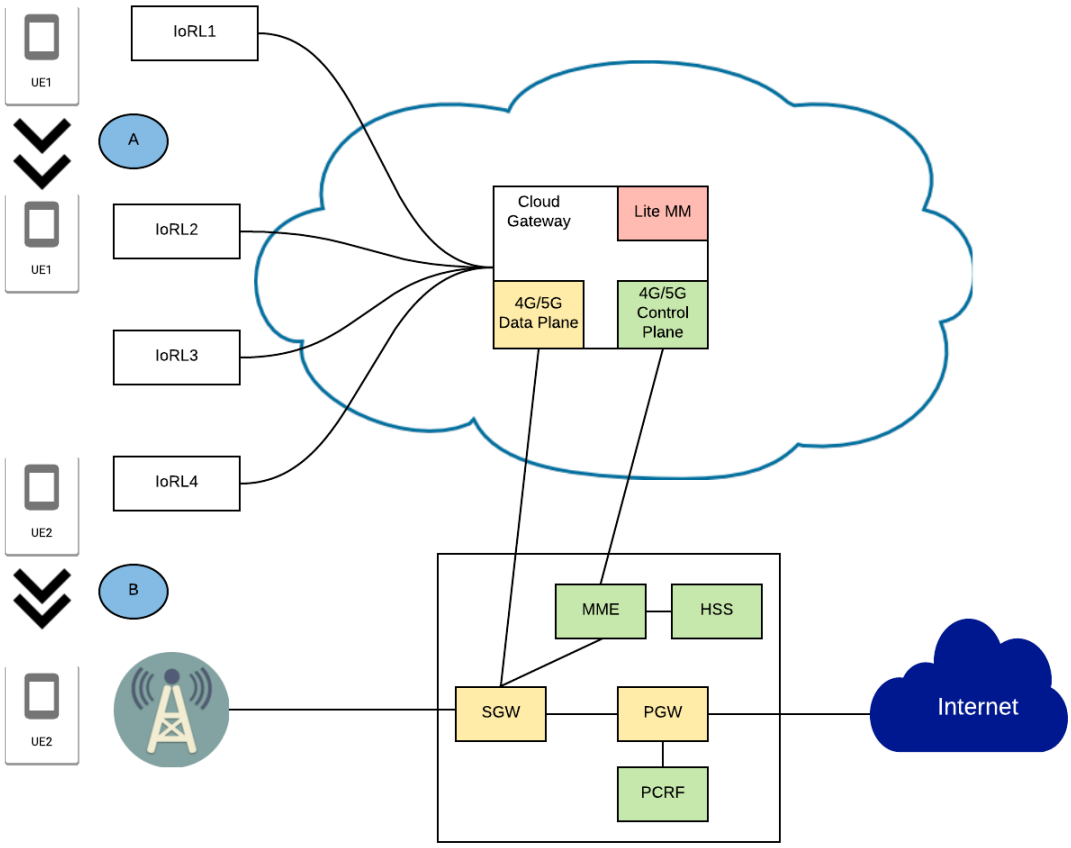


Figure 3-8 - Integrated with the mobile network through cloud gateway

the data received from 4G/5G user plane VNF. During the handover, some data may forward from EPC to 4G/5G data plane VNF;

5. Then, the SDN controller responds to the 4G/5G control plane VNF, and upon receiving the message, the 4G/5G control plane VNF sends Handover Response message to EPC;
6. After receiving the Handover Response message, the EPC sends handover command to the eNodeB which forwards the command to the UE to reconnect to the IoRL. At the same time, control signaling is exchanged between the 4G/5G control plane VNF and the EPC to setup the data bearer between the IoRL and the EPC;
7. Then, the EPC asks the eNodeB to release the network resource that includes radio bearer between the eNodeB and the UE, and network resource between the eNodeB and the EPC. The handover procedure has been completed.

3.1.3.4 Architecture of the inter-handover process integrated with the mobile network through cloud gateway

In this mode multiple IoRL RRLHs are connected to a cloud gateway, while the cloud gateway is connected to the mobile network operator. This means there is no direct connection between the IoRL network and the MNO's EPC. The cloud gateway includes light mobility management element to handle the handover between multiple IoRL and at the same time works with the MNO's EPC to perform inter-RAT handover between the IoRL and the MNO's access network. Also, the cloud gateway is responsible for maintaining the UE bearers and redirects them the MNO during the handover.

 © IoRL consortium 2018

Note that by considering such setup, the MNO's EPC can realise the cloud gateway as another eNodeB as shown in Figure 3-8. Also, the handover procedures explained in section 3.1.4.2 can be considered here with only differences that (i) the cloud gateway is the point of interaction with the MNO's EPC instead of the IoRL, and (ii) the functionality of the 4G/5G control and data plane VNFs is included as part of the cloud gateway.

From Figure 3-8 we observe two cases, namely Case A (upper side of the figure) and Case B (bottom side of the figure). Case A will help reducing the control signalling between the access and the core network be letting UE 1 to be handled over from IoRL1 to IoRL2, whereas IoRL 2 can be handled locally through the cloud gateway and the IoRL SDN controllers without any help from the MNO's EPC. In Case B, UE 2 will be handled over from IoRL 4 to the 4G/5G access by performing procedure similar to this in section 3.1.4.2. The difference is that since EPC can only see the cloud gateway (not the whole network as previously), the process in Case 2 focuses on the interaction between the cloud gateway and the MNO's EPC as highlighted below.

1. Based on the reports form UE measurements, the IoRL network takes the handover decision and the RRC sends handover request message through the SDN-FD to the cloud gateway by utilising the control plane VLAN tag;
2. The cloud gateway keeps the UE data bearer's information and upon receiving the handover request message, it identifies that the handover to the MNO's access network. In that way, the request is forwarded to the MNO's EPC to be forwarded to the eNodeB in question;
3. All the communication continues between the cloud gateway and the MNO's EPC, while only the handover command can be sent to the UE to change its attachment points and control the signalling to the SDN controller, which is commonly used to remove the flow rules from the SDN-FD.

After the exchange of control signalling between the involved entities, the inter-handover command can be sent to the UE to reconnect to the MNO's eNodeB, while the SGW will be able to redirect the GTP tunnels to the eNodeB.

3.1.4 **Modelling of VNF application for security monitoring**

Security monitoring VNF service is responsible to detect various types of network attacks, scanning activity and denial of service (DoS). Its functionality is implemented as an integrated security framework (ISF). The implementation of IoRL's security monitoring takes place in a single Linux-based dedicated VM, which comprises of three network interfaces as listed below.

- Interface no. 1 - is connected to network available for all IoRL RAN users;
- Interface no. 2 - is connected to the SDN management network, and allows access to the SDN controller;
- Interface no. 3 - is connected to dedicated switch port analyzer (SPAN) port in Open vSwitch, managed by SDN application.

Moreover, the security monitoring VNF include three main elements:

- Attack detection - is performed by a module to implement the main logic of the detection mechanism by taking as input data from SPAN interface;

 © IoRL consortium 2018

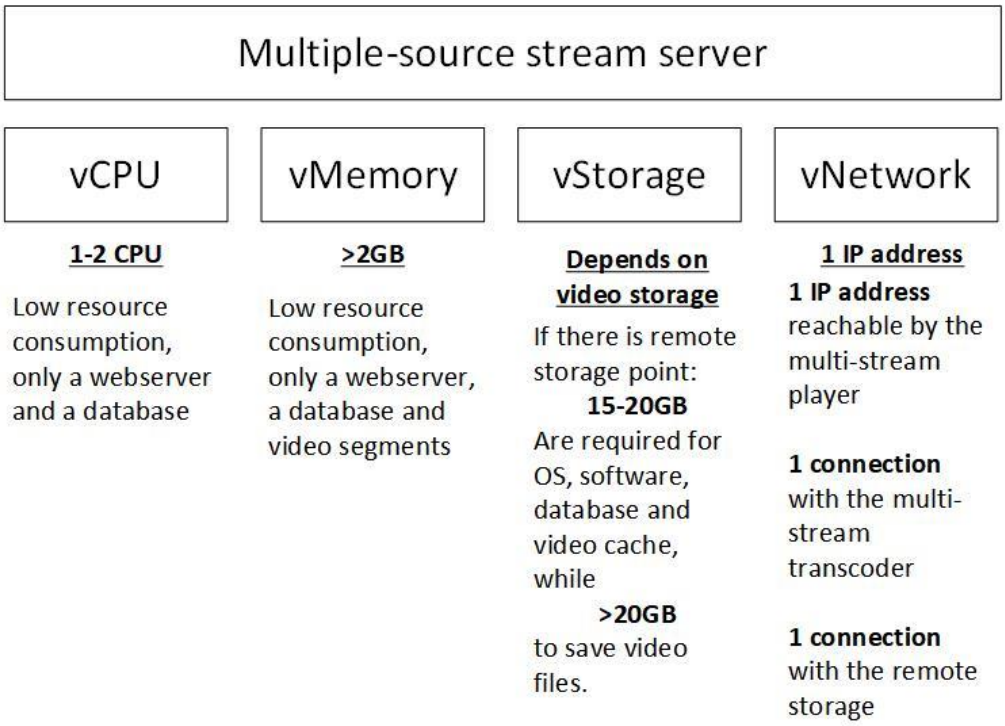


Figure 3-9 - Illustration of multiple-source server with functional requirements

- SDN security monitoring and application management - is performed via an SDN application, with main role to manage the access to “interesting data” received from and directed to the RAN. The optimal compunction method between security SDN application and the SDN controller is a RESTful API;
- web-based security dashboard - is to allow management of the whole ISF functionality by users with appropriate access rights to IoRL system management. Additionally, authorised users can use such dashboard to access logs that concern current security state of the IoRL system and its connected users.

3.1.5 **Modelling of VNF application for multiple-source streaming**

The IoRL home network can manage data streaming in two different ways.

- To stream the data from the server to the end user by using the IoRL network as black box that provides internet access. Such an approach allows our system to be compatible with actual streaming services over the internet, especially video communication tools.
- To stream the data to the end user using multiple-source streaming process, which allows streaming sub-flows of data from different sources, thereby increasing service reliability at the application level. These sub-flows can be read either independently to provide lower service quality or merged for higher service quality.

Multiple-source streaming over remote VLC head is composed of three main modules that rely on the HTTP/TCP protocol for communication:

- The multiple-source stream server, which is part of the server side and can be deployed in the Intelligent home IP gateway as virtual functions shown in Figure 3-9.

 © IoRL consortium 2018

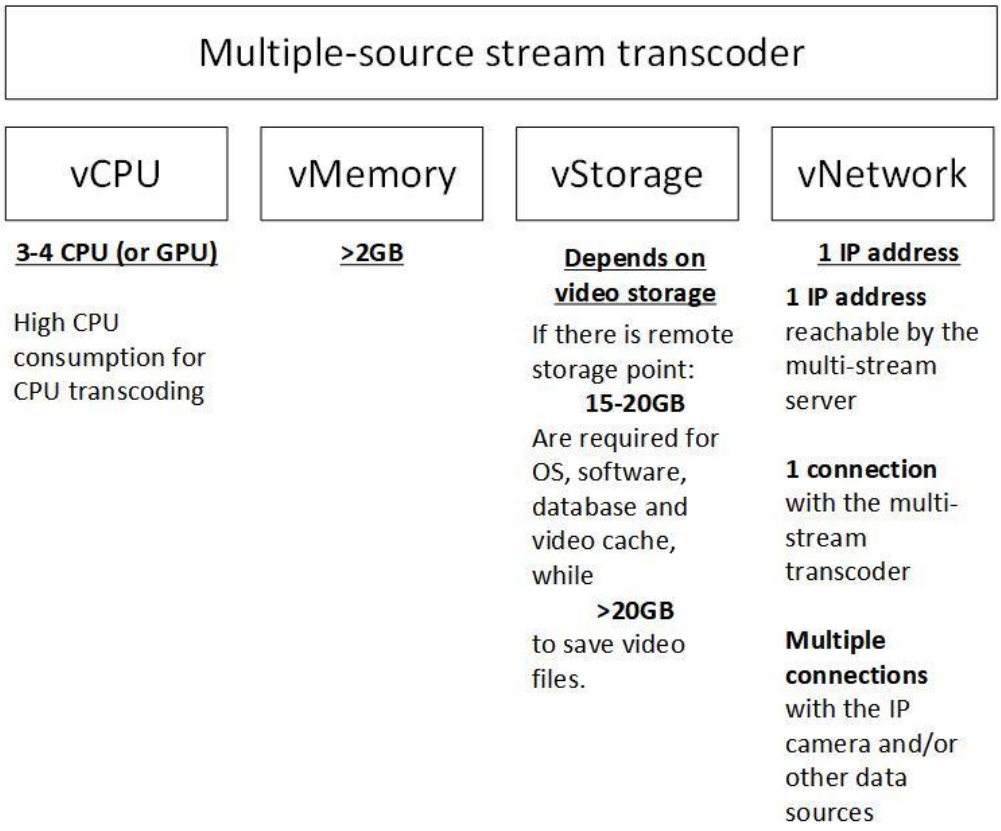


Figure 3-10 - Illustration of multiple-source transcoder with functional requirements

The multiple-source stream server is to be developed in Java with a tomcat server and an SQL database;

- The multiple-source stream player, which can be deployed similarly to the multiple-source stream server;
- The multiple-source stream transcoder, which can be deployed at the client side, as part of an application running in the user equipment shown in Figure 3-10. The multiple-source-stream transcoder is to be developed in NodeJS and includes an FFmpeg [180] tool for the transcoding.

On the client side, the multiple-source stream player can be seen as video player, which will be integrated in a web page and accessed through a web browser. The client may also be running in a native application for specific user equipment. As the algorithm of multiple-source stream is defined as client-centric, the multiple-source stream client is responsible for the creation of the HTTP requests sent to the multiple-source stream server through the RRLH network and the relay WLAN WiFi access for a multiple-source streaming session. The adaptation algorithm is implemented in the multiple-source stream client.

On the server side, the multiple-source stream server can be seen as an application that will answer client requests for specific video contents. This module will be responsible for the creation of video segments adapted to multiple-source streaming. The video segments are created from video data transcoded in numerous different qualities. For that purpose, the transcoder comes along with the server. Such a transcoder is a module that will transcode input video data into multiple-source-ready video data in one or several qualities. These data can then be pushed to the server to be available for the stream player at user equipment.

 © IoRL consortium 2018

The multiple-source streaming VNF application will be deployed in modules using qemu [181], while the VMs will have a linux OS running on embedded Docker [182] containers. The intended tests will include the following scenarios.

