# ETSI GS NFV-EVE 005 V1.1.1 (2015-12)


# Network Functions Virtualisation (NFV); Ecosystem; Report on SDN Usage in NFV Architectural Framework

***Disclaimer***

Reference
DGS/NFV-EVE005

Keywords
NFV, SDN

***ETSI***

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

***Important notice***

The present document can be downloaded from:
http://www.etsi.org/standards-search

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the only prevailing document is the print of the Portable Document Format (PDF) version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status.
Information on the current status of this and other ETSI documents is available at
http://portal.etsi.org/tb/status/status.asp

If you find errors in the present document, please send your comment to one of the following services:
https://portal.etsi.org/People/CommiteeSupportStaff.aspx

***Copyright Notification***

# Contents

# Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://ipr.etsi.org).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

# Foreword

This Group Specification (GS) has been produced by ETSI Industry Specification Group (ISG) Network Functions Virtualisation (NFV).

# Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

# 1 Scope

The present document identifies the most common design patterns for using SDN in an NFV architectural framework. It also identifies potential recommendations to be fulfilled by the entities that perform the integration.

# 2 References

## 2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the reference document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

Not applicable.

## 2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the reference document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ETSI GS NFV-INF 005 (V.1.1.1) - (12-2014): "Network Functions Virtualisation (NFV); Infrastructure; Network Domain".

[i.2] ETSI GS NFV-MAN 001 (V1.1.1) - (12-2014): "Network Functions Virtualisation (NFV); Management and Orchestration".

[i.3] ETSI GS NFV 002 (V1.2.1) - (12-2014): "Network Functions Virtualisation (NFV); Architectural Framework".

[i.4] ETSI GS NFV 003 (V1.2.1) - (12-2014): "Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV".

[i.5] ETSI GS NFV-SWA 001: "Network Functions Virtualisation (NFV); Virtual Network Functions Architecture".

[i.6] Open Networking Foundation TR-502: "SDN Architecture", Issue 1.0, June 2014.

[i.7] Metro Ethernet Forum (V1.1) (February 2012): "Carrier Ethernet for Delivery of Private Cloud Services".

[i.8] Recommendation ITU-T Y.3300 (06-2014): "Framework of software-defined networking".

[i.9] IETF SFC Architecture.

NOTE: Available at http://tools.ietf.org/pdf/draft-ietf-sfc-architecture-07.pdf.

[i.10] IETF Service Function Chain Extension Architecture.

NOTE: Available at https://tools.ietf.org/pdf/draft-gu-sfc-extend-architecture-00.pdf.

[i.11] IETF I2NSF Interface to Network Security Functions.

NOTE: Available at http://datatracker.ietf.org/doc/charter-ietf-i2nsf/.

[i.12] Floodlight®: http://www.projectfloodlight.org/floodlight/.

[i.13] OpenDaylight®: http://www.opendaylight.org/.

[i.14] OpenContrail: http://www.opencontrail.org/.

[i.15] ONOS®: http://onosproject.org/.

[i.16] Ryu: http://osrg.github.io/ryu/.

[i.17] Midonet®: https://www.midonet.org/.

[i.18] IETF RFC 5493 (April 2009): "Requirements for the Conversion between Permanent Connections and Switched Connections in a Generalized Multiprotocol Label Switching (GPMLS) Network".

NOTE: Available at https://tools.ietf.org/html/rfc5493.

[i.19] IETF RFC 6830 (January 2013): "The Locator/ID Separation Protocol (LISP)".

[i.20] IETF RFC 7285 (September 2014): "Application-Layer Traffic Optimization (ALTO) Protocol".

[i.21] IETF RFC 7348: "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks".

NOTE: Available at http://www.rfc-editor.org/rfc/rfc7348.txt.

[i.22] IETF RFC 7426 (January 2015): "Software-Defined Networking (SDN): Layers and Architecture Terminology".

NOTE: Avaiable at http://www.rfc-editor.org/rfc/rfc7426.txt.

[i.23] IETF RFC 7432: "BGP MPLS-Based Ethernet VPN".

NOTE: Avaiable at https://tools.ietf.org/rfc/rfc7432.txt.

[i.24] ONOS® Developer's Guide.

NOTE: Available at https://wiki.onosproject.org/display/ONOS/Developer's+Guide.

# 3 Definitions and abbreviations

## 3.1 Definitions

For the purposes of the present document, the terms and definitions given in ETSI GS NFV 003 [i.4] and the following apply:

**Openflow:** trademark from the Open Networking Foundation (ONF) for an SDN standard protocol which enables remote programming of the forwarding plane

## 3.2 Abbreviations

For the purposes of the present document, the abbreviations given in ETSI GS NFV 003 [i.4] and the following apply:

| | |
|---|---|
| DC | Data Center |
| EM | Element Manager |
| ETSI | European Telecommunications Standards Institute |
| FIB | Forwarding Information Base (OSI Layer 3 forwarding table) |

| | |
|---|---|
| HSDN | Hierarchical SDN |
| ISG | Industry Standards Group |
| ITU-T | International Telecommunication Union-Standardization Sector |
| LFIB | Label Forwarding Information Base (MPLS forwarding table) |
| MPLS | Multi-Protocol Label Switching |
| OAM | Operation and Management |
| NCT | Network Connectivity Topology |
| NE | Network Element |
| NFP | Network Forwarding Path |
| POP | Point of Presence |
| SDN | Software Defined Networks |
| SFC | Service Function Chaining |
| SP | Service Provider |
| VTN | Virtual Tenant Network |

# 4 Overview of SDN in the NFV architectural framework

## 4.1 Introduction

ETSI ISG NFV has defined an NFV architectural framework operating on the basis of the principle of separating network functions from the hardware they run on by using virtual hardware abstraction. The major components in this framework are (From ETSI GS NFV 002 [i.3]):

- Network Functions Virtualisation Infrastructure (NFVI): subsystem which encompasses Compute, Network and Storage resources, i.e. the totality of all hardware and software components that build up the environment in which VNFs are deployed.
- Management and Orchestration (MANO): subsystem which includes the Network Functions Virtualisation Orchestrator (NFVO), the Virtualised Infrastructure Manager (VIM) and Virtual Network Function Manager (VNFM).
- Virtual Network Functions (VNFs): deployed in the NFVI.

The present document provides an overview of SDN in relation to this ETSI NFV architectural framework as well as a summary of current industry work including a comparison of network controllers and PoCs including NFV and SDN.