 © IoRL consortium 2018

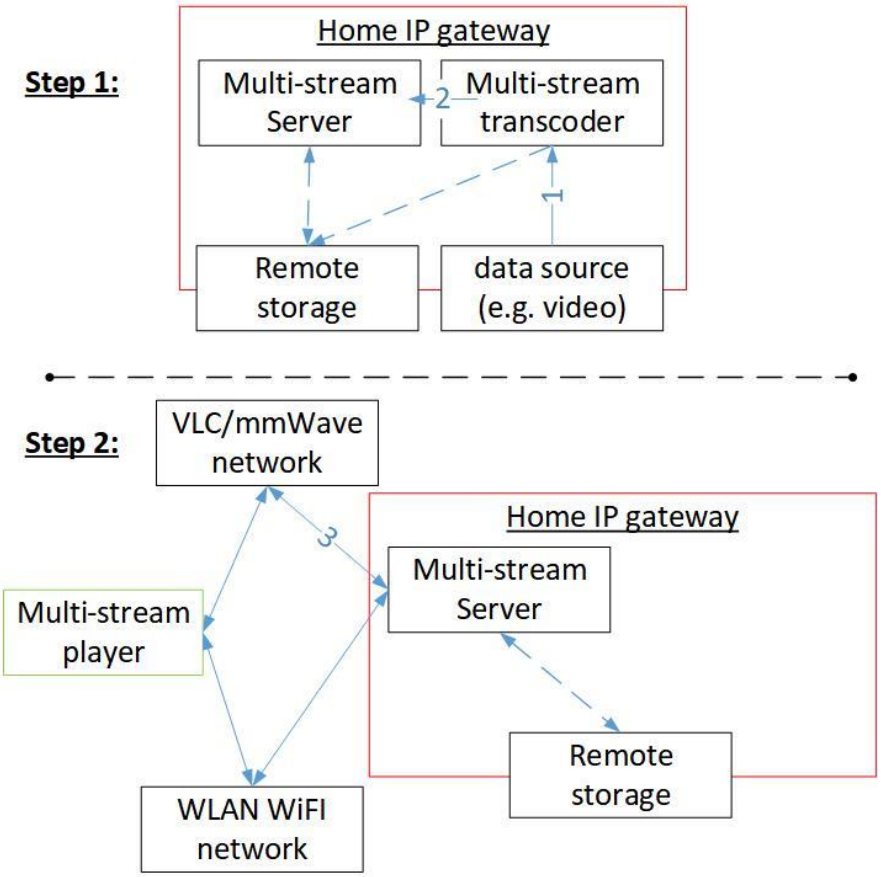


Figure 3-11 - Illustration of multiple-source streaming process under video-on-demand testing scenario

- End-to-end video-on-demand streaming service: in this scenario, the multi-streaming transcoder will be used to transcode the videos into multi-stream compatible content, while a user will be able to watch a video on its equipment by sending HTTP requests to the stream server through different network access, e.g., VLC, WLAN WiFi,

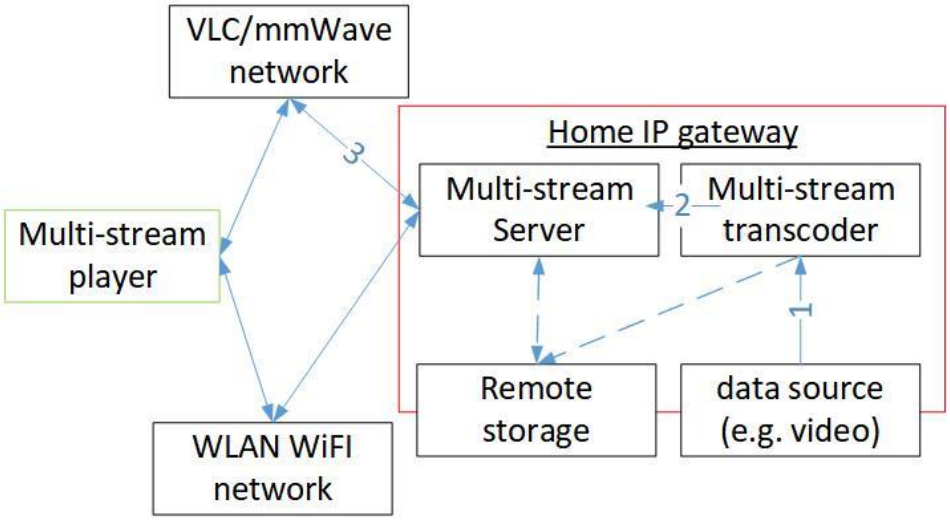


Figure 3-12 - Illustration of multiple-source streaming process under live streaming testing scenario

 © IoRL consortium 2018

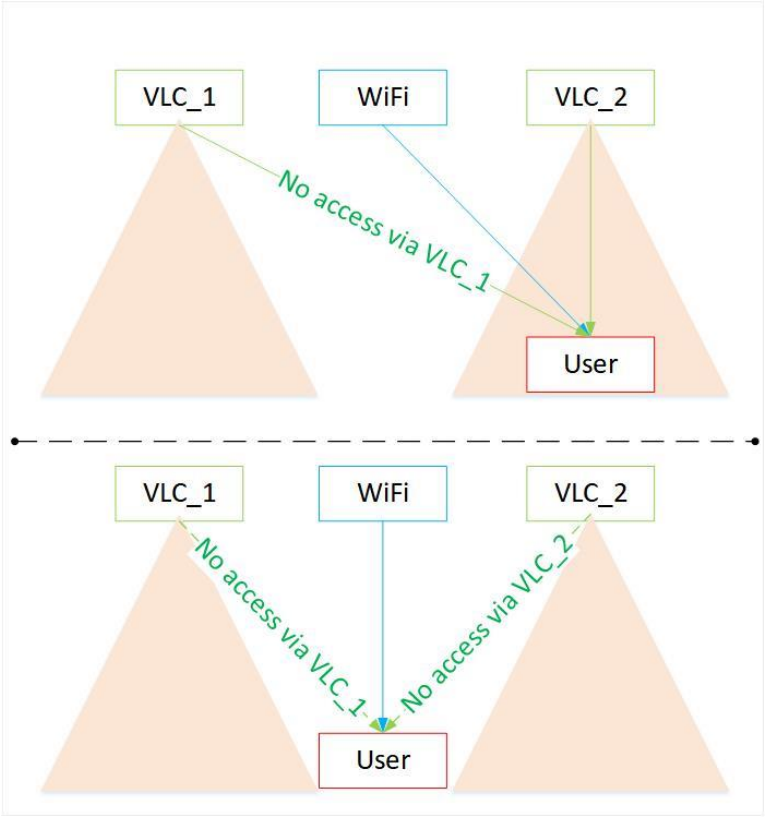


Figure 3-13 - Illustration of multiple-source streaming process under video streaming for a moving user testing scenario

mmWave. An illustration of such testing scenario is shown in Figure 3-11;

- Live streaming service: in the scenario the transcoder will be used to transcode the video stream of IP camera into multiple-stream-compatible live content and save them into the video storage. A user will be able to watch a video live by sending HTTP requests to the multiple-source streaming server through different network access as shown in Figure 3-12.
- Application-level reliability, with fall-back on alternative WLAN WiFi network in case VLC/mmWave network is unavailable: this testing scenario will demonstrate the application level reliability of multiple-source streaming application considering the case where a VLC access is unavailable due to obstacles between VLC modules and users, while WLAN WiFi network is only available for accessing the network, as shown in Figure 3-13.

3.1.6 **Modelling of VNF application for location sensing**

The main goal of VNF application for location sensing in the IoRL home network is to support the indoor location-based data access, monitoring and guiding. Its modelling includes a location server, location service client, RRLH controller and location database described below.

- Location database is a VNF implemented at IHIPG to store the measured VLC and wwWave parameters and provide location estimates with location assistance data, e.g., coordinates of mmWave antennas and LEDs lights;

 © IoRL consortium 2018

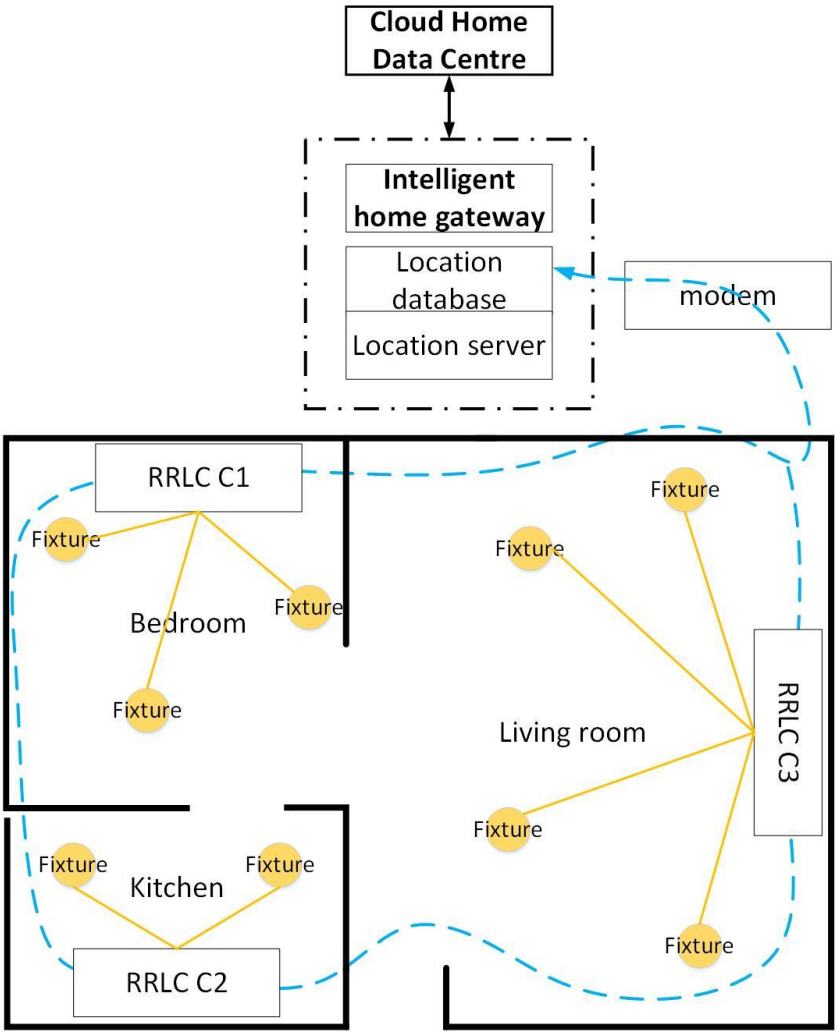


Figure 3-14 - Illustration of main components of the VNF for position sensing

- Location server is a VNF implemented at intelligent HIPG to estimate location coordinates of all connected UEs in the IoRL network. RRLHC and UEs measure location relevant parameters that are used by location server in location estimation process;
- RRLH controller is a VNF to collect new sets of mmWave parameters and report them back to main SDN controller of the network using PDCP packets;
- UE is a VNF to collect new sets of VLC parameters and report them to location database using IP packets;
- Location service client is a switch application situated at either the UE or cloud home data control server. It requires location information for location-based data access, for monitoring and guiding and for interactive applications.

The main components of the position sensing architecture are summarised in Figure 3-14.

More particularly, location database is a MySQL database, which stores three different sets of parameters:

- Location relevant mmWave and VLC parameters of all connected UEs that are measured by RRLHC and UE;
- Estimated location coordinates of all connected UEs;

 © IoRL consortium 2018

- Coordinates of all mmWave antennas and LEDs and eventually also floor plans.

The location relevant signal parameters are measured by RRLH controller and by UE. RRLH controller is measuring mmW pseudo time-of-arrival in the uplink direction. UE is in charge of VLC received signal strength which is measured in the downlink direction. Both VLC received signal strength and the wwWave pseudo time-of-arrival measures are then uploaded to location database to be processed by location server towards estimating the UE location coordinates. Each entry of the location relevant that parameterises the database consists of the UE ID, RRLHC ID, estimated mmWave/VLC parameters and a timestamp based on which the parameters are estimated. The reason to store pseudo time-of-arrival instead of time-difference-of-arrival measurements (as done conventionally) is to create a unique relationship between antenna coordinates. Also, time-difference-of-arrival entries are more complex as they need to additionally indicate which of the antennas were used to provide the estimates. Instead, the proposed pseudo time-of-arrival approach is more robust as it is not necessary to assess the relation between the antenna coordinates.

Location server is represented by a VNF implemented at IHIPG to compute position estimates of all UEs that are connected to IoRL home network. The computation is based on estimated location relevant signal parameters, coordinates of mmWave antennas and LED lights situated within RRLHs and eventually floor plans of the environment to enhance the precision as shown in Figure 3-13. Location server includes three main algorithms to compute the location estimation. The first algorithm estimates location coordinates based on VLC received signal strength measurements. The second algorithm estimates location coordinates based on mmWave pseudo time-of-arrival measurements. The third algorithm represents a set of data fusion and tracking algorithms which fuse information obtained from VLC and mmWave sensors and exploit additional assistance data such as ground plans etc.

Localisation procedure which involves interaction between RRLH controller and the UE is enabled by means of the IoRL positioning protocol. Such protocol is an adaptation of the LTE positioning protocol with its main function to initiate the localisation procedure in the IoRL network as follows.

- For mmWave estimations
  - RRLH controller provides UE assistance data (the base sequences, their group, sequence number, cyclic shift, frequency hopping scheme, etc.)
  - RRLH controller triggers the UE to issue a series of sounding reference signals for uplink time-of-arrival measurements;
- For VLC estimations
- UE triggers the RLHC to issue a series of reference VLC orthogonal frequency division multiplexing symbols for each RRLH in turn for downlink received signal strength measurements at the UE.

Note that the IoRL positioning protocol will not be elaborated for IoRL demonstrations. It is for the sake of holding the implementation of demos as simple as possible. Due to the fact that IORL demonstrator will contain only very small number of UE units, there is no need for protocol-based scheduling of the localization procedure - for providing the assistance data, planning the resources and for requesting UEs/RRLHs to initiate the measuring the localization procedure. In IoRL demos the UEs and RRLH controllers will be preconfigured. More details for the functionality of position sensing in the IoRL system are available in D5.1.

 © IoRL consortium 2018

3.1.7 **VNF descriptors**

VNF descriptor (VNFD) is "a deployment template which describes a VNF in terms of deployment and operational behavior requirements". As the VNFs are participating in the networking path, the VNFD also contains information on connectivity, interfaces and KPIs requirements. The latter is critical for the correct deployment of the VNF as it is used by the NFVO in order to establish appropriate Virtual Links within the NFV Infrastructure (NFVI) between VNF Component (VNFC) instances, or between a VNF instance and the endpoint interface to other Network Functions. This section attempts a specification of the VNFDs that we will use for the implementation of the cache node and cache controller VNFs. The described VNFD has been defined and is currently used by T-NOVA project. Assuming a VNFD for a VNF that is composed by more than one VDUs, the VNFD contains the following segments:

- The VNFD preamble, which provides the necessary information for the release, id, creation, provider etc.
- The virtual deployment unit (VDU) segment, which provides information about the required resources that will be utilised in order to instantiate the VNFC. The configuration of this part may be extremely detailed and complex depending on the platform specific options that are provided by the developer. However, it should be noted that the more specific are the requirements stated here the less portable the VNF might be, depending on NFVO policies and service-level agreement (SLA) specifications. It is assumed that each VDU describes effectively the resources required for the virtualisation of one VNFC. Fields within the VDU VNFD segment include:
    i) IT resources and platform related information;
    ii) The internal structure of the VNF and the connection points of the VDU (name id, type) and the virtual links where they are connected;
    iii) Monitoring parameters to be collected by this VNF, including system related information and VNF specific metrics;
    iv) Scaling parameters defining the thresholds for scaling in-out.
- VNF lifecycle events segment, which is related to lifecycle events, where the technology used for interfacing with the VNF is defined, as well as the appropriate commands allocated to each lifecycle event.

3.1.8 **VNF lifecycle**

Considering the virtualisation of the satellite hub elements and the ground segment functions, it is necessary to define hereby the Network Function lifecycle, which comprises eight stages, as depicted in Figure 3-15.

The development of the VNFs is performed by Function Developers, the role of which can be played by either the Service Providers themselves, or third-party entities, but also software houses, which will be capable of entering to the current HW-specific market, since many HW-based elements will be gradually softwarised.

Validation and debugging is also an important procedure, to ensure that the developed function works and performs as expected. Buggy function code can have severe impact on the stability of the service, affecting both the performance and the reliability of the service provider. Function code and VMs will be digitally signed to ensure authenticity and, in any case, each service provider reserves the right to disallow the deployment of certain third-

 © IoRL consortium 2018

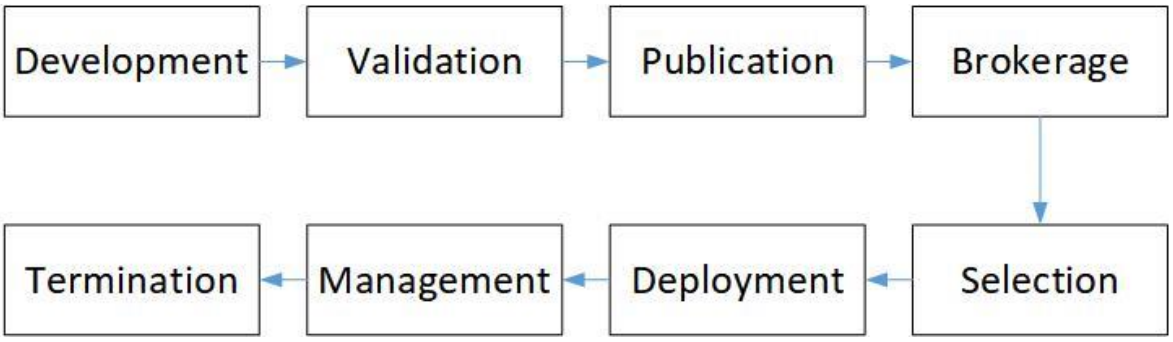


Figure 3-15 - Virtual network function lifecycle

party functions into the network infrastructure. Although network traffic analysed by the NFs will correspond to a portion of the total network traffic, as the Orchestrator and SDN functions will only forward specifically selected flows towards each VNF.

Publication of functions should be performed at various Function Stores, whose repository will host both the function image (as stand-alone application or integrated VM) and the associated description/metadata. Through the functions stores, the service providers will have the option to download the respective function in a similar way as mobile apps are disseminated. Alternatively, the SDN/NFV-enabled service provider may limit the installation of function to only internal ones, without extending its installation to external sources.

In case that the external installation of a function is allowed by the business model, then Function Brokerage is undertaken by the brokerage platform, which will be able to match user "high-level" service requirements with the specific technical specifications of the NFs, ensuring that the resources required for VNF deployment are available.

Upon VNF S*election* by the user, the Deployment phase includes the transfer of the VM image containing the function from the Store to in-network cloud infrastructure and the VM instantiation. The Orchestrator utilises the SDN control plane for network reconfiguration.

After instantiation, to facilitate Management, the function will expose, as aforementioned, an open VNF Control API for uniform VNF configuration and parameterisation by operator, the customers and by their applications.

Function Termination involves the removal of the VNF instance from the virtualised infrastructure, also involving the necessary network re-configuration.

A critical issue in programmable SatCom architectures will be security; deploying a third-party network appliance on a programmable network raises several security issues and may cause severe network malfunction due to either buggy code or malicious actions. This issue is of particular importance in SDN-as-a-Service (SDNaaS) platforms, where users are enabled to deploy arbitrary network applications for controlling network slices, but it does not fall within the scope and the objectives of IoRL project.

3.2 IoRL-specific network service structure

The network service (NS) composition happens through the dashboard (customer portal) allowing the choice of already defined NSDs, modifications of defined NS templates or the composition of a new NSD through the catalogues of available VNFs and PNFs. To deploy a NS,

 © IoRL consortium 2018

different VNFs and PNFs need to be deployed and chained. The most important fields that the NSD will contain are summarized as the following:

- The reference to the VNFD files, defining the virtual network functions to be deployed, that will refer to the VNF packages already on-boarded on the NFV manager;
- The reference to the PNFD files defining the connectivity and configuration of the needed physical network functions.

In short, VNFFG can be seen as a chain upon which the NFV Manager performs the correct routing for the chaining.

3.2.1 **Network service descriptors**

Once the NSD is defined, it is passed to the NFV manager for the deployment via the SO-NFVM interface. The NFV manager will be responsible for the applicability check of the NSD and successively for its deployment. Note that in contrast with the ETSI MANO standard, for the needs of the satellite service provision, the PNFDs is not providing only the description of the connectivity information but also need to provide the necessary parameters to be set in the PNFs. Generally speaking, each PNF parameter might have different implementation and configurability. Since the NFVO is following the ETSI MANO approach, the check of the NS prior to deployment will be composed by the following steps:

- The NSD is sent to the NFVO;
- The NFVO checks the applicability of the NS deployment, considering both the requested VNFs and PNFs for connectivity (through its south-bound interfaces);
- The SO checks the applicability of the PNF related to the requested PNFs configuration.

If both applicability checks are satisfied, then the VIM deploys the VNFs and sets the connection points between the deployed VNFs and the requested PNFs and the SO configures the PNFs accordingly to the requested configurations in the NSD.

3.2.2 **Network service lifecycle**

The network service lifecycle refers to the stages a VNF application needs to follow to implement successfully in the IoRL home network. It consists of six main stages namely (i) development, (ii) publication, (iii) selection, (iv) deployment, (v) management and (vi) termination.

- Development: This stage regards the software implementation of network services, which is performed by function providers to allow services to be published and aggregated in the IoRL system. During development stage a network service can be uploaded as VNF to the NFVI using OpenStack API and run in the execution environment under the coordination of the IoRL orchestrator. It also allows the VNF developer to provide the metadata description of the network service, which includes both functional and non-functional information that will be used by different elements of the IoRL framework. The outcome of the development stage is one or more VM images and the related metadata files to be available for publication.
- Publication: This stage performs the network service publication at the network store, which includes a repository that can host both the service/function image (as stand-alone application or integrated VM) and the associated description/metadata.
- Selection: This stage allows users the most suitable network services according to their requirements. Note that the selection of appropriate service for desired services is

 © IoRL consortium 2018

based on the metadata of the VNFs. Thus, the selection can be made among the already uploaded VNFs.

- Deployment: This stage installs and initialises the network services with their metadata in the NFVI such that VNFs can be ready to be activated. Since the deployment stage of a network service consists of transferring the VNF service from the VM image(s) to the network infrastructure, it requires the interaction between the IoRL orchestrator and the VIMs.
- Management: This is the running stage of a network service and it is dedicated to real-time management of the running VNF. During management stage, to setup and control a VNF service in real-time, an API is exposed to the orchestrator, which is composed by the set of the following four sub-stage:

  - set-up: The set-up sub-stage consists of the initialisation phase for network services. During set-up the orchestrator asks the VIM to start the service, which enables direct interactions with the VNFs (e.g. configuration of the IP interfaces). The interactions occur through the IoRL API supported by the VNF service. The bootstrap initialisation results from an indirect interaction with the NFVI, while the NIC configuration is a direct interaction with the orchestrator. NIC uses the network service metadata information for learning about the VNF IP interfaces. Moreover, it needs to know the service description for configuring the service graph or forwarding graph in the right way. If needed, the orchestrator can also run network configuration tasks. In this case the IoRL orchestrator asks the VIM to reconfigure the network. In case a network service is composed by numerous VMs or numerous VNF components the orchestrator repeats the set-up process for all VMs.

  - start: Is a sub-stage to instruct VNFs to start providing their network services.

  - stop: Is a sub-stage to instruct VNFs to stop providing their network services. When stop command occurs VNFs receive a new start command to restart providing their services, otherwise, the service lifecycle proceed to termination stage. Note that there are two kinds of stop sub-stages: the graceful and immediate stop. Graceful stop is a request to the VNF to stop accepting new network service session request. However, active service sessions continue to be supported until their natural termination. Immediate stop is a request to force down the termination of all active service session.

  - terminate: Is a sub-stage to terminate services from VNFs.

  Notice that the management stage is controlled by the IoRL orchestrator to allow direct interactions between the VNFs and VIMs. However, indirect interactions between the VNFs and the virtual infrastructure can also occur due to the execution of the virtual machines through VNF composition. Such indirect interactions are common to all VMs in the cloud network environment.

- Termination: This stage removes the VNF instance from the virtualised infrastructure using network re-configuration process when necessary. This is the final phase of the VNF lifecycle that occurs at the end of the provisioning of the service implemented by the VNF. In case a VNF has been composed in numerous either VMs or VNF components, the orchestrator repeats the termination process for all the involved

 © IoRL consortium 2018

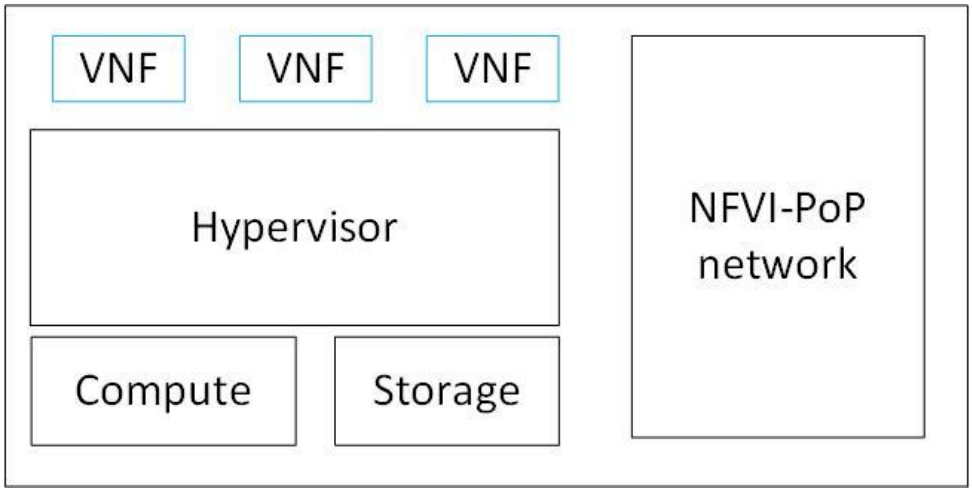


Figure 3-16 - Illustration of the NFVI-PoP structure

VMs. In such instances the NIC configuration may require complex interactions with the VNF.

From the aforementioned stages of the network service lifecycle we see that actual interactions between the IoRL orchestrator and network services form VNFs occur during management and termination states.

3.2.3 **Design of home IP gateway with NFVI-PoP layered architecture**

This Section describes the IoRL NFVI infrastructure with orchestration and monitoring process and presents the generic and specific metrics to be utilized for evaluating the internal status and performance of each VNF in the IoRL system.

3.2.3.1 **NFVI infrastructure layer/hardware specifications**

An NFVI compute cluster is inherently virtualisation-capable and consists by the three domains illustrated in Figure 3-16.

- The *Compute* domain represents the lowest (physical) level of the virtual network and comprises computing and storage equipment (standard high-volume servers with or without specialized hardware accelerations and storage infrastructure). For standard-scale data centre implementations, servers based on the x86 architecture are a common choice. The adoption of features for hardware-assisted virtualization, such as data-plane-development-kit support and single-root I/O vIrtualisation, seems quite promising for the enhancement of VNF performance and is thus recommended.
- The *Hypervisor* domain is responsible for the abstraction of the physical compute and storage resources (possibly aggregated across multiple physical elements) and their assignment/allocation to VNFs. The hypervisor domain mediates the resources of the computer domain to the virtual machines of the software appliances. Hypervisors as developed for public and enterprise cloud requirements place great value on the abstraction they provide from the actual hardware such that they can achieve very high levels of portability of virtual machines. In essence, the hypervisor can emulate every piece of the hardware platform even in some cases, completely emulating a CPU instruction set such that the VM believes it is running on a completely different CPU architecture from the actual CPU on which it is running. Such emulation, however, has

 © IoRL consortium 2018

a significant performance cost. The number of actual CPU cycles needed to emulate virtual CPU cycle can be large. The hypervisor commonly exposes a northbound interface for the interaction with the Management layer. Several choices are available for the hypervisor technology (Hyper-V, VMware, KVM, Xen etc.), heavily depending on the compatibility with the VIM and also with the physical infrastructure. KVM would be a safe recommendation, given its openness, wide compatibility and full-featured integration with OpenStack.

- The *infrastructure network domain*, which within the NFVI includes all networking elements, such as SDN and non-SDN switches and routers that interconnect all the compute/storage infrastructure of the compute domain. SDN-enabled switches are considered throughout the IoRL architecture for the NFV Infrastructure network domain in order to facilitate SDN-based network management. Both physical and virtual switches are foreseen. In regard to the SDN technology, Openflow would be the prevailing candidate technology for network controller, due to its flexibility and significant momentum, preferably at its latest version although other SDN alternatives such as NETCONF/YANG could be also considered.

For the virtual switches, Open vSwitch is by far the prevailing option, used in many production infrastructures, including large-scale cloud deployments.

3.2.3.2 **Infrastructure management layer**

In IoRL virtual network, the core management element of the Infrastructure Management Layer is the virtualised infrastructure manager (VIM), conforming to ETSI ISG NFV terminology, which is the functional entity responsible for controlling and managing the infrastructure (compute, storage and network) resources. The management scope of the VIM is generally restricted within a single NFVI-PoP. Thus, in a full deployment architecture, multiple VIMs may operate across operator data centres providing multi NFVI-PoPs that can operate independently or cooperatively as required under the control of a Federator/Orchestrator.

While a VIM, in general, can potentially offer specialisation in handling certain NFVI resources, in the IoRL context (but also in other architectures), the VIM is seen to encompass all management and control functionalities needed for the proper administration of the infrastructure, as well as the virtualised services running on top of it.

In specific, the following key tasks are performed by the VIM:

- Maintenance of a resource, capability and topology repositories/inventories, thus establishing a comprehensive “map” of the underlying hardware;
- Joint management of the infrastructure (compute, storage, networking) resources;
- Association/mapping of the virtualised services to the infrastructure resources;
- Basic network control services, including topology management and path computation;
- Management (create, query, update, delete) of service function chains i.e. the interconnections among VNFs, by creating and maintaining Virtual Links, virtual networks, sub-nets, and ports;
- Management of VM software images (add, delete, update, query, copy) that host VNFs;
- Management of virtual networks, tunnels and QoS, where applicable;

 © IoRL consortium 2018

- Collection and communication of measurements and faults/events information relative to physical and virtual resources.

In order to realise these tasks, the VIM needs to comprise the following components:

- A resource repository database - for maintaining a comprehensive landscape of the underlying infrastructure, the exposed capabilities and the available resources;
- A topology and service function chain management module - this component undertakes most network management tasks, including virtual network management, interconnection of virtualised components and tunnel establishment;
- A compute/hypervisor management module - this component undertakes the management of VMs/VNFs;
- An integrated monitoring and event/alarm management framework - for efficient and effective collection of metrics and production of events/alarms;
- A set of southbound interfaces for managing the infrastructure (compute nodes, hypervisors, network elements). These commonly come in form of “plug-ins” in order to accommodate multiple infrastructure technologies (such as several hypervisors, network management protocols etc.);
- A set of northbound APIs (commonly REST-based) for communication with the upper layer (NFV Management).

From the implementation point of view, a VIM is commonly realized by coupling a network controller and a cloud controller platform. As an example, most VIM platforms under development, including the one to be released via the OPNFV initiative are based on a customized combination of the OpenStack cloud controller with the OpenDayLight network controller platform. Indeed, OpenStack and OpenDayLight constitute the two most popular candidate technologies for VIM implementation to date.