## 4.2 SDN scope

Recommendation ITU-T Y.3300 [i.8] defines SDN as 'a set of techniques that enables to directly program, orchestrate, control and manage network resources, which facilitates the design, delivery and operation of network services in a dynamic and scalable manner'. Although this broad definition translates in many different ways in terms of specifications and implementations, most SDN-labelled solutions relocates the control of network resources to dedicated network elements, namely SDN controllers and might be mapped to the 3-layers reference model depicted in figure 1.

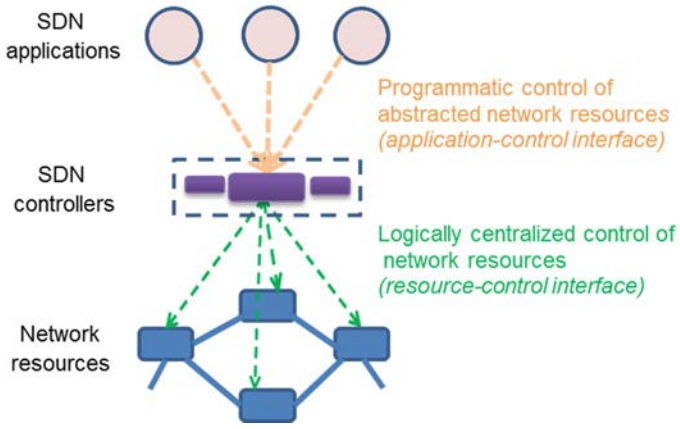



**Figure 1: Concept of SDN (from Recommendation ITU-T Y. 3300 [i.8])**

While there are other possible models, the present document focuses on the model described in figure 1.

Within the NFV architectural framework [i.3], SDN solutions might be used in the infrastructure domain, in the tenant domain or both.

When used in the infrastructure domain, the SDN controller acts as a Network Controller, as per ETSI GS NFV-MAN 001 [i.2]. In the present document SDN refers to software control of physical or virtual network resources that use standard interfaces (open APIs) to facilitate interoperability and evolution in a multi-vendor environment.

The SDN controller is not necessarily a stand-alone physical entity, e.g. a software component(s) of the VIM. Ideally, when the SDN technology is used in the infrastructure domain, the VIM, the SDN controller, and the Network Resources (physical or virtual) form a hierarchy for delivering connectivity services. In some cases, multiple SDN controllers form a hierarchy across management and resources, depending on the placement of the functionality. The SDN controller responsibility includes very specific control functions, interfacing with management agents responsible for control and management functions.

NOTE: SDN applications, SDN controllers and SDN resources come from one or different vendors, and are typically implemented in different VNF. The NFVO while on-boarding these VNFs would want to know that they are related and able to communicate with each other.

There is a large landscape of SDN architectures (NFV and non-NFV) and SDN controller functionality encompassing a variety of capabilities e.g. service negotiation, network element provisioning, and control of resources. This broader interpretation of SDN is beyond the scope of the present document.

## 4.3 SDN in the NFV architectural framework

### 4.3.1 General

As stated above, the focus of the present document lays on how network services and associated resources implemented, according to an SDN architecture, might be integrated with the NFV architectural framework by identifying possible design patterns and associated requirements. Many technical and non-technical issues need to be formulated and answered regarding all the functional entities that constitute this integrated architectural framework, such as:

- The position of the SDN resources.
- The position of the SDN controller.
- The softwarization & virtualisation of the various SDN entities.
- The interaction between the Element Managers, VNF Managers, SDN controllers and SDN applications that become enabled VNFs.
- The hierarchy of SDN networks.
- The position of the overlay SDN networks.
- Others.

### 4.3.2 SDN management plane

IETF RFC 7426 [i.22] discusses the distinction between control and management plane in an SDN environment. In brief the control plane is mostly responsible for making decisions on how packets are forwarded by one or more network resources and pushing such decisions down to the network resources for execution, whilst the management plane is mostly responsible for monitoring, configuring, and maintaining network devices, e.g. making decisions regarding the state of a network resources.

IETF RFC 7426 [i.22] also discusses the differentiation between these two planes by identifying their characteristics. It does so by showcasing four characteristics: (1) timescale, i.e. how often and how fast resources are configured and state persistence; (2) longevity of the state; (3) locality, i.e. centralized or distributed; (4) insights from the CAP theorem, e.g. the control plane is available, whilst the management plane is consistent.

Actually the CAP theorem (https://en.wikipedia.org/wiki/CAP_theorem) states that it is impossible for a distributed computer system to simultaneously provide consistency, availability and partition tolerance.

NOTE: The distinction between the control and management plane has become somewhat muddled due to the logical centralization of the control plane which is more of the domain of the management plane.

### 4.3.3 Position of SDN resources in an NFV architectural framework

The first entities to be considered are the SDN resources. Multiple scenarios might be envisaged for their actual location or for their images:

- Case a: physical switch or router
- Case b: virtual switch or router
- Case c: e-switch, software based SDN enabled switch in a server NIC
- Case d: switch or router as a VNF

In case d the resource might be logically part of the NFVI or belong to an independent tenant's domain. An example of case d is illustrated in NFV PoC#14 (clause B.5). This PoC has demonstrated the usage of SDN in an NFV environment by splitting the Service Gateway (SGW) and Packet Gateway (PGW) of the Long Term Evolution (LTE) architecture into a control and data plane for each, using an open interface, in this case IETF's ForCES. PoC#14 demonstrates that the data plane functionality might be deployed as VNF and controlled as a network resource.

Figure 2 shows the functional entities in the NFV architectural framework [i.3] for the scenarios identified above.