3.2.3.3 **NFV orchestration layer**

In the IoRL virtual network, we consider an NFVO platform as the single top-level management entity of the domain. The *NFV Orchestration layer* includes for the IoRL service provider domain the management entity of the NFVI PoPs. The NFVO is the management entity, which is responsible for the management of the NS lifecycle, which includes the NS instantiation, the dimensioning and the termination.

The NFV manager receives appropriate commands from the upper layer, which include the NS descriptors, which will initiate the VNF instantiation with the appropriate network configuration internally in the NFVI PoP and the NMS will accordingly apply the appropriate cross domain network rules, i.e., inter-NFVI-PoP connectivity.

The NFVO is the entity to maintain a complete view of the whole infrastructure of the domain; it keeps a record of installed and available resources, as well as the infrastructure topology.

3.2.3.4 **Introduction to NFVI-PoP monitoring**

Up to this point we have identified the architectural concepts and requirements for the NFVI and VIM layers of PoP. The technical requirements to drive the specification and development of the IoRL monitoring framework can be directly derived/inherited by the specific NFVI/VIM requirements. In Table 1 we identify the NFVI/VIM requirements that - either directly or indirectly - are associated to IVM monitoring, focusing on NFVI-PoP resources to describe how each of these are translated to a specific requirement for the monitoring framework.

 © IoRL consortium 2018

Table 1 - NFVI/VIM requirements which affect the monitoring framework

IVM Req.ID	IVM Requirement Name	Requirement for the Monitoring Framework
VIM.1	Ability to handle heterogeneous physical resources	The MF must provide a vendor agnostic mechanism for physical resource monitoring.
VIM.2	Ability to provision virtual instances of the infrastructure resources	The MF must be able to report the status of virtualized resources as well as from physical resources in order to assist placement decisions
VIM.3	API Exposure	The MF must provide an interface to the Orchestrator for the communication of monitoring metrics.
VIM.6	Translation of references between logical and physical resource identifiers	The MF must re-use resource identifiers when linking metrics to resources.
VIM.8	Control and Monitoring	The MF must monitor in real time the physical network infrastructure as well as the vNets instantiated on top of it, providing measurements of the metrics relevant to service level assurance.
VIM.9	Scalability	The MF must keep up with dynamic increase of the number of resources to be monitored
VIM.18	Query API and Monitoring	The MF must provide an API for communicating metrics (in either push or pull mode)
VIM.21	Virtualised Infrastructure Metrics	The MF must collect performance and utilisation metrics from the virtualised resources in the NFVI.
C.7	Compute Domain Metrics	The MF must collect compute domain metrics.
C.12	Hardware accelerator metrics	The MF must collect hardware accelerator metrics
H.1	Compute Domain Metrics	The MF must collect compute metrics from the Hypervisor.
H.2	Network Domain Metrics	The MF must collect network domain metrics from the Hypervisor.
H.12	Alarm/Error Publishing	The MF must process and dispatch alarms.
N.5	Usage monitoring	The MF must collect metrics from physical and virtual networking devices.
N.8	SDN Management	The MF must leverage SDN monitoring capabilities.

By consolidating the aforementioned requirements, it becomes clear that the basic required functionalities of the IVM monitoring framework are as follows:

1. Collection of IT and networking metrics from virtual and physical devices of the NFVI. It should be noted that at the NFVI/VIM level, metrics correspond only to physical and

 © IoRL consortium 2018

virtual nodes and are not associated to services since the VIM does not have knowledge of the end-to-end Network Service. Metrics are mapped to Network Services at Orchestrator level;

2. Processing and generation of events and alarms;
3. Communication of monitoring information and events/alarms to the Orchestrator in a scalable manner;

The following Section overviews several technological frameworks for NFV monitoring which could be partially exploited towards fulfilling these requirements.

3.2.3.5 NFVI-PoP monitoring architecture and functional entities

The overall architecture of the IoRL VIM monitoring framework can be defined by taking into account the technical requirements, as identified in Section 2, as well as the technical choices made for the NFVI and VIM infrastructure. The specification phase has concluded that the OpenStack platform will be used for the control of the virtualised IT infrastructure, as well as the OpenDayLight controller for the management of the SDN network elements.

In this context, it is proper to leverage the OpenDayLight (Statistics API) and OpenStack (Telemetry API) capabilities for collecting metrics, rather than directly polling the network elements and the hypervisors at NFVI layer, respectively. Theoretically, it would be possible for the Orchestrator to directly poll the cloud and network controllers of each NFVI-PoP and retrieve resource metrics respectively. This approach, although simple and straightforward, would introduce significant scalability issues on the Orchestrator side. Thus, it seems appropriate to introduce a mediator/processing entity at the VIM level to collect, consolidate, process metrics and communicate them to the Orchestrator. We call this entity VIM monitoring manager (VIM MM), as a stand-alone software component. VIM MM is designed and developed in IoRL as a novel component, without depending on the modification of existing monitoring frameworks. With regard to the collection of monitoring information, OpenStack and OpenDayLight already provide a rich set of metrics for both physical and virtual nodes, which should be sufficient for most IoRL requirements. However, in order to gain a more detailed insight on the VNF and the NFVI status and operation, we consider advisable to also collect a rich set of metrics from the guest OS of the VNF container - including information which cannot be obtained via the hypervisor - as well as the compute node itself.

For this purpose, we introduce an additional VNF monitoring agent (VNF-MA), deployed within the VNF VMs. VNF-MA intends to augment VNF monitoring capabilities, by collecting a large variety of metrics, as declared in the VNFD document of each VNF and also at high temporal resolution. The monitoring VNF-MA can be either pre-installed in the VNF image or installed upon VNF deployment. It must be noted, however, that in some cases the presence of an VNF-MA might not be desirable by the VNF developer for several reasons (e.g. resource constraints, incompatibilities etc.). In this case, the system can also work in VNF-MA -less mode, solely relying on Ceilometer data for VNFs, which do not have an VNF-MA installed. In addition to collecting generic VNF and infrastructure metrics, the VIM MM is also expected to retrieve VNF-specific metrics from the VNF application itself. For this purpose, we have developed specific lightweight libraries (currently in Python, but planned to expand to other languages), which can be used by the VNF developer to dispatch application-specific metrics to the VIM MM.

 © IoRL consortium 2018

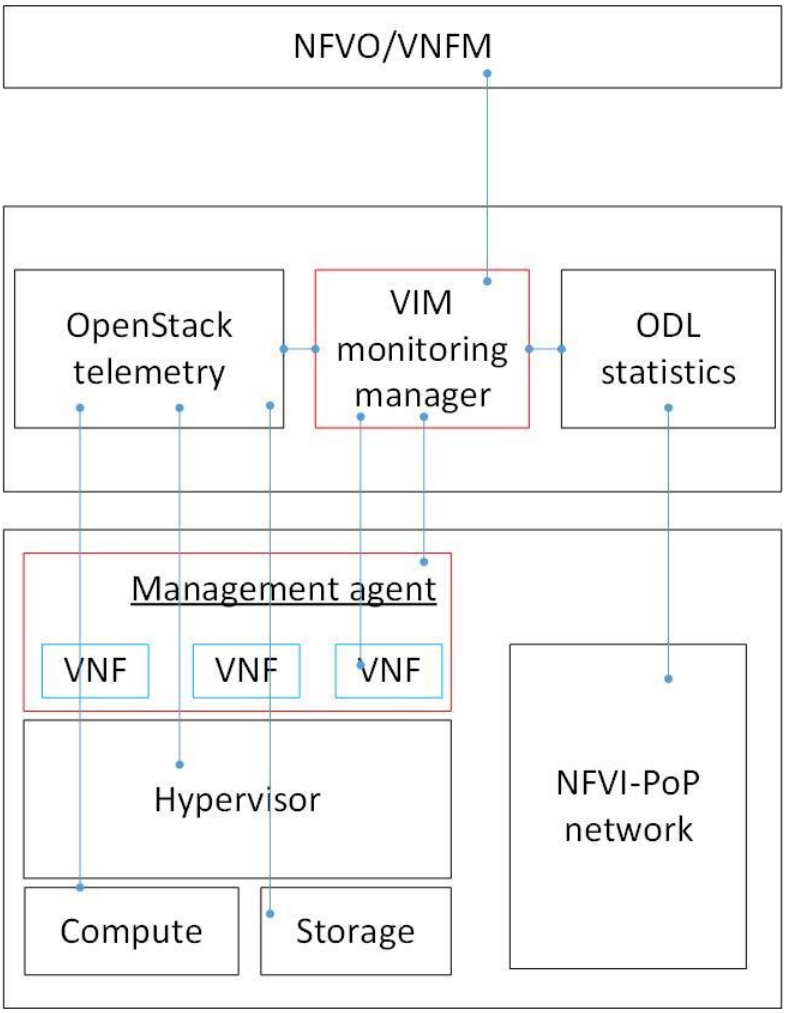


Figure 3-17 - Illustration of the IoRL VIM monitoring modules

Although traditionally the VNF metrics are supposed to be directly sent to the VNF Manager, for the sake of simplicity we chose to exploit the already established VIM monitoring framework to collect and forward VNF metrics to the VNF Manager through the VIM, rather than implement a second parallel "monitoring channel". Based on the above design choices, we can define the architecture of the IoRL VIM monitoring as shown in Figure 3-17.

The VIM MM aggregates metrics by polling the cloud and network controllers and by receiving additional information from the monitoring agents as well as the VNF applications, consolidates these metrics, produces events/alarms if appropriate and communicates them to the Orchestrator. For the sake of scalability and efficiency, it was decided that metrics will be pushed by the VIM MM to the Orchestrator, rather than being polled by the latter. Moreover, the process of metrics collection/communication and event generation can be partially configured by the Orchestrator via a relevant configuration service to be exposed by the VIM MM. More details on the introduced modules can be found in the sections to follow.

3.2.4 **NFVI monitoring metrics**

3.2.4.1 **Generic metrics**

A crucial task when defining the IoRL approach for monitoring is the identification of metrics that need to be collected from the virtualised infrastructure. Although the list of metrics that

 © IoRL consortium 2018

are available via the existing controllers can be quite extensive, it is necessary, for the sake of scalability and efficiency, to restrict this list to include only the information that is needed for the implementation of IoRl use cases, as defined in Deliverable D2.1. In Table 2 we summarise a list of such metrics, which are "generic" in the sense that they are not VNF application-specific. This list is meant to be continuously updated throughout the project to align with the technical capabilities and requirements of the components under development and the use cases which are implemented.

Table 2 - NFV generic service quality monitoring metrics

Domain	Metric	Units	Origin	Relevant UCs
VM/VNF	CPU utilisation (user & system)	%	VNF Mon.Agent	UC3, UC4
VM/VNF	Free space in root FS	MB	VNF Mon.Agent	UC3, UC4
VM/VNF	RAM available	MB	VNF Mon.Agent	UC3, UC4
VM/VNF	System load (short/mid/long term)	%	VNF Mon.Agent	UC3, UC4
VM/VNF	No. of processes (running/sleeping etc)	#	VNF Mon.Agent	UC3, UC4
VM/VNF	Network Interface in/out bitrate	Mbps	VNF Mon.Agent	UC3, UC4
VM/VNF	Network Interface in/out packet rate	pps	VNF Mon.Agent	UC3, UC4
VM/VNF	No. of processes	#	VNF Mon.Agent	UC4
Compute Node	CPU utilisation	%	OS Telemetry	UC2, UC3, UC4
Compute Node	RAM available	MB	OS Telemetry	UC2, UC3, UC4
Compute Node	Disk read/write rate	MB/s	OS Telemetry	UC3, UC4
Compute Node	Network i/f in/out rate	Mbps	OS Telemetry	UC3, UC4
Storage (Volume)	Read/write rate	MB/s	OS Telemetry	UC3, UC4
Storage (Volume)	Free space	GB	OS Telemetry	UC2, UC3, UC4
Network (virtual/physical switch)	Port in/out bit rate	Mbps	ODL Statistics	UC2, UC3, UC4
Network (virtual/physical switch)	Port in/out packet rate	pps	ODL Statistics	UC3, UC4

 © IoRL consortium 2018

Network (virtual/physical switch)	Port in/out drops	#	ODL Statistics	UC3, UC4

In regard to metrics identification, ETSI defines and describes some metrics related to service quality, as perceived by the NFV consumer. These metrics are overviewed in Table 3.

Table 3 - NFV service quality metrics specified by ETSI

Service Metric Category	Spec	Accuracy	Reliability
Orchestration Step 1 (e.g. Resource Allocation, Configuration and Setup)	VM Provisioning Latency	VM Placement Policy Compliance	VM Provisioning Reliability VM Dead-on-Arrival (DOA) Ratio
Virtual Machine operation	VM Stall (event duration and frequency) VM Scheduling Latency	VM Clock Error	VM Premature Release Ratio
Virtual Network Establishment	VN Provisioning Latency	VN Diversity Compliance	VN Provisioning Reliability
Virtual Network Operation	Packet Delay Packet Delay Variation (Jitter) Delivered Throughput	Packet Loss Ratio	Network Outage
Orchestration Step 2 (e.g. Resource Release)	-	-	Failed VM Release Ratio

From the table we see that apart from the service latency metrics which are related to the provisioning and/or reconfiguration of the service and essentially refer to the response of management commands (e.g. VM start), the remaining metrics can be directly or indirectly derived from the elementary metrics identified in Table 2 as well as the events/alarms associated. However, it is up to the orchestrator, which has a complete view of the service, to assemble/exploit VIM metrics in order to derive the service quality metrics to be exposed to the SP and the customer via the dashboard. These metrics will be used as input to enforce the service level agreement (SLA) that will be finally evaluated at Marketplace level for the applicability of possible rewards to the customer in case of failure.

3.2.4.2 **VNF-specific metrics**

Apart from the generic metrics identified in previous Section, each VNF generates specific dynamic metrics to monitor its internal status and performance.

These metrics:

 © IoRL consortium 2018

- are specified inside the VNFD as monitoring-parameters (both for the VDUs and for the whole VNF) to define the expected performance of the VNF under certain resource requirements.
- are sent by the VNF application to the VIM monitoring manager, either via the agent or directly (see details in Section 2.3.2.1)
- are processed, aggregated and forwarded, if required, to the upper layers (orchestrator and marketplace).

At the Orchestration level, some of the VNF-specific metrics can be used for automating the selection of the most efficient VNF flavour in terms of usage of resources, to achieve a given SLA (for example using automated scaling procedures - see details in IoRL Deliverable 3.3).

These VNF-specific metrics that may be part of the SLA agreed between SP and customer will be evaluated for business and commercial clauses (e.g: penalties, rewards, etc.) that will finally impact in the billing procedure (see D6.4).

Table 4, Table 5 and Table 6 present lists of VNF-specific metrics of the VNFs being developed in IoRL for virtual Home Gateway (vHG), virtual proxy and virtual traffic classifier, respectively. The presented lists are tentative and are meant to be continuously updated as the VNF applications evolve. Notice most of these metrics, which refer to the specific functionality of each VNF as well as its component software modules.