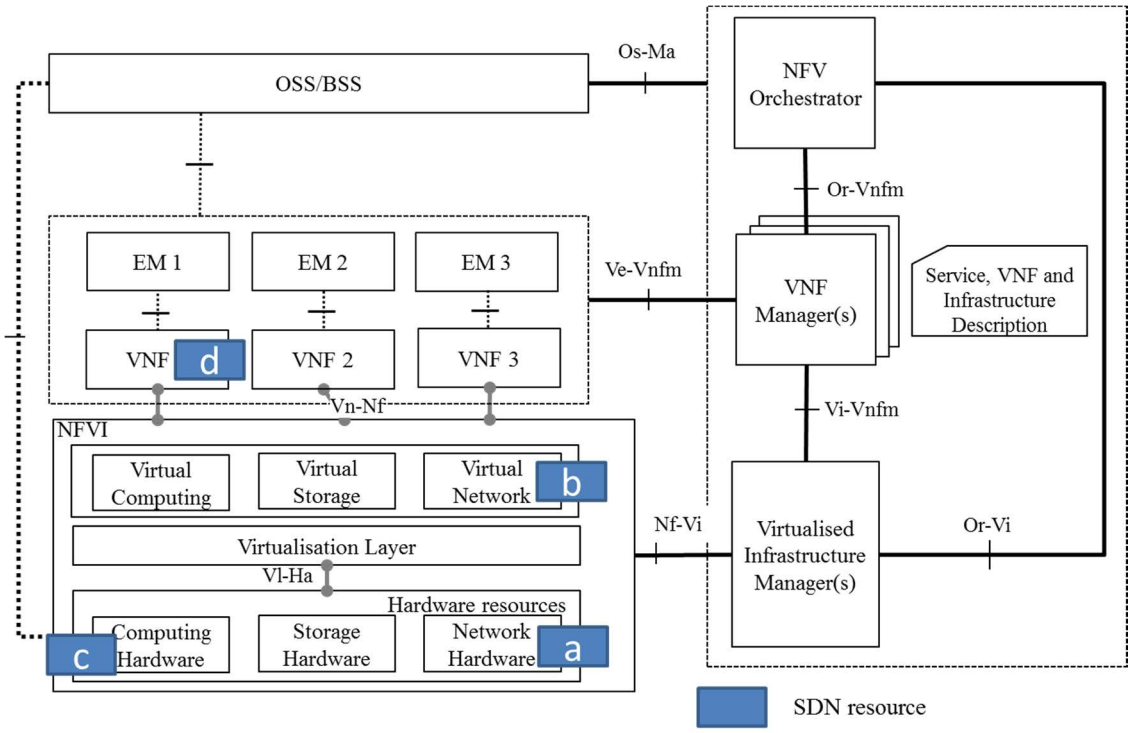



**Figure 2: Possible SDN Resource Locations in the NFV Architectural Framework (adaptation from [i.3])**

### 4.3.4 Position of the SDN controller in an NFV architectural framework

The second entity in this context is the SDN controller, which interfaces with SDN network resources via the Resource Control Interface. One SDN controller might interface with multiple SDN network resources.

Multiple scenarios exist to illustrate the possible locations of an SDN Controller in the context of an NFV framework:

- Case 1: the SDN controller is merged with the Virtualised Infrastructure Manager functionality.

- Case 2: the SDN controller is Virtualised as a VNF.
- Case 3: the SDN controller is part of the NFVI and is not a VNF.
- Case 4: the SDN controller is part of the OSS/BSS.
- Case 5: the SDN controller is a PNF.

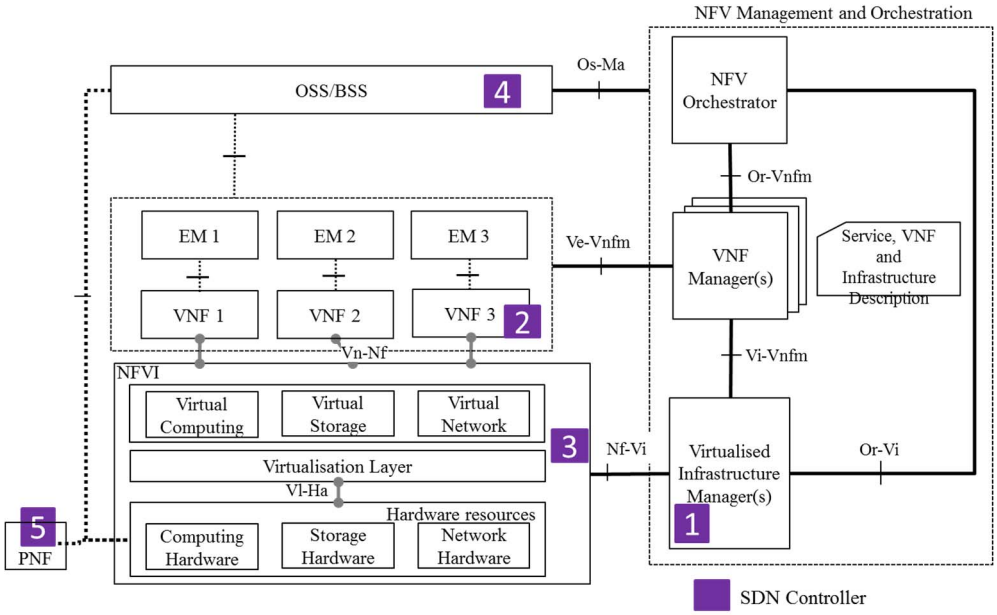



**Figure 3: Possible SDN Controller Locations in the NFV Architectural Framework**

Case 1: SDN controller functionality merged with the VIM functionality, in such case the two functions are not distinguishable.

Case 2: SDN controller as a VNF is typically the case of an SDN controller Virtualised as a VNF itself, or being part of a VNF. This VNF might be logically part of the NFVI and therefore belong to a special infrastructure tenant or belong to an independent tenant.

Case 3: SDN controller in the NFVI is a classic case of SDN controller for the network connectivity in the NFVI, where the SDN controller is not implemented as a VNF.

Case 4: SDN controller part of the OSS, is illustrated in clause 5.7, figures 27 and 28 as the tenant SDN controller.

Case 5: SDN controller as a PNF - while this case exists, it has not been studied in the rest of the present document.

### 4.3.5 Position of SDN applications in an NFV architectural framework

The third entity to be considered is the SDN application which interfaces with the SDN controller. An SDN application might interface with multiple SDN controllers. Multiple case scenarios might be envisioned, for the position of the SDN applications in the NFV architectural framework, such as:

- Case i: as part of a PNF
- Case ii: as part of the VIM
- Case iii: Virtualised as a VNF
- Case iv: as part of an EM
- Case v: as part of the OSS/BSS

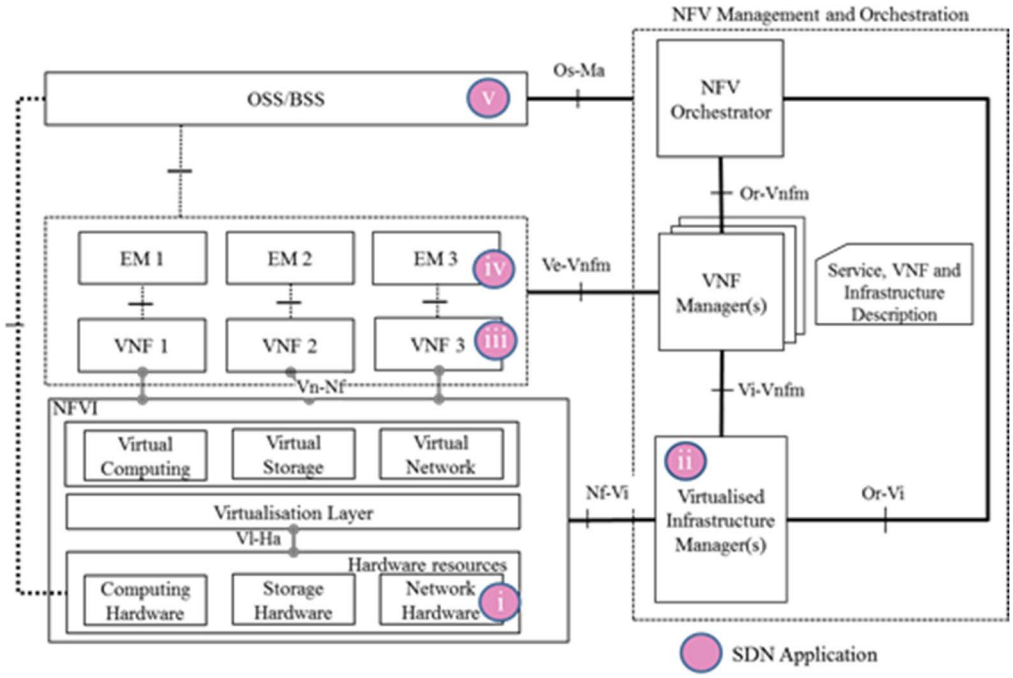



**Figure 4: Possible SDN Application Locations in the NFV Architectural Framework**

The positions of the SDN applications are further expanded:

Case i: the network hardware might be a physical appliance talking to an SDN controller, or a complete solution including multiple SDN components, such as SDN controller + SDN application for instance.

Case ii: the VIM might be an application interfacing with an SDN controller in the NFVI - for instance OpenStack Neutron as a VIM interfacing with an SDN controller in the NFVI.

Case iii: the SDN application might be a VNF talking to an SDN controller, being Virtualised or not. For instance a PCRF VNF might talk to an SDN controller for some policy management for traffic steering.

Case iv: the SDN application might be an element manager interfacing with an SDN controller to collect some metrics or configure some parameters.

Case v: the SDN application might be an application interfacing with an SDN controller for instance in the OSS-BSS for tenant SDN service definitions.

## 4.4 SDN controller interfaces in the NFV architectural framework

### 4.4.1 Introduction

The SDN controller entity has several possible interfaces. The interfaces are shown in figure 5.

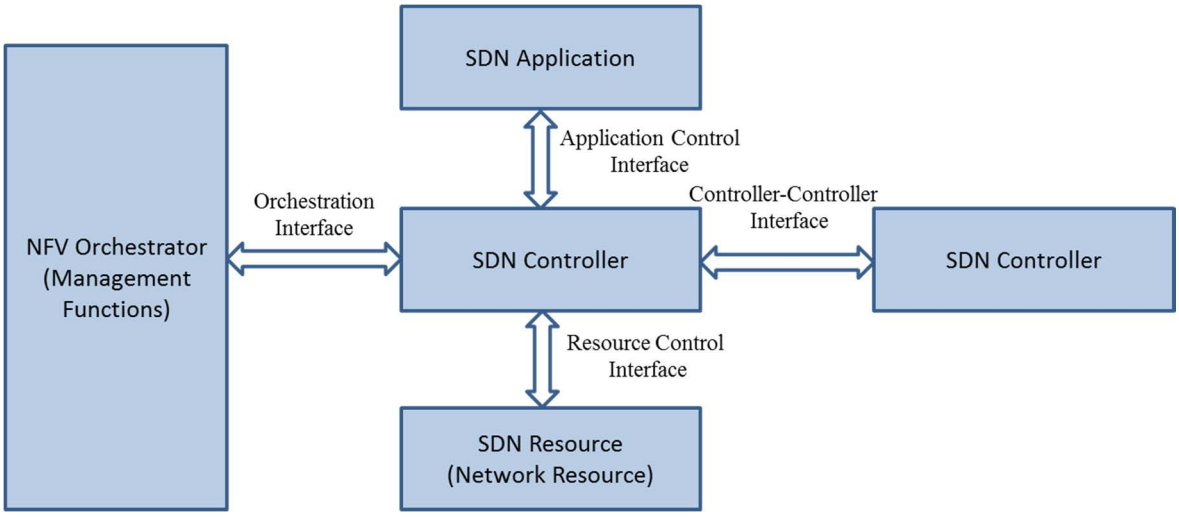



**Figure 5: SDN Controller Interfaces**

Figure 5 is based on figure 1, but with names for the interfaces so these interface names can be referenced.

As described above in clause 4.2 (and taken from Recommendation ITU-T Y.3300 [i.8]), an SDN controller comprises the following elements:

- Application Control Interface - interface between an SDN controller and an SDN application - it provides an application programmatic control of abstracted network resources.
- Resource Control Interface - Interface between an SDN controller and SDN resources - it is used to control network resources.
- Orchestration Interface - interface between an SDN controller and an NFV Orchestrator:
  - It might need to pass information between the 2 entities, such as topology information in both directions.
  - The interface might be an indirect interface.
  - The same interface might also be used between an SDN application and an NFV Orchestrator.
- Controller-Controller Interface - interface between SDN controllers:
  - It might need to pass information between SDN controllers either in the same hierarchy or in different hierarchies.

The different SDN controller interfaces need to be supported in an NFV environment.

Some open source projects, such as ODL or ONOS® (see note), have also defined a Controller-Controller interface for SDN controllers to interact with each other. This might be exploited when federation of SDN controllers is being used for scalability reasons for instance. Today these interfaces remain specific to a given controller implementation and are not open to be used across different types of SDN controllers.

NOTE: "ONOS is the trade name of a collaborative open source project from The Linux Foundation. This information is given for the convenience of users of the present document and does not constitute an endorsement by ETSI of the product named. Equivalent products may be used if they can be shown to lead to the same results".

Also some entities, typically ONF and ODL, have defined Application Control interfaces such as the Intent NorthBound Interface.