Table 4 - Monitoring metrics for VNF of virtual home gateway

Metric	Description	Units
remaining_storage_size	Remaining Storage Size	Bytes
transcoding_score	Transoding Score	double
httpnum	Number of HTTP requests received	# (incremental)
hits	Cache hits percentage of all requests for the last 5 minutes	%
hits_bytes	Cache hits percentage of bytes sent for the last 5 minutes	%
cachediskutilization	Cache disk utilization	%
cachememkutilization	Cache memory utilization	%

Table 5 - Monitoring metrics for VNF of virtual proxy

Metric	Description	Units
httpnum	Number of HTTP requests received	# (incremental)
hits	Cache hits percentage of all requests for the last 5 minutes	%
hits_bytes	Cache hits percentage of bytes sent for the last 5 minutes	%

 © IoRL consortium 2018

diskhits	Disk hits percentage for the last 5 minutes (hits that are logged as TCP_HIT)	%
cachediskutilization	Cache disk utilization	%
cachememkutilization	Cache memory utilization	%
usernum	Number of users accessing the proxy	%
cpuusage	CPU consumed by Squid for the last 5 minutes	%

Table 6 - Monitoring metrics for VNF of traffic classifier

Metric	Description	Units
pps	Packets per second processed	pps
flows	Flows per second	# (average)
totalflows	Total flows	# (incremenetal)
protocols	Application Protocols	# (incremenetal)
mbits_packets_all	Total Throughput	Mbps

3.3 Deployment of the identified VNFs in the IoRL home network with preliminary implementation testbed

Having identified the IoRL-specific VNFs with network service structure and NFVI monitoring metrics, we can now update Figure 3-2 with Figure 3-18 to summarise the virtualised environment that will be integrated in the IoRL home network. Such environment will be able to transform VMs into VNFs and instantiate these VNFs with OSM, which gives rise to a means fully controlling and monitoring all the SDN/NFV-related functions of the IoRL home network.

Notice that in the context of VLC module implementation, shadowing phenomena arise due to the physical susceptibility blocking and misalignment (or path) obstructions, which may significantly limit QoS provision along with network reliability. In addition, although the synergy between lighting industry and ICT seems promising, it is yet theoretically unexplored, while practically untested, intriguing thereby to exploit the perceptual advantages of SDN/NFV virtualisation towards evolving VLC-oriented networks according to 5G demands. In this regard, next subsections present a first-step preliminary approach to implement an SDN/NFV-assisted version of the IoRL network considering VLC with WLAN WIFi modules and measure its performances using real-world testbed.

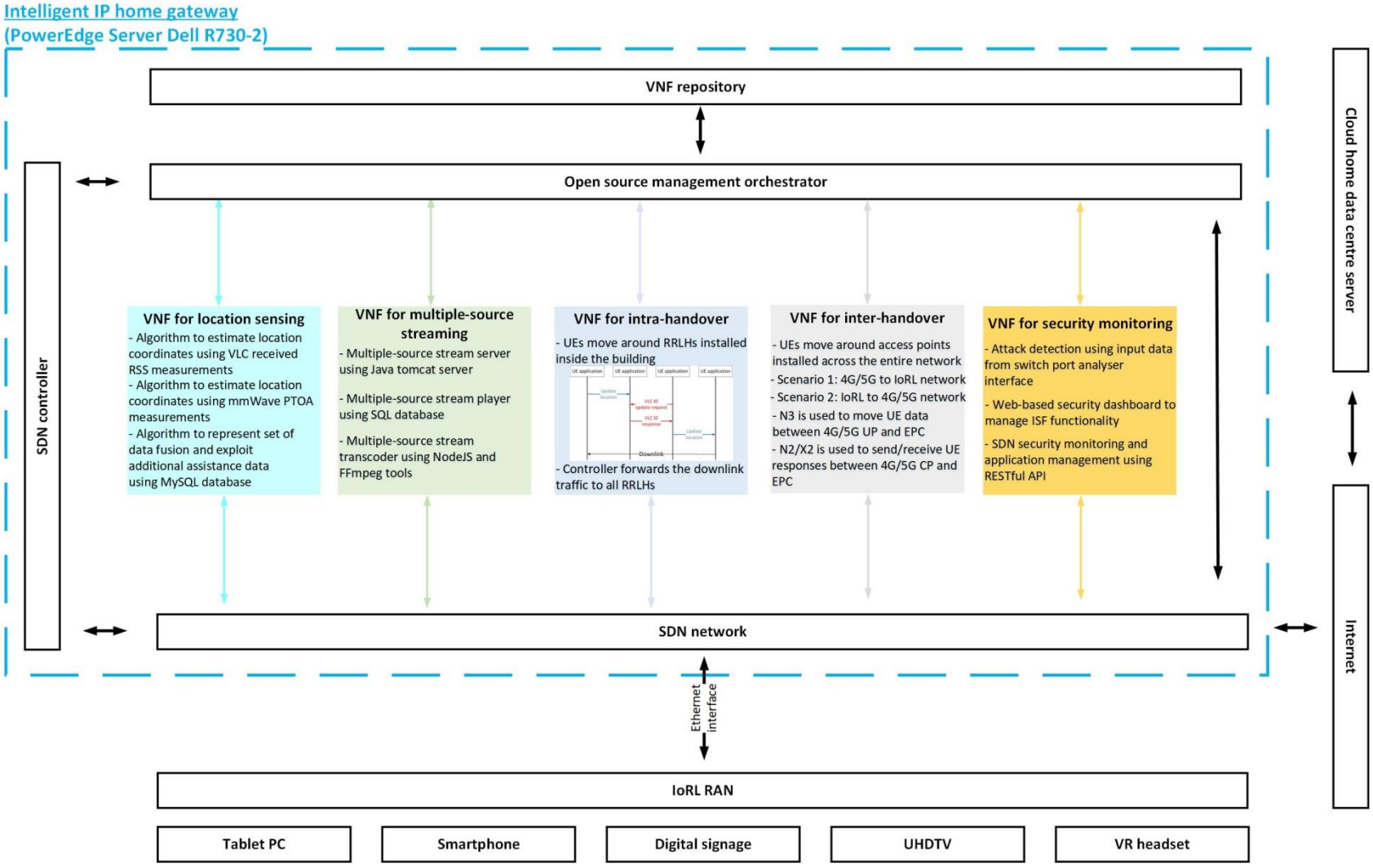
3.3.1 Aim of preliminary implementation testbed with system modelling

For this first-step experimental attempt, we have yet considered elaborating with mmWave module, which however is scheduled to be addressed in future experiments and in accordance to the findings and products of the IoRL project partners. Our intension in this experiment is to improve the provided QoS and assure reception quality by eliminating

 © IoRL consortium 2018

misalignment or path obstructions, especially focusing on users positioned in so called “dead coverage zones”, which

 © IoRL consortium 2018



 © IoRL consortium 2018

Figure 3-18 - Illustration of the IoRL Home Network with deployment of the identified VNF modellings in conjunction with SDN/NFV recent findings

© IoRL consortium 2018

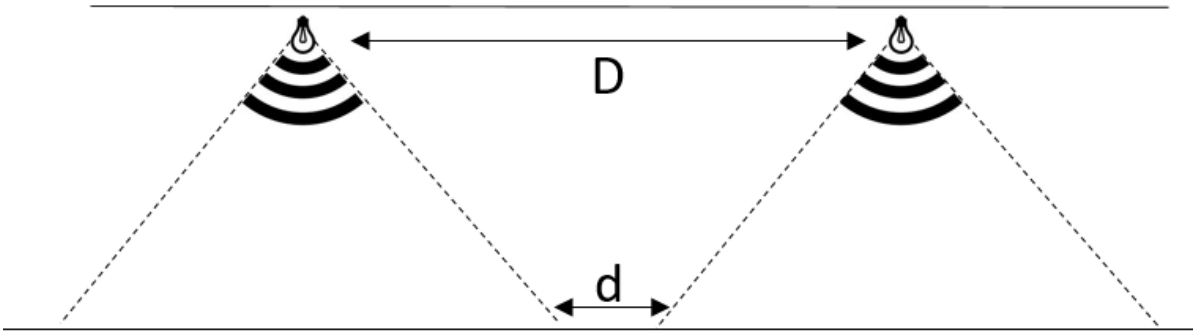


Figure 3-20 - Illustration of the dead zones between two VLC lamps

regularly exist due to shadowing in the area between two consecutive VLC transmitters. Our motivation rests on evaluating how the WiFi access network with the assistance of SDN can effectively couple VLC connectivity across two LED lamps, facing loss of connectivity at the dead zones. The considered topology has been reconstructed in lab environment and is depicted in Figure 3-20.

In particular, the two LED lamps are placed in a relatively long distance apart, as usually considered in typical indoor environment setting, in order to study how dead reception zones (denoted as d) impact the system's ability to provide service continuation, especially focusing on most demanded zones that require constant data rate and higher QoS (e.g. video services) than others. The scenario is that as users move away from the LED light centre, the signal-to-noise rate (SNR) (by means of ambient lightning) decreases, decreasing thereby the reception quality as well. Our scenario can be fully epitomised in indoor environments with users moving across different VLC/LED lamps, where in such case, SNR fluctuates from maximum to

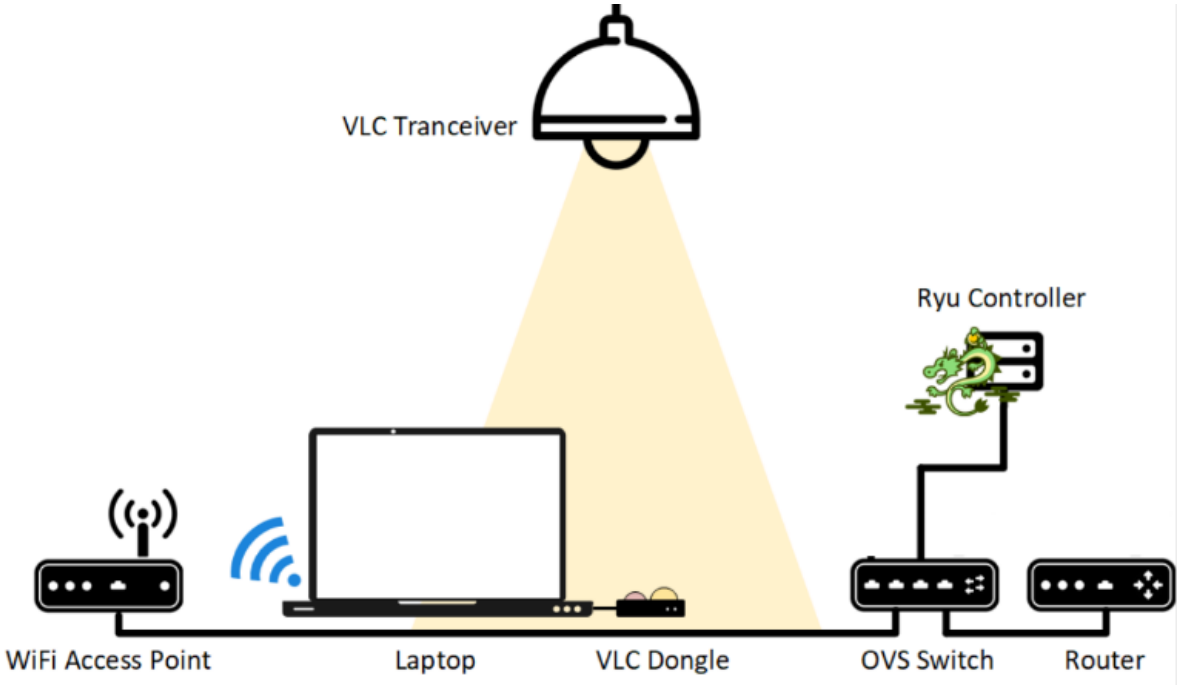


Figure 3-19 - Illustration of the experimental testbed setup

© IoRL consortium 2018

Figure 3-21 - Illustration of the actual experimental testbed including CPE with VLC receiver and Arduino photodetector

minimum values for each position change of the user between the installed lamps. Thus, depending on their relative distance from VLC points, users may not only experience SNR degradation, but also drop into “dead zones” and completely lose their connection with the network.

To study such phenomenon, we initially consider setting with D=4m the relative distance between VLC points and d=10cm the “dead zones” length. Then, we focus on the downlink direction (e.g. VLC to user) to examine how the SNR fluctuation can be improved by redirecting the user access to the WLAN WiFi module, and vice versa, using SDN-assisted approach with corresponding application. The SDN-assisted approach aims to address how a 2.4GHz WiFi 802.11g access point can be used as the dynamically coupled access technology in conjunction with VLC transceivers to address SNR fluctuation at “dead zones” where users have poor reception quality.

An SDN application is also developed to handle the seamless switching of the access

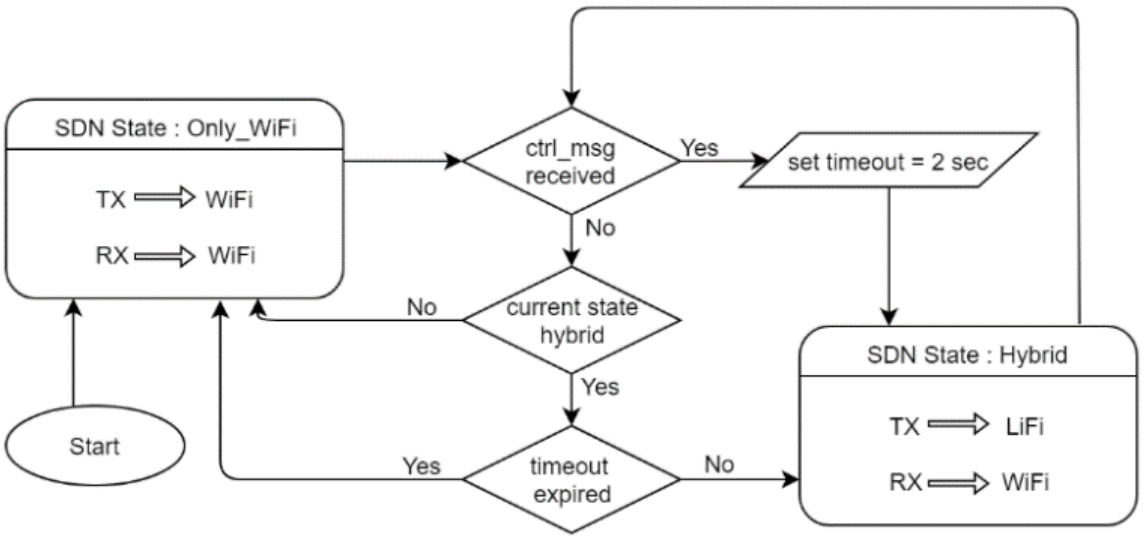


Figure 3-22 - Flow diagram of the proposed SDN application

 © IoRL consortium 2018

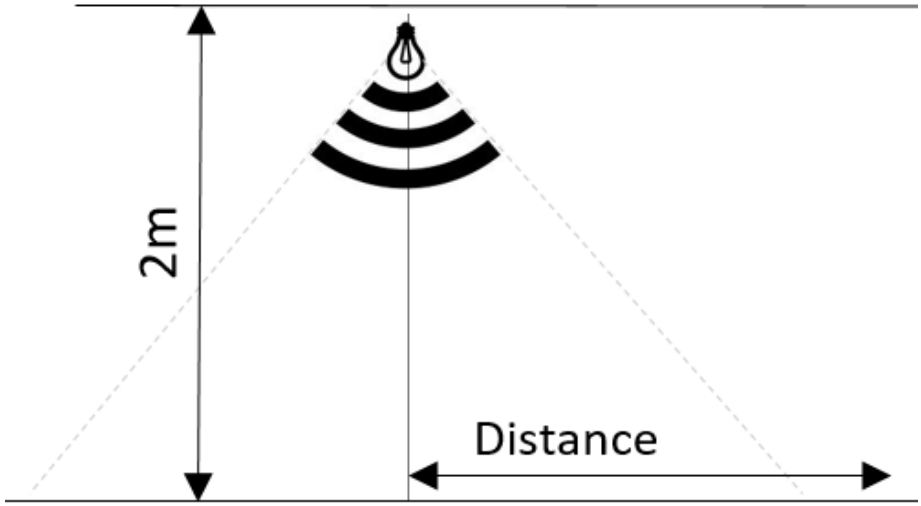


Figure 3-23 - Distance definition between the experimental setup

technology (i.e. WiFi or VLC) used each time to maintain good reception quality.

To this end, Figure 3-19 illustrates the experimental setup of both VLC and WLAN WiFi modules in the considered topology to evaluate how SDN-assisted networks can guarantee the continuous provision of adequate QoS to a moving user across “dead zones”. In particular, our experimental setup includes an SDN network domain, which is managed by Ryu controller using python-based SDN applications (see Section 2 for more details about the efficiency of Ryu platform). We also consider a customer-premises equipment (CPE) device by means of a laptop equipped with a VLC receiver dongle connected through universal-bus-system (USB) and an Arduino-based luminance detector [183], placed next to the VLC dongle. Figure 3-21 depicts the actual system setup that was used for the execution of the experiment.

3.3.2 Proposed SDN architecture with development of SDN application

The SDN approach executes the proposed experiment using three-tier structure. The first tier comprises the physical network infrastructure to include all the required devices, which can be virtualised at second tier through the Openflow protocol for SDN controller design. The idea is to decouple the network control from network devices by transforming it into a software application (i.e. the SDN controller). Such SDN controller can act as a strategic control point in the SDN network by intelligently managing the flow control to the switches/routers (via southbound APIs) and applications (via northbound APIs). These controllers, used to initiate and terminate traffic, consist the second (virtualised) tier of the architecture. Furthermore, the third tier refers to the SDN applications, which are specific functions directly processed into the SDN controller. In other words, the SDN application is the software program designed to perform a specific task in the SDN environment such as programs for network virtualisation, intrusion detection (IDS), flow balancing (the SDN equivalent of load balancing), network monitoring, etc. In our experiment, the SDN application is to employ handover management by coupling the VLC access channel with WLAN WiFi to reassure the user access and satisfaction of QoS requirements.

 © IoRL consortium 2018

Figure 3-22 illustrates the flow diagram of the developed SDN application from its finite state machine (FSM) point of view. From the figure we see that the proposed SDN application supports two modes of operation, namely the “WiFi only” and the “Hybrid” mode, which can be described as below.

- The SDN application starts by gaining connectivity via the WLAN WiFi access technology (i.e. “WiFi only” mode);
- Then, the SDN application performs continuous monitoring to capture VLC control messages via the VLC receiver through which messages we can confirm that the user is located at a coverage area of a VLC transceiver;
- Continuing, the SDN application mandates the SDN controller to apply the appropriate OpenFlow commands at the SDN devices of the domain (in our experimental topology an OpenVSwitch is used) in order the end-user terminal to start receiving the download link via the VLC and the return link via the WiFi (i.e. “Hybrid” mode);
- While in “Hybrid” mode the user receives service data from the VLC access technology and uses the WiFi channel only for requests and ACKs (as a return channel). Note that the SDN application forces the system to remain in “Hybrid” mode for at least 2 seconds before performing a check loop for moving to the next state;
- Then, the system checks if the user remains at a coverage area of VLC lamp to either remain in “hybrid mode” for another two seconds, or
- returns to “WiFi only” mode.

3.4 Experimental results considering SDN architecture with application in the IoRL home network

We perform two key experiments to study how SDN applications can apply and implement in an IoRL home network consisting of VLC and WLAN WiFI modules for user access. The first experiment regards measures for the distribution of LED power with respect to user positioning and QoS provision, while the second experiment deals with SDN-enabled hybrid access using either VLC or WLAN WiFi modules.

3.4.1 Experiment on LED lamp power distribution

Our first experiment aims to verify the QoS provide to an IoRL user moving across LED lights and dead zones by measuring the LED power distribution between two successive VLC transceivers. In our setup, we situate the VLC lamps two meters higher than the luminance sensor and VLC receiver position (e.g. at the ceiling) as shown in Figure 3-23. Our focus is to measure the luminance intensity (in lux) and packet loss percentage using the CPE at time points where the IoRL user requests an HTTP streaming video service, while setting the buffer refresh rate at the client side at 1 second interval time. In Figure 3-25 we illustrate our measurements taken by placing the CPE with the VLC module and luminance sensors at distances from 0-300 cm (e.g. the distances between the LED lamp and the floor), following a step of approximately 30 centimeters. We see that while the user moves away from the centre of the LED light, the luminance power (which is respective to SNR) decreases in linear fashion. This consequence is that as distance increases the reception quality becomes worse, degrading thereby the status of the http-streaming video service which variates from normal to no playback (e.g. occasional pauses occur). Gradually, luminance becomes equal to zero as the user is positioned in a dead reception zone (e.g. between two successive lamps), where in that case the http-streaming video service is always equal to zero (e.g. continuously paused).

 © IoRL consortium 2018

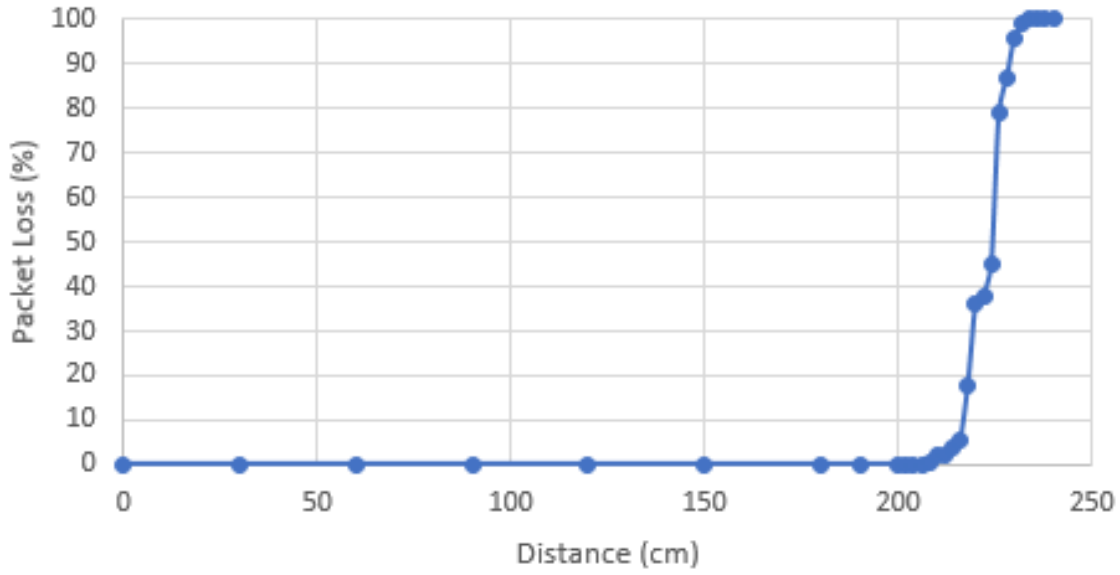


Figure 3-25 - Simulation result regarding the luminance LED power (in lux) versus distance from CPE (in cm) considering VLC module with http-streaming video user QoS requirement

The video service playback re-initiates when the user reaches the c overage area of next VLC transmitter (e.g. when the distance from next LED light source decreases), where luminous with SNR starts increasing gradually. In conclusion, using VLC setup, a moving user between two successive LED lights, can experience reception fluctuations, which affect its QoS, especially when the user is positioned in dead zones with no VLC service coverage.

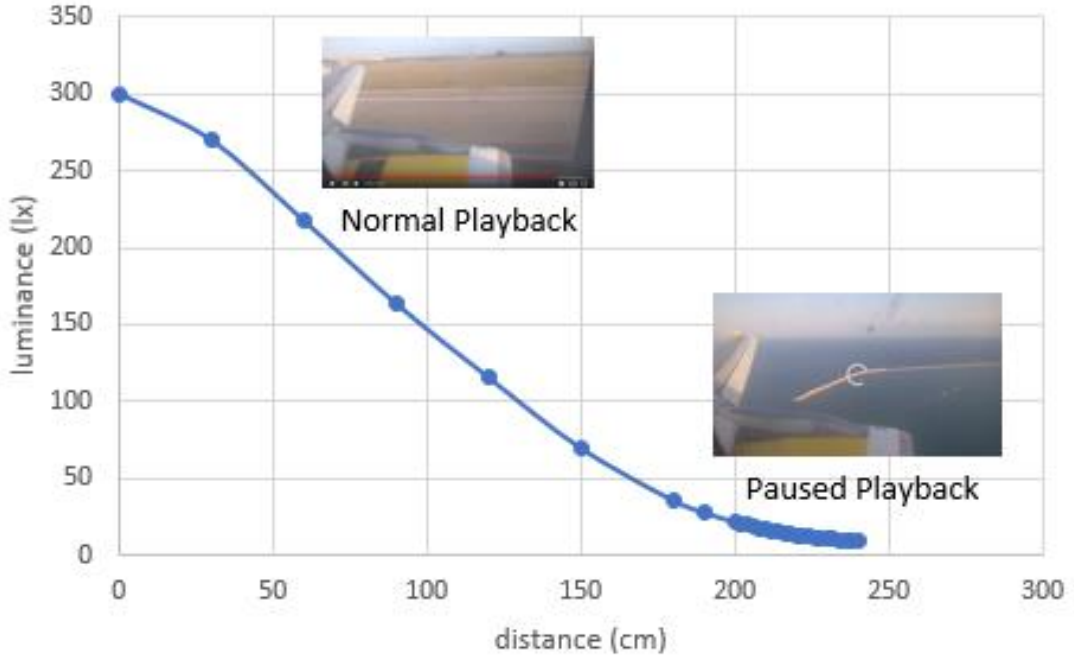


Figure 3-24 - Simulation result regarding the packet loss (%) versus distance from CPE (in cm) considering VLC module with http-streaming video user QoS requirement

 © IoRL consortium 2018

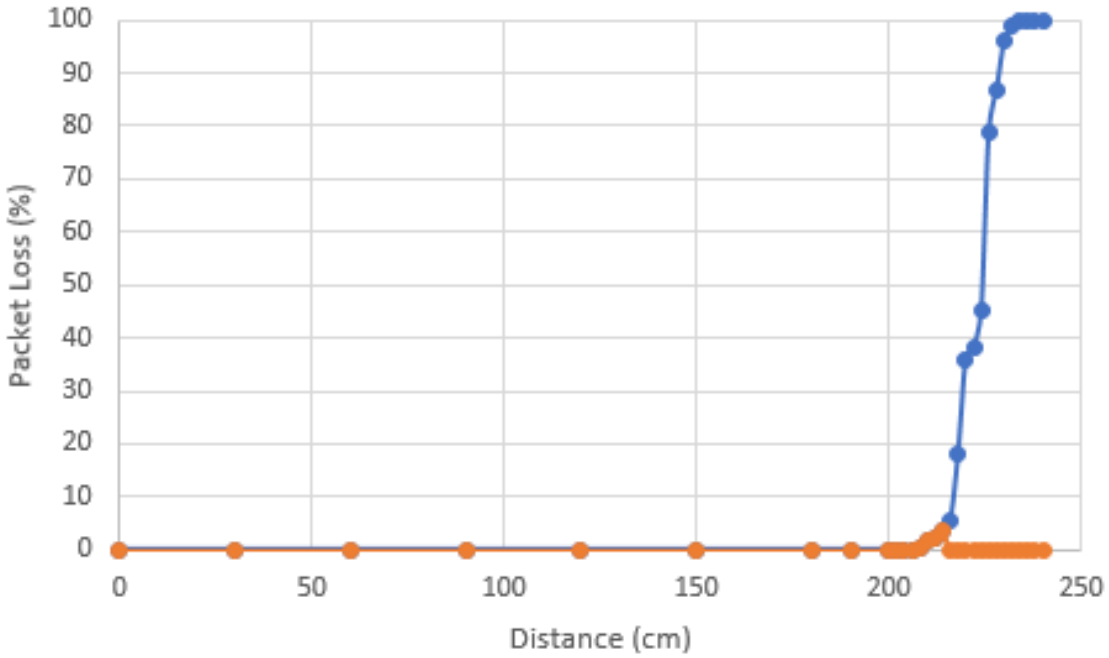


Figure 3-26 - Simulation result regarding the packet loss (%) versus distance from CPE (in cm) considering SDN/NFV-enabled joint VLC-WLAN WiFi module (orange graph) with video user QoS requirement

3.4.2 Experiment on SDN-enabled hybrid VLC-WLAN WiFi access IoRL home network

Our second experiment aims to study the QoS provision in the IoRL home network by allowing user access through either VLC or WLAN WiFI modules. For this experiment, we use a SDN controller, which is configured according to the proposed SDN application. Our intension is to study the packet loss at the http-streaming video service considering that user moves across the VLC LED lights with step of approximately 30 centimeters.

As shown in Figure 3-24, when user covers two-meter distance (e.g. it reaches dead coverage zone between two sequential LED lights), packet loss starts increasing. In that situation, our SDN/NFV application can detect the packet loss using the so-called ctrl_messages via the VLC channel, and within 2-second time intervals it instructs diverting the downloading flow to WLAN WiFi module using the SDN/NFV appropriate commands in OpenFlow platform for orchestrating the OVS switching process. The resulted performance in packet loss is shown in Figure 3-26, which reveals that our OVS switching process performs seamlessly as it maintains the required http-streaming video service with no pauses or QoS degradation. In conclusion, using VLC together with WLAN WiFiI setup driven by SDN/NFV application for OVS switching, a moving user between two successive LED lights, does not experience reception fluctuations or packet loss, while its QoS is not affected even moving in dead zones with no VLC service coverage.

 © IoRL consortium 2018

References

[1] W. Xia, Y. Wen, C. H. Foh, D. Niyato and H. Xie, “A survey on softeware-define networking,” *IEEE Commun Surveyw & Tutorials,* vol. 17, no. 1, pp. 27-51, 2015.

[2] I. F. Akyildiz, A. Lee, P. Wang, M. Luo and W. Chou, "A roadmap for traffic engineering in SDN-OpenFlow networks," *Computer Networks,* vol. 71, no. 1, pp. 1-30, 2014.

[3] OpenFlow Switch Specifcation - Version 1.3.5., March 2015. [Online].

[4] P. Ben and D. Bruce, “The Open vSwitch Database Management Protocol. RFC 7047,” 2013.

[5] F. Callegati, W. Cerroni, C. Contoli and G. Santandrea, "Implementing dynamic chaining of Virtual Network Functions in OpenStack platform," *17th International Conference on Transparent Optical Networks (ICTON),* 2015.

[6] A. Doria, J. H. Salim, R. Haas, H. Khosravi, W. Wang, R. G. L. Dong and J. Wang, "Forwarding and Control Element Separation (ForCES) Protocol Specification," *Internet Engineering Task Force (IETF),* 2010.

[7] E. Haleplidis, D. Joachimpillai, J. H. Salim, D. Lopez, J. Martin, K. Pentikousis, S. Denazis and O. Koufopavlou, "ForCES Applicability to SDN-Enhanced NFV," *Third European Workshop on Software Defined Networks,* pp. 43-48, 2014.

[8] E. Haleplidis, J. H. Salim, S. Denazis and O. Koufopavlou, "Towards a Network Abstraction Model for SDN," *Journal of Network and Systems Management,* vol. 23, no. 2, pp. 309-327, 2014.

[9] P. Saint-Andre, XMPP The Defnite Guide, Safari Book, O’Reilly, 2009.