The Intent NorthBound Interface interactions between applications and controllers are run-time (i.e. real-time) negotiations between an application layer that defines a network service and a controller layer that implements that service. The application layer network service defines the "what" of a given service, with some constraints on the requested service implementation or properties (SLA, latency, performance, etc.). Negotiations start by a set of interactions that begin with a service request by the application to the controller and terminate with agreement or not by the controller to deliver the requested service within the defined constraints, and by the application layer to accept delivery of that network service. Typically an Intent NorthBound interaction is a negotiation with four stages:

- Request
- Response
- Selection
- Confirmation

It defines the "what": substance (e.g. "get me from A to B") with constraints (e.g. "before time C").

From an SDN controller perspective, Intent NorthBound Interface functions might be implemented as a controller-associated abstraction between applications and controller, or directly within/as part of the controller itself.

From an NFV architectural framework perspective, the Intent NorthBound Interface could be considered as the Application Control Interface provided by the SDN controller if that layer is embedded in the SDN controller, or it could be considered as an SDN application if it seats on top of the SDN controller. As an SDN application it could be a VNF itself.

The diagrams and tables below show the different combinations between the different components. The analysis has been divided in 2 cases to simplify the diagrams and tables: SDN controller and SDN application, then SDN controller and SDN resources.

The cases that were considered irrelevant have been removed, again to simplify the diagrams and tables, and focus on the most relevant cases.

## 4.4.2 SDN resource control interface options

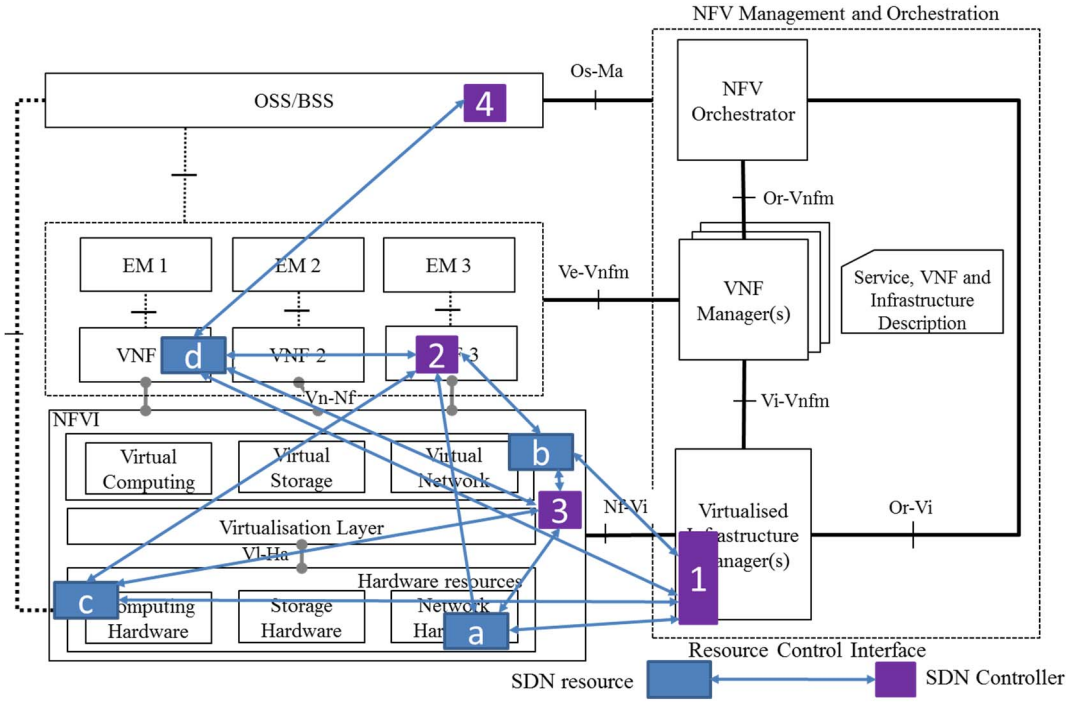



**Figure 6: SDN Resource Control Interface Options in NFV**

Figure 6 shows the different implementation options for the SDN Resource Control Interface. The NFV environment would support the different implementation options.

**Table 1: A List of SDN Resource Control Interface Options in NFV**

| I/F Ref. | SDN Controller Location | SDN Resource Location | Comments |
|---|---|---|---|
| 1a | 1. VIM | a. NFVI (Network Hardware) | |
| 1b | 1. VIM | b. Virtual Network | |
| 1c | 1. VIM | c. Computing Hardware | |
| 1d | 1. VIM | d. VNF | Only if the VNF logically belongs to the NFVI |
| 2a | 2. VNF | a. NFVI (Network Hardware) | Only if the VNF logically belongs to the NFVI |
| 2b | 2. VNF | b. Virtual Network | Only if the VNF logically belongs to the NFVI |
| 2c | 2. VNF | c. Computing Hardware | Only if the VNF logically belongs to the NFVI, |
| 2d | 2. VNF | d. VNF | No need to specify VNF-VNF interface, |
| 3a | 3. NFVI | a. NFVI (Network Hardware) | Internal NFVI interface |
| 3b | 3. NFVI | b. Virtual Network | Internal NFVI interface |
| 3c | 3. NFVI | c. Computing Hardware | Internal NFVI interface |
| 3d | 3. NFVI | d. VNF | Only if the VNF logically belongs to the NFVI, Not common. |
| 4d | 4. OSS/BSS | d. VNF | Applicable to the OSS/BSS of a VNF tenant |

There are several suggested SDN Resource Control Interfaces such as OpenFlow. NFV might optionally use some of these suggested SDN Resource Control Interfaces. The list in table 1 suggests that the same SDN Resource Control Interface be used in various implementation options. Some of the interfaces listed might be indirect interfaces. This means that the SDN Resource Control Interface between the SDN controller and the SDN resource might flow through other NFV entities.

### 4.4.3 SDN controller orchestration interface options

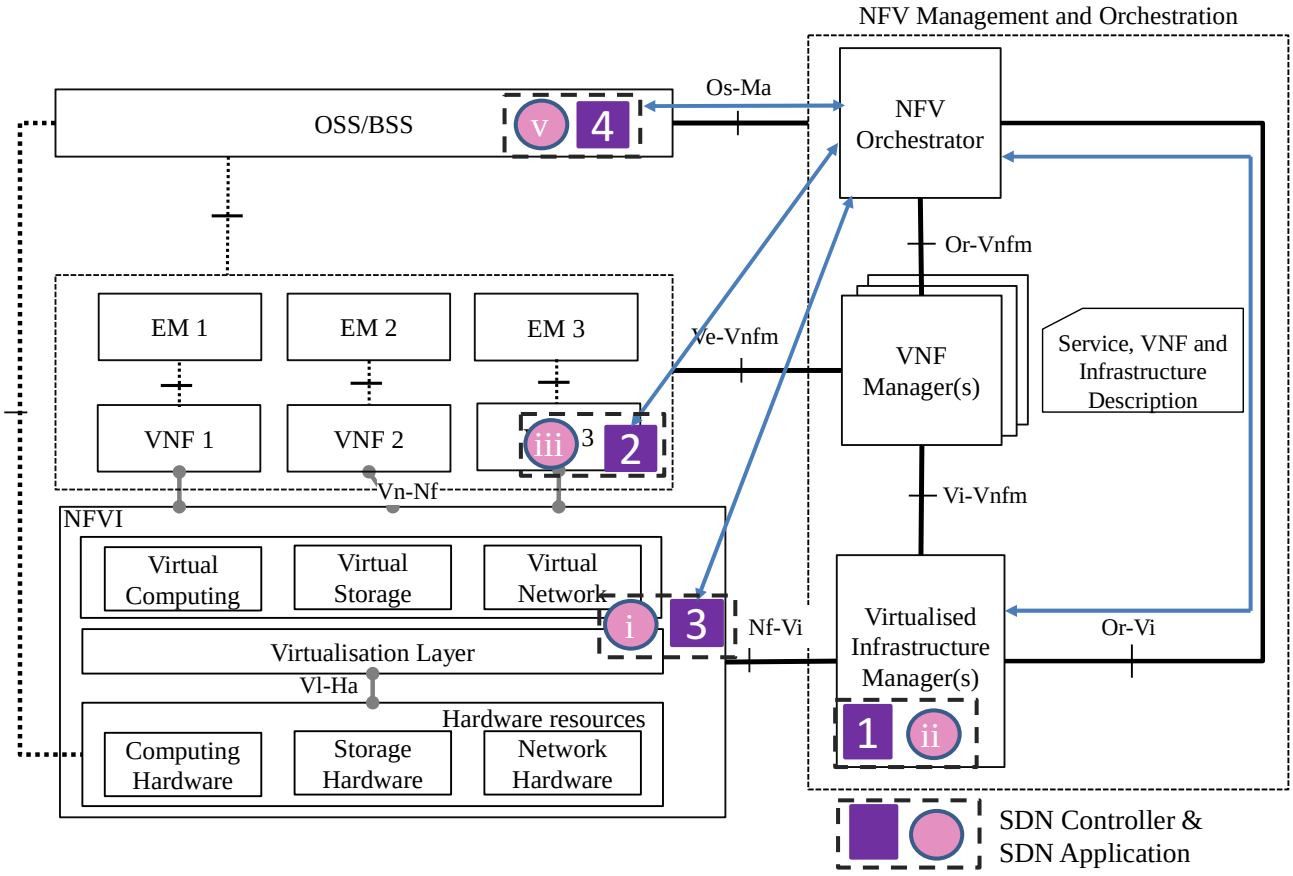



**Figure 7: SDN Controller/Application Orchestration Interface Options**

The orchestration might need to access either SDN controllers or SDN applications. From an orchestration perspective, an SDN controller and SDN applications have a lot in common so the orchestrator might interface both using the same interface. Figure 7 puts the SDN controller and SDN application together, reusing the same orchestration interface. This is also an actual implementation option, as many SDN controllers also include SDN applications.

Figure 7 shows the different implementation options for the SDN controller Orchestration Interface. Below are the different implementation options.

**Table 2: A List of SDN Controller/Application Orchestration Interface Options in NFV**

| I/F Ref | SDN Controller/SDN Application Location | Orchestration Location | Comments |
|---|---|---|---|
| 1 | 1. VIM | NFVO | Uses Or-Vi interface |
| 2 | 2. VNF | NFVO | |
| 3 | 3. NFVI | NFVO | |
| 4 | 4. OSS/BSS | NFVO | Uses Os-Ma interface |

The information conveyed between the NFVO and the SDN controller is similar in all cases. A single SDN controller Orchestration Interface is used in all cases. It is recommended to use existing interfaces where possible. In the case where the SDN controller is implemented as a VNF, the SDN controller Orchestration Interface might possibly be achieved indirectly through the VNF Manager. In the case where the SDN controller is implemented in the NFVI, the SDN controller Orchestration Interface might possibly be achieved indirectly through the VIM.