[10] Cisco OpFlex, , "Cisco Application Centric Infrastructure Policy-Based Redirect Service Graph Design," [Online]. Available: https://www.cisco.com/c/en/us/solutions/data-center-virtualization/application-centric-infrastructure/white-paper-c11-739971.html.

[11] R. T. Fielding, "Chapter 5: representational state transfer (REST)," *Architectural Styles and the Design of Network-Based Sofware Architectures,* pp. University of California, Irvine, Calif USA, 2000.

[12] Build SDN Agilely, “COMPONENT-BASED SOFTWARE DEFINED NETWORKING FRAMEWORK,” [Online]. Available: http://osrg.github.com/ryu/.

[13] OpenDayLight, "http://www.OpenDayLight.org/project/technical-overview".

[14] L. Li, W. Chou, W. Zhou and M. Luo, “Design Patterns and Extensibility of REST API for Networking Applications,” *IEEE Transactions on Network and Service Management,* vol. 13, no. 1, pp. 154-167, 2016.

[15] A. Voellmy, H. Kim and N. Feamster, “Procera: A Language for High-level Reactive Network Control,” pp. 43-48, 2012.

[16] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story and D. Walker, "Frenetic: A Network Programming Language," *16th ACM SIGPLAN International Conference on Functional Programming (ICFP ’11),* p. 279–291, 2011.

[17] H. Cummins and T. Ward, Enterprise OSGi in Action, Manning 1st edition, 2013.

[18] D. Erickson, “The beacon openflow controller,” *2nd ACM SIGCOMM Workshop on Hot Topics in Sofware Defned Networking (HotSDN ’13),* pp. 13-18, 2013.

[19] FloodLight Project, [Online]. Available: http://www.projectfloodlight.org/floodlight/.

[20] O. Project, “http://onosproject.org/”.

[21] A. Karaf, *http://karaf.apache.org/.*

[22] OpenDayLight Controller, [Online]. Available: https://wiki.OpenDayLight.org/view/OpenDayLight_Controller:Main.

[23] Internet Engineering Task Force (IETF) , " YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)," *https://tools.ietf.org/html/rfc6020,* no. RFC 6020, 2010.

 © IoRL consortium 2018

[24] R. Hirannaiah, "Overview of Model-Driven SAL and Creating and Application based on MD-SAL," *http://events.linuxfoundation.org/sites/events/fles/slides/Radhika Hirannaiah MD-SAL ONS2016.pdf,* 2016.

[25] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig, "Sofware-defned networking: a comprehensive survey," *IEEE Proceedings,* vol. 103, no. 1, pp. 14-76, 2015.

[26] M. Pham and D. B. Hoang, "SDN applications-the intent based Northbound Interface realisation for extended applications," *IEEE NetSof Conference and Workshops (NetSof '16),* p. 372–377, 2016.

[27] Cisco, "Microservices Infrastructure," *https://github.com/CiscoCloud/microservices-infrastructure.*

[28] Open Network Foundation, "The ONOS Project," *http://onosproject.org/,* 2017.

[29] ONF, "Blog: Intent, what. Not how," *https://www.opennetworking.org/?p=1633&option=com wordpress&Itemid=155.*

[30] ODL, "Network Modelling Language (NEMO)," *https://wiki.OpenDayLight.org/view/Network Intent Composition:NEMO_Model.*

[31] B. J. v. Asten, N. L. M. v. Adrichem and F. A. Kuipers, "Scalability and resilience of sofware-defned networking: an overview," *https://arxiv.org/abs/1408.6760,* 2014.

[32] M. C. N. G. e. a. T. Koponen, "Onix: a distributed control platform for large-scale production networks," in *9th USENIX Conference on Operating Systems Design and Implementation (OSDI '10)*, Vancouver, Canada, 2010.

[33] A. Tootoonchian and Y. Ganjali, "Hyperflow: a distributed control plane for openflow," in *Internet Network Management Conference on Research on Enterprise Networking*, 2010.

[34] S. H. Yeganeh and Y. Ganjali, "Kandoo: a framework for efcient and scalable ofoading of control applications," in *1st ACM International Workshop on Hot Topics in Sofware Defned Networks (HotSDN '12)*, 2012.

[35] NOXRepo.org, "The POX Controller," 2016.

[36] B. Sonkoly, R. Szabo, D. Jocha, J. Czentye, M. Kind and F. J. Westphal, "UNIFYing Cloud and Carrier Network Resources: An Architectural View," *IEEE Global Communications Conference (GLOBECOM),* 2015.

[37] A. Lombardo, A. Manzalini, G. Schembra, G. Faraci, C. Rametta and V. Riccobene, "An open framework to enable NetFATE (Network Functions at the edge)," *1st IEEE Conference on Network Softwarization (NetSoft),* pp. 1-6, 2015.

[38] B. S. Networks, "The Floodlight Project," 2016.

[39] G. Cheng, H. Chen, H. Hu, Z. Wang and J. Lan, "Enabling network function combination via service chain instantiation," *Computer Networks,* vol. 92, no. 2, pp. 396-407, 2015.

[40] R. Vilalta, A. Mayoral, R. Muñoz, R. Casellas and R. Martínez, "Multi-tenant transport networks with SDN/NFV," *European Conference on Optical Communication (ECOC),* 2015.

[41] R. Munoz, R. Vilalta, R. Casellas, R. Martinez, T. Szyrkowiec, A. Autenrieth, V. Lopez and D. Lopez, "SDN/NFV orchestration for dynamic deployment of virtual SDN controllers as VNF for multi-tenant optical networks," *Optical Fiber Communications Conference and Exhibition (OFC),* 2015.

[42] J. Schulz-Zander, C. Mayer, B. Ciobotaru, S. Schmid and A. Feldmann, "OpenSDWN: Programmatic Control over Home and Enterprise," vol. 16, no. 1, pp. 1-12, 2015.

[43] J. Batalle, J. F. Riera, E. Escalona and J. A. Garcia-Espin, "On the Implementation of NFV over an OpenFlow Infrastructure: Routing Function Virtualization," *IEEE Future Networks and Services (SDN4FNS),* 2013.

[44] BT, Huawei, EZChip, AMD, Tilera, Altera, Broadcom, EANTC, Ixia, "- Network Intensive and Compute Intensive Hardware Acceleration," *Technical Report. ETSI - the European Telecommunications Standards Institute,* no. PoC#21 , 2015.

[45] J. Lai, Q. Fu and T. Moors, "Rapid IP Rerouting with SDN and NFV," *IEEE Global Communications Conference (GLOBECOM),* pp. 1-7, 2015.

[46] J. Costa-Requena, J. L. Santos, V. F. Guasch, K. Ahokas, G. Premsankar, S. Luukkainen, O. L. Pérez, M.

 © IoRL consortium 2018

U. Itzazelaia, I. Ahmad, M. Liyanage, M. Ylianttila and E. M. d. Oca, "SDN and NFV integration in generalized mobile network architecture," *European Conference on Networks and Communications (EuCNC),* p. 154–158, 2015.

[47] S. V. Rossem, W. Tavernier, B. Sonkoly, D. Colle, J. Czentye, M. Pickavet and P. Demeester, "Deploying elastic routing capability in an SDN/NFV-enabled environment," pp. 22-24, 2015.

[48] Y. D. Lin, P. C. Lin, C. H. Yeh, Y. C. Wang and Y. C. Lai, "An extended SDN architecture for network function virtualization with a case study on intrusion prevention," *EEE Network,* vol. 29, no. 3, pp. 48-53, 2015.

[49] Telecom Italia, Nokia Networks, EXFO, Coriant, Aalto University, "PoC#26 - Virtual EPC with SDN Function in Mobile Backhaul Networks," *echnical Report. ETSI - the European Telecommunications Standards Institute,* 2015.

[50] NTT, Cisco, HP, Juniper Networks, "PoC#2 - Service Chaining for NW Function Selection in Carrier Networks," *Technical Report. ETSI - the European Telecommunications Standards Institute,* 2014.

[51] F. Callegati, W. Cerroni, C. Contoli and F. Foresta, “Performance of intent-based virtualized network infrastructure management,” *IEEE International Conference on Communications (ICC),* pp. 1-6, 2017.

[52] J. Schulz-Zander, C. Mayer, B. Ciobotaru, S. Schmid and A. Feldmann, “Unified Programmability of Virtualized Network Functions and Software-Defined Wireless Networks,” *IEEE Transactions on Network and Service Management,* vol. 14, no. 4, pp. 1046 - 1060, 2017.

[53] Linux Foundation, “The OpenDayLight Platform,” 2016. [Online]. Available: http://www.OpenDayLight.org.

[54] Telenor, Vodafone, Hewlett Packard Enterprise, ImVision Tech, Mavenir, Redhat, Altiostar, “PoC#34 - SDN Enabled Virtual EPC Gateway,” *Technical Report. ETSI - the European Telecommunications Standards Institute,* 2016.

[55] R. Cziva, S. Jouet, K. J. S. White and D. P. Pezaros, "Container-based network function virtualization for software-defined networks," *IEEE Symposium on Computers and Communication (ISCC),* p. 415–420, 2015.

[56] R. Cziva and D. P. Pezaros, "Container Network Functions: Bringing NFV to the Network Edge," *IEEE Communications Magazine,* vol. 55, no. 6, pp. 24-31, 2017.

[57] R. Vilalta, A. Mayoral, V. Lopez, V. Uceda, R. Casellas, R. Martinez, R. Munoz, A. Aguado, J. Marhuenda, R. Nejabati, D.Simeonidou, N. Yoshikane, T. Tsuritani, I. Morita, T. Szyrkowiec and A. Autenrieth, "Peer SDN Orchestration: End-to-End Connectivity Service Provisioning Through Multiple Administrative Domains," *42nd European Conference on Optical Communication,* pp. 1-3, 2016.

[58] R. Casellas, R. Muñoz, R. Vilalta and R. Martínez, "Orchestration of IT/cloud and networks: From Inter-DC interconnection to SDN/NFV 5G services," *International Conference on Optical Network Design and Modeling (ONDM),* pp. 1-6, 2016.

[59] R. Vilalta, A. Mayoral, R. Casellas, R. Martínez and R. Munoz, "SDN/NFV orchestration of multi-technology and multi-domain networks in cloud/fog architectures for 5g services," *21st OptoElectronics and Communications Conference (OECC) held jointly with 2016 International Conference on Photonics in Switching (PS),* pp. 1-6, 2016.

[60] R. Martínez, A. Mayoral, R. Vilalta, R. Casellas, R. Muñoz, S. Pachnicke, T. Szyrkowiec and A. Autenrieth, "Integrated SDN/NFV orchestration for the dynamic deployment of mobile virtual backhaul networks over a multilayer (packet/optical) aggregation infrastructure," *IEEE/OSA Journal of Optical Communications and Networking,* vol. 9, no. 2, pp. 135-142, 2017.

[61] R. Muñoz, L. Nadal, R. Casellas, M. S. Moreolo, R. Vilalta, J. M. Fàbrega, R. Martínez, A. Mayoral and F. J. Vílchez, "The ADRENALINE testbed: An SDN/NFV packet/optical transport network and edge/core cloud platform for end-to-end 5G and IoT services," *European Conference on Networks and Communications (EuCNC),* pp. 1-5, 2017.

[62] R. Vilalta, A. Mayoral, R. Casellas, R. Martínez and R. Muñoz, "Experimental demonstration of distributed multi-tenant cloud/fog and heterogeneous SDN/NFV orchestration for 5G services," *European Conference on Networks and Communications (EuCNC),* pp. 52-56, 2016.

 © IoRL consortium 2018

[63] W. Shen, M. Yoshida, K. Minato and W. Imajuku, "vConductor: An enabler for achieving virtual network integration as a service," *IEEE Communications Magazine,* vol. 53, no. 2, p. 116–124, 2015.

[64] G. M. Saridis, S. Peng, Y. Yan, A. Aguado, B. Guo, M. Arslan, C. Jackson, W. Miao, N. Calabretta, F. Agraz, S. Spadaro, G. Bernini, N. Ciulli, G. Zervas, R. Nejabati and D. Simeonidou, "Lightness: A Function-Virtualizable Software Defined Data Center Network With All-Optical Circuit/Packet Switching," *Journal of Lightwave Technology,* vol. 34, no. 7, 2016.

[65] Project FloodLight , "http://www.projectfloodlight.org/floodlight/".

[66] Helios by NEC, , "http://www.nec.com/".

[67] Trema openflow controller framework, [Online]. Available: https://github.com/trema/trema.

[68] SNAC Repository,, [Online]. Available: https://github.com/bigswitch/snac.

[69] R. Sherwood, M. Chan and A. C. e. al., "Carving research slices out of your production networks with openflow," *ACM SIGCOMM Computer Communication Review,* vol. 40, no. 1, pp. 129-130, 2010.

[70] Project FloodLight, [Online]. Available: http://www.projectfloodlight.org/floodlight/.

[71] IRIS, "The Recursive SDN Openflow Controller by ETRI," [Online]. Available: http://openiris.etri.re.kr/.

[72] Z. Cai, A. L. Cox and T. S. E. Ng, "Maestro: a system for scalable openflow control," *Tech. Rep. TR10-08 Rice University Houston, Tex, USA,* 2010.

[73] Open vSwitch, [Online]. Available: http://openvswitch.org/support/.

[74] "The Linux Kernel Archives," [Online]. Available: https://www.kernel.org/.

[75] OpenFlow 1.3 switch., [Online]. Available: https://github.com/CPqD/ofsoftswitch13.

[76] Indigo, “Open source openflow switches,” [Online]. Available: http://www.openflowhub.org/display/Indigo/.

[77] Pica8, "http://pica8.com/products/".

[78] Pantou: Openflow 1.0 for openwrt, [Online]. Available: http://www.openflow.org/wk/index.php/.

[79] Node.js, “http://nodejs.org/”.

[80] OpenFlowJ, [Online]. Available: https://github.com/floodlight/loxigen/wiki/OpenFlowJ-Loxi.

[81] OpenFaucet. [Online]. Available: https://github.com/rlenglet/openfaucet.

[82] A. Networks, "CLOUD-NATIVE APP DELIVERY & SECURITY," [Online]. Available: https://www.a10networks.com/products/lightning-application-delivery-controller.

[83] Big vSwitch, [Online]. Available: http://www.bigswitch.com/sites/default/fles/sdnresources/bvsdatasheet.pdf.

[84] Brocade ADX Series, , "http://www.brocade.com/en/productsservices/sofware-networking/application-delivery-controllers".

[85] IBM RackSwitch G8264, [Online]. Available: http://www.redbooks.ibm.com/.

[86] HP OpenFlow Enabled Switches, [Online]. Available: https://www.hpe.com/uk/.

[87] Juniper Junos MX, EX, QFX Series, "http://www.juniper.net/techpubs/en US/junos15.1/topics/concept/virtual-chassis-exqfx-series-mixed-understanding.html".

[88] OpenNaas, "https://github.com/dana-i2cat/opennaas," [Online].

[89] OFERTIE, "http://www.i2cat.net/en/projects/ofertie," [Online].

[90] SODALES, "https://cordis.europa.eu/project/rcn/106389_en.html," [Online].

[91] I. A. J., R. E., A. G.-E. J. and D. F., "OpenNaaS-based networking solution for DC automated management," in *IEEE 3rd International Conference on Cloud Networking (CloudNet),*, Luxembourg, 2014.

[92] N. OpenStack, "https://wiki.OpenStack.org/wiki/Neutron," [Online].