## 4.4.4 SDN application control interface options

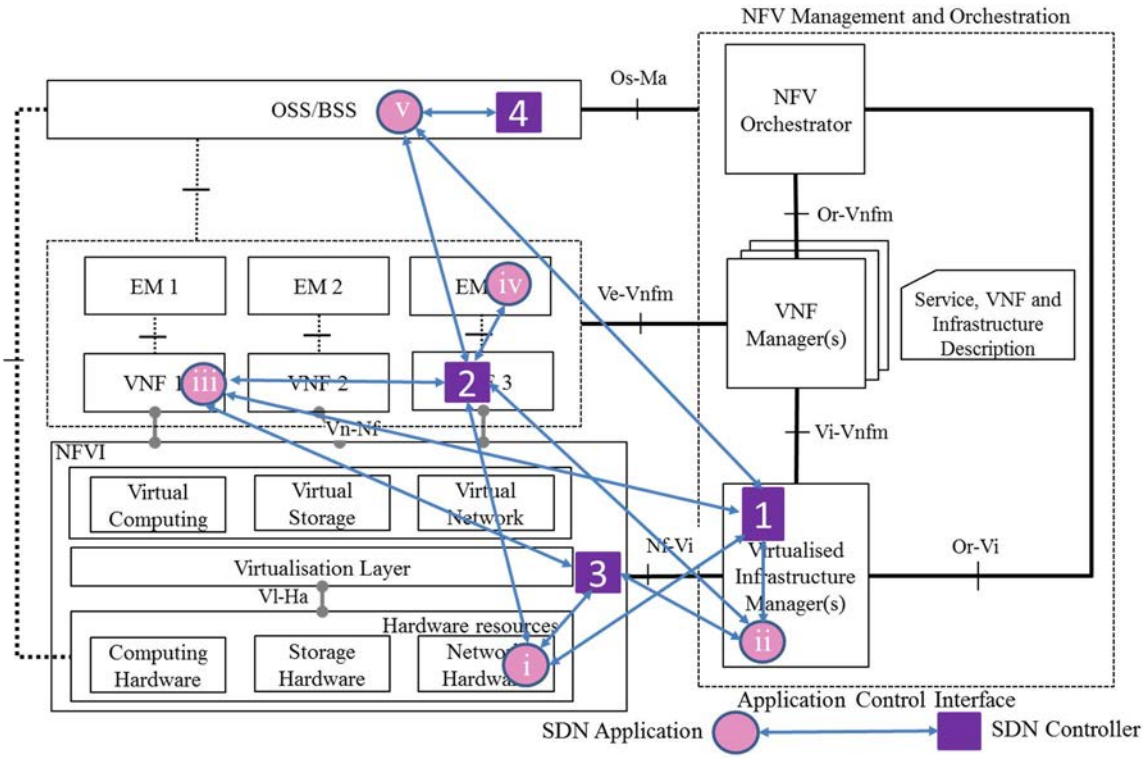



**Figure 8: SDN Application Control Interface Options in NFV**

Figure 8 shows the different implementation options for the SDN Application Control Interface. There are several cases where the SDN controller and the SDN applications reside in similar entities. In these cases the interface might be an internal interface. For example, an SDN controller in a VNF might include one or more SDN applications. In this case the interface between the SDN applications and the SDN controller in the same VNF is an internal interface.

**Table 3: A List of SDN Application Control Interface Options in NFV**

| I/F Ref | SDN Controller Location | SDN Application Location | Comments |
|---|---|---|---|
| 1i | 1. VIM | i. NFVI (Network Hardware) | Not common |
| 1ii | 1. VIM | ii. VIM | VIM includes an SDN application and the SDN controller functionality is merged with the VIM. An internal VIM Interface. |
| 1iii | 1. VIM | iii. VNF | Assumes the VNF is logically part of the NFVI |
| 1v | 1. VIM | v. OSS/BSS | |
| 2i | 2. VNF | i. NFVI (Network Hardware) | Not common. Assumes the VNF is logically part of the NFVI |
| 2ii | 2. VNF | ii. VIM | Not common |
| 2iii | 2. VNF | iii. VNF | Case 1: SDN controller includes an SDN application In this case the interface is internal. Out of scope of the present document.<br>Case 2: SDN controller and SDN application VNFs. VNF will expose REST API, as Application Control Interface<br>Both VNFs either logically belong to the NFVI or to independent infrastructure tenants. |
| 2iv | 2. VNF | iv. EM | Not common |
| 2v | 2. VNF | v. OSS/BSS | Not common |
| 3i | 3. NFVI | i. NFVI (Network Hardware) | Internal NFVI interface. |
| 3ii | 3. NFVI | ii. VIM | |
| 3iii | 3. NFVI | iii. VNF | |
| 4v | 4. OSS/BSS | v. OSS/BSS | OSS/BSS includes both an SDN application and an SDN controller. An internal OSS/BSS interface. |

There are several suggested SDN Application Control Interfaces such as the Intent NorthBound Interface. NFV might optionally use some of these suggested SDN Application Control Interfaces. The list in table 3 shows that the same SDN Application Control Interface can be used in various implementation options. Some of the interfaces listed might be indirect interfaces. This means that the SDN Application Control Interface between the SDN controller and an SDN application might flow through other NFV entities.

## 4.5 SDN controller to controller interface in NFV

### 4.5.1 SDN controller to controller interface options in NFV

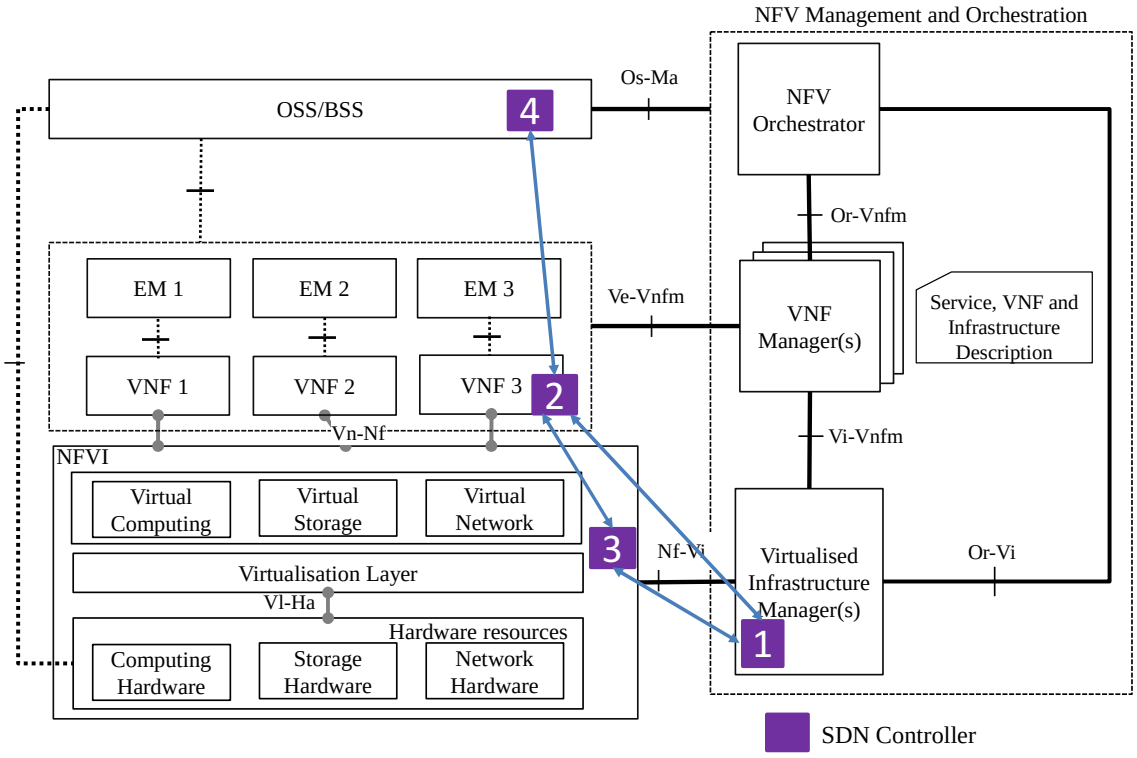



**Figure 9: SDN Controller to Controller Interface Options in NFV**

SDN controllers might have several possible location within the NFV reference architecture, as shown in figure 3. The SDN controllers interface between each other using the SDN Controller to Controller Interface. Figure 9 shows the different implementation options for the SDN Controller to Controller Interface. There are several cases where the SDN controllers reside in different NFV entities. In these cases SDN Controller to Controller Interface needs to be an external interface. For example, an SDN controller in a VNF might interface with an SDN controller in the VIM.