[93] OpenDayLight VTN, "https://docs.OpenDayLight.org/en/stable-oxygen/user-guide/virtual-tenant-network-(vtn).html," [Online].

[94] FlowVisor, "https://www.sdxcentral.com/projects/on-lab-flowvisor/," [Online].

 © IoRL consortium 2018

[95] OpenVirtex Platform, "https://ovx.onlab.us/," [Online].

[96] OpenContrail Platform, "http://www.opencontrail.org/," [Online].

[97] Mininet, "http://mininet.org/".

[98] ns-3 emulator, [Online]. Available: https://www.nsnam.org/docs/release/3.13/models/html/openflow-switch.html.

[99] N. p. s. series, "https://www.necam.com/sdn/".

[100] OMNeT++, "https://omnetpp.org/".

[101] "SDN troubleshooting simulator," [Online]. Available: http://ucb-sts.github.com/sts/.

[102] NICE, "https://code.google.com/archive/p/nice-of/".

[103] OFTest, "http://archive.openflow.org/wk/index.php/OFTestTutorial".

[104] H. Mai, A. Khurshid, R. Agarwal, M. Caesar, P. B. Godfrey and S. T. King, "Debugging the data plane with anteater," in *ACM SIGCOMM Conference (SIGCOMM '11)*, Toronto, Canada, 2011.

[105] A. Khurshid, W. Zhou, M. Caesar and P. B. Godfre, "VeriFlow: verifying network-wide invariants in real time," in *1st ACM International Workshop on Hot Topics in Sofware Defned Networks (HotSDN '12)*, Helsinki, Finland, 2012.

[106] A. Wundsam, D. Levin, S. Seetharaman and A. Feldmann, "OFRewind: enabling record and replay troubleshooting for networks," *SENIX Conference on USENIX Annual Technical Conference (USENIXATC '11),* 2011.

[107] N. Handigol, B. Heller, V. Jeyakumar, D. Mazieres and N. McKeown, "Where is the debugger for my sofware-defned network?," *1st ACM International Workshop on Hot Topics in Sofware Defned Networks (HotSDN '12),* pp. 55-60, 2012.

[108] Wireshark, [Online]. Available: https://www.wireshark.org/.

[109] ETSI, "Network Operator Perspectives on NFV priorities for 5G," *White Paper,* 2012.

[110] ETSI, "Network Functions Virtualisation (NFV) - Network Operator Perspectives on Industry Progress," *White Paper,* 2015.

[111] ETSI, "Network Functions Virtualisation (NFV) - Architectural Framework," *ETSI GS NFV 002 V1.2.1,* 2014.

[112] XEN Project, [Online]. Available: https://www.xenproject.org/.

[113] Linux KVM, *https://www.linux-kvm.org.*

[114] Dell, VMware, *https://www.vmware.com/.*

[115] M. Mahalingam, D. Dutt and K. D. e. al., "VXLAN, A Framework for Overlaying Virtualized Layer 2 Networks Over Layer 3 Networks," *Internet Engineering Task Force,* 2011.

[116] M. Sridharan, K. Duda, I. Ganga, A. Greenberg, G. Lin and M. P. e. al., "NVGRE: network virtualization using generic routing encapsulation, Internet Engineering Task Force," 2011.

[117] ETSI, "Network Functions Virtualisation (NFV)," *Ecosystem Report on SDN Usage in NFV Architectural,* 2015.

[118] ETSI, "Network Functions Virtualisation (NFV) - Management and Orchestration," *ETSI GS NFV-MAN 001 V1.1.1,* 2014.

[119] T-NOVA Project, "T-NOVA: Network Functions-as-a-Service (NFaaS) over Virtualized Infrastructures," 2015.

[120] A. Császár, W. John, M. Kind, C. Meirosu, G. Pongrácz, D. Staessens, A. Takács and F. J. Westphal, "Unifying Cloud and Carrier Network: EU FP7 Project UNIFY," *IEEE/ACM,* vol. Utility and Cloud Computing (UCC), no. 6th International Conference, 2013.

[121] SONATA Consortium, "SONATA NFV," 2016.

[122] Project Vital, "VIrtualized hybrid satellite-TerrestriAl systems for resilient and fLexible future networks," 2016.

[123] Project 5GEx, "https://5g-ppp.eu/5gex/," [Online].

 © IoRL consortium 2018

[124] H2020 5G-TRANSFORMER Project, "5G Mobile Transport Platform for Verticals," 2017.

[125] H. Karl, S. Drxler, M. Peuster, A. Galis, M. Bredel, A. Ramos, J. Martrat, M. S. Siddiqui, S. v. Rossem, W. Tavernier and G. Xilouris, "Developments for network function virtualisation: an architectural approach," *IEEE Transactions on Emerging Telecommunications Technologies,* vol. 27, no. 9, p. 1206–1215, 2016.

[126] S. Draxler, H. Karl, M. Peuster, H. R. Kouchaksaraei, M. Bredel, J. Lessmann, T. Soenen, W. Tavernier, S. Mendel-Brin and G. Xilouris, "SONATA: Service programming and orchestration for virtualized software networks," *IEEE International Conference on Communications Workshops (ICC Workshops),* p. 973–978, 2017.

[127] SONATA Consortium, "D2.2 Architecture Design," *Tech. Rep,* 2015.

[128] SONATA Consortium, "D2.3 Updated Requirements and Architecture Design," *Tech. Rep.,* 2016.

[129] VITAL Project, "D2.3 System Architecture: Final Report," 2017.

[130] NFV, ETSI Industry Specification Group (ISG), "ETSI GS NFV 002 V1.1.1: Network Functions Virtualisation (NFV) Architectural Framework," *Tech. Rep.,* 2013.

[131] A. Francescon, G. Baggio, R. Fedrizzi, R. Ferrusy, I. G. B. Yahiaz and R. Riggio, "X-MANO: Cross-domain management and orchestration of network services," *IEEE Conference on Network Softwarization (NetSoft),* 2017.

[132] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy and V. Sekar, "Making Middleboxes Someone else's Problem: Network Processing As a Cloud Service," *ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'12),* pp. 13-24, 2012.

[133] R. Cziva, S. Jouet and D. P. Pezaros, "GNFC: Towards network function cloudification," *IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN),* pp. 142-148.

[134] Computing, Rackspace Cloud, "OpenStack Open Source Cloud Computing Software," 2016.

[135] J. Deng, H. Hu, H. Li, Z. Pan, K. C. Wang, G. J. Ahn, J. Bi and Y. Park, "VNGuard: An NFV/SDN combination framework for provisioning and managing virtual firewalls," pp. 107-114, 2015.

[136] CloudLab, *https://www.cloudlab.us/.*

[137] ETSI, "Network Functions Virtualisation (NFV) - Use Cases. ETSI GS NFV 001 V1.1.1," 2013.

[138] I. Cerrato, A. Palesandro, F. Risso, M. Sune, V. Vercellone and H. Woesner, "Toward dynamic virtualized network services in telecom operator networks," *Computer Networks,* vol. 92, no. 2, pp. 380-395, 2015.

[139] J. Soares, M. Dias, J. Carapinha, B. Parreira and S. Sargento, "Cloud4NFV: A platform for Virtual Network Functions," *IEEE 3rd International Conference on Cloud Networking (CloudNet),* p. 288–293, 2014.

[140] J. Soares, C. Gonçalves, B. Parreira, P. Tavares, J. Carapinha, J. P. Barraca, R. L. Aguiar and S. Sargento, "Toward a telco cloud environment for service functions," *IEEE Communications Magazine,* vol. 53, no. 2, p. 98–106, 2015.

[141] G. Carella, J. Yamada, N. Blum, C. Lück, N. Kanamaru, N. Uchida and T. Magedanz, "Cross-layer service to network orchestration," *IEEE International Conference on Communications (ICC),* 2015.

[142] FOKUS, Fraunhofer, "OpenSDNCore - Reasearch and testbed for the carrier-grade nfv/sdn environment," 2016.

[143] L. Suresh, J. Schulz-Zander, R. Merz, A. Feldmann and T. Vazao, "Towards Programmable Enterprise WLANS with Odin," New York USA, 2015.

[144] J. Vestin and A. Kassler, "QoS enabled WiFi MAC layer processing as an example of a NFV service," in *1st IEEE Conference on Network Softwarization (NetSoft)*, 2015.

[145] P. Dely, J. Vestin, A. Kassler, N. Bayer, H. Einsiedler and C. Peylo, "CloudMAC: An OpenFlow based architecture for 802.11 MAC layer processing in the cloud," in *IEEE Globecom Workshops*, 2012.

[146] W. Kim, "Toward network function virtualization for cognitive wireless mesh networks: a TCP case

 © IoRL consortium 2018

study," *EURASIP Journal on Wireless Communications and Networking,* pp. 1-16, 2015.

[147] A. M. Medhat, G. Carella, J. Mwangama and N. Ventura, "Multi-tenancy for Virtualized Network Functions," *1st IEEE Conference on Network Softwarization (NetSoft),* 2015.

[148] J. Ordonez-Lucena, P. A. D. Lopez, J. J. Ramos-Munoz, J. Lorca and J. Folgueira, "Network Slicing for 5G with SDN/NFV: Concepts, Architectures, and Challenges," *IEEE Communications Magazine,* vol. 55, no. 5, pp. 80-87, 2017.

[149] I. F. Akyildiz, S.-C. Lin and P. Wang, *Computer Networks,* vol. 93, no. 1, pp. 66-79, 2015.

[150] J. Mwangama, N. Ventura, A. Willner, Y. Al-Hazmi, G. Carella and T. Magedanz, "Towards Mobile Federated Network Operators," *1st IEEE Conference on Network Softwarization (NetSoft).*

[151] A. Mayoral, R. Vilalta, R. Casellas, R. Martinez and R. Munoz, "Multi-tenant 5G Network Slicing Architecture with Dynamic Deployment of Virtualized Tenant Management and Orchestration (MANO) Instances," *42nd European Conference on Optical Communication,* pp. 1-3, 2016.

[152] R. Muñoz, L. Nadal, R. Casellas, M. S. Moreolo, R. Vilalta, J. M. Fàbrega, R. Martínez, A. Mayoral and F. J. Vílchez, "The ADRENALINE testbed: An SDN/NFV packet/optical transport network and edge/core cloud platform for end-to-end 5G and IoT services," *European Conference on Networks and Communications (EuCNC),* pp. 1-5, 2017.

[153] ETSI, "Mobile Edge Computing (MEC): Technical Requirements," 2016.

[154] ETSI, "Mobile Edge Computing (MEC): Framework and Reference Architecture," 2016.

[155] J. L. M. M. Rodrigo Roman, "Mobile Edge Computing: A Survey and Analysis of Security Threats and Challenges," 2016.

[156] 5G PPP Architecture Working Group, "5G Vision," *Technical Report,* 2015.

[157] EU SELFNET Project H2020, "Framework for Self-Organized Network Management in Virtualized and Software Defined Networks," 2016.

[158] P. Neves, R. Calé, M. Costa, G. Gaspar, J. Alcaraz-Calero, Q. Wang, J. Nightingale, G. Bernini, G. Carrozzo, Á. Valdivieso, L. J. G. Villalba, M. Barros, A. Gravas, J. Santos, R. Maia and R. Preto, "Future mode of operations for 5G – The SELFNET approach enabled by SDN/NFV," *Computer Standards & Interfaces,* vol. 54, no. 4, pp. 229-246, 2017.

[159] P. Neves, R. Calé, M. R. Costa, C. Parada, B. Parreira, J. Alcaraz-Calero, Q. Wang, J. Nightingale, E. Chirivella-Perez, W. Jiang, H. D. Schotten, K. Koutsopoulos, A. Gavras and M. J. Barros, "The SELFNET Approach for Autonomic Management in an NFV/SDN Networking Paradigm," *Journal in Distributed Sensor Networks,* 2016.

[160] Y. D. Lin, P. C. Lin, C. H. Yeh, Y. C. Wang and Y. C. Lai, "An extended SDN architecture for network function virtualization with a case study on intrusion prevention," *IEEE Network,* vol. 29, no. 3, pp. 48-53, 2015.

[161] ETSI, "Open Source MANO".

[162] A. Israel, A. Hoban, A. Tierno, F. Salguero, G. Blas, K. Kashalkar, M. Ceppi, M. Shuttleworth, M. Harper, M. Marchetti, R. Velandy, S. Almagia and V. Little, "OSM Release Three: A Technical Overview," *ETSI, Tech. Rep.,* Oct 2017.

[163] Linux Foundation, "ONAP – Open Network Automation Platform," 2017.

[164] Linux Foundation, "Open Orchestrator".

[165] AT&T, "Enhanced Control , Orchestration , Management & Policy," *Architecture White Paper,* 2016.

[166] C. N. Digest, "The Open Network Automation Platform looks like a turning point for telecom architecture," 2017.

[167] L. Reading, "ONAP Makes Splashy ONS Debut," 2017.

[168] A. Francescon, G. Baggio, R. Fedrizzi, R. Ferrusy, I. G. B. Yahiaz and R. Riggio, "X-MANO: Cross-domain management and orchestration of network services," *IEEE Conference on Network Softwarization (NetSoft),* pp. 1-5, 2017.

[169] Fraunhofer and T. Berlin, "Open Baton: An open source reference implementation of the ETSI Network Function Virtualization MANO specification," 2017.

 © IoRL consortium 2018

[170] "Apache ARIA TOSCA ORCHESTRATION ENGINE," 2017. [Online]. Available: http://ariatosca.incubator.apache.org/.

[171] Cloudify, “A Primer on Project ARIA - Simple, Open Source TOSCA-Based Orchestration Engine,” 2016.

[172] J. F. Riera, J. Batall, J. Bonnet, M. Das, M. McGrath, G. Petralia, F. Liberati, A. Giuseppi, A. Pietrabissa, A. Ceselli, A. Petrini, M. Trubian, P. Papadimitrou, D. Dietrich, A. Ramos, J. Melin, G. Xilouris, A. Kourtis, T. Kourtis and E. K. Markakis, “Tenor: Steps towards an orchestration platform for multi-pop nfv deployment,” *IEEE NetSoft Conference and Workshops (NetSoft),* p. 243–250, 2016.

[173] OpenStack Foundation, “Tacker - OpenStack,” 2016. [Online]. Available: https://wiki.OpenStack.org/wiki/Tacker.

[174] “Radically Simplifying Multi-Cloud Orchestration,” [Online]. Available: http://cloudify.co/.

[175] L. Peterson, S. Baker, M. D. Leenheer, A. Bavier, S. Bhatia, M. Wawrzoniak, J. Nelson and J. Hartman, “Xos: An extensible cloud operating system,” *in Proceedings of the 2Nd International Workshop on Software-Defined Ecosystems,* pp. 23-30, 2015.

[176] “XOS: A Service Abstract Layer for CORD”.

[177] Gohan, “Gohan - REST-based api server to evolve your cloud service very rapidly,” 2015.

[178] UNIFY, "Unifying Cloud and Carrier Networks," 2013.

[179] B. Sonkoly, J. Czentye, R. Szabo, D. Jocha, J. Elek, S. Sahhaf, W. Tavernier and F. Risso, "Multi-domain service orchestration over networks and clouds: A unified approach," *ACM SIGCOMM Computer Communication Review,* vol. 45, no. 4, p. 377–378, 2015.

[180] FFmpeg, "https://www.ffmpeg.org/".

[181] Qemu, "https://www.qemu.org/".

[182] "Docker," [Online]. Available: https://www.docker.com/.

[183] Arduino, [Online]. Available: https://www.arduino.cc/.

 © IoRL consortium 2018