**Table 4: A List of SDN Controller to Controller Interface Options in NFV**

| I/F Ref | SDN Controller Location | SDN controller Location | Comments |
|---|---|---|---|
| 12 | 1. VIM | 2. VNF | |
| 13 | 1. VIM | 3. NFVI | |
| 23 | 2. VNF | 3. NFVI | |
| 24 | 2. VNF | 4. OSS/BSS | Not common |

There are several suggested SDN Controller to Controller Interface. NFV might optionally use some of these suggested SDN Controller to Controller Interface. The list in table 4 suggests that the same SDN Controller to Controller Interface be used in various implementation options. Some of the interfaces listed might be indirect interfaces. This means that the SDN Controller to Controller Interface flow through other NFV entities.

### 4.5.2 SDN controller federation options in NFV

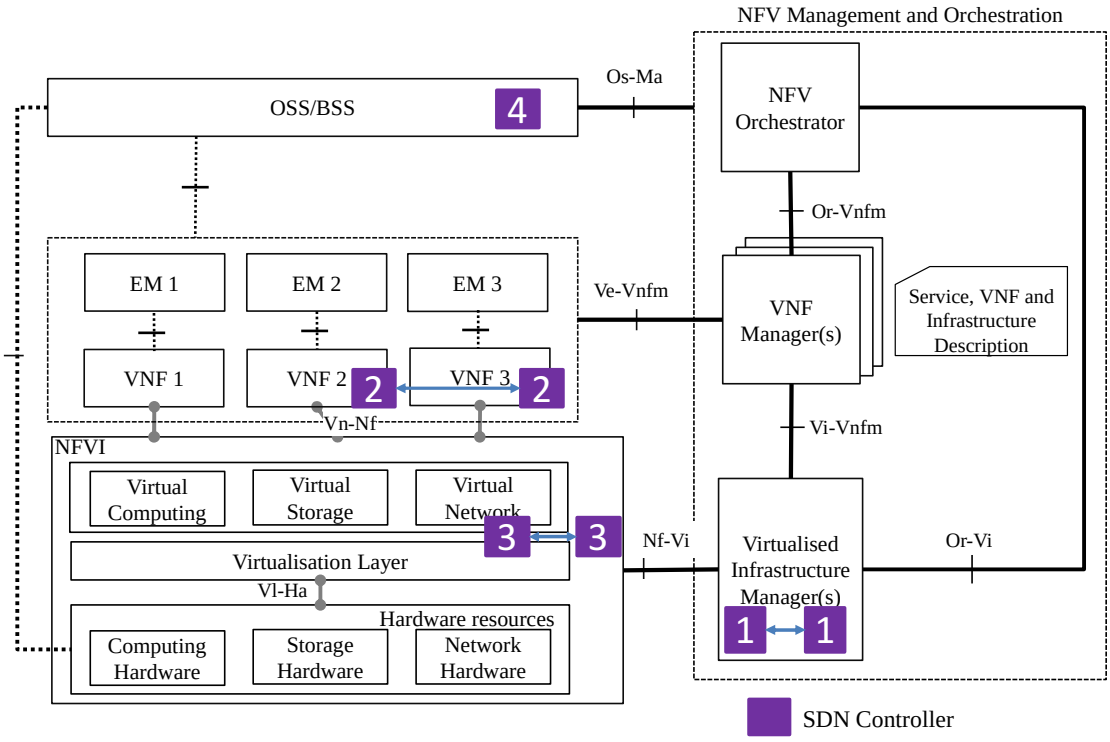



**Figure 10: SDN Controller Federation Options in NFV**

SDN controllers' federation might take place in several locations within the NFV reference architecture. The SDN controllers' federation is done using the SDN Controller to Controller Interface. Figure 10 shows the different implementation options for SDN federation in the NFV reference architecture.

**Table 5: List of SDN Controller Federation Options in NFV**

| I/F Ref | SDN Controller Location | SDN Controller Location | Comments |
|---|---|---|---|
| 11 | 1. VIM | 1. VIM | |
| 22 | 2. VNF | 2. VNF | |
| 33 | 3. NFVI | 3. NFVI | Not common |

SDN controllers' federation might take place within a single domain. For example several SDN controllers residing as VNFs in a single NFVI-PoP might form a cluster. This cluster might be used for high availability and load balancing purposes.

SDN controllers' federation might also take place across different administrator domains or NFVI-PoPs. For example several SDN controllers implemented across different VIMs might be federated. This federation allows for sharing and passing resources between VIMs.

# 5 Design patterns of SDN in the NFV architectural framework

## 5.1 Introduction

This clause is describing common usage patterns of SDN design and operations. The goal is to capture all practically relevant points in applying SDN application, SDN controller, and SDN resource, from which recommendations on these entities might be derived.

## 5.2 SDN technology integration options with NFV

### 5.2.0 Introduction

This clause describes options for how SDN technology might be integrated into the NFV framework.

NOTE: Data plane is used here to refer to the network associated with the primary traffic a VNF is designed to process. Not considered here are how SDN and NFV are integrated in the data plane or the management and the control planes.

### 5.2.1 Interconnecting VNFCs using SDN

A VNF might be composed of several VNFCs with SDN used to both establish links between VNFCs and support other functions such as managing traffic between the VNFCs.

### 5.2.2 Interconnecting VNFs using SDN

#### 5.2.2.0 Introduction

A network service might be composed of a source, a destination, and a set of intermediate interconnected VNFs that process the traffic from the source to the destination. This chaining of VNFs might occur in a number of scenarios, outlined below.

#### 5.2.2.1 Chaining based on network service designed according to a VNF-FG

A Network Service, as defined in [i.2], figure 6.5, contains one or more VNF-FG definitions that might be used when creating an instance of that Network Service. VNFs, therefore, might be chained together based upon the graph defined in one or more of a Network Service's VNF-FGs. This is a static chaining, applied at instantiation time of the network service. The interconnection between the VNF might be implemented by using L2/L3 segments and is established based on the fixed VNF-FGs that are defined ahead of time in the Network Service Descriptor.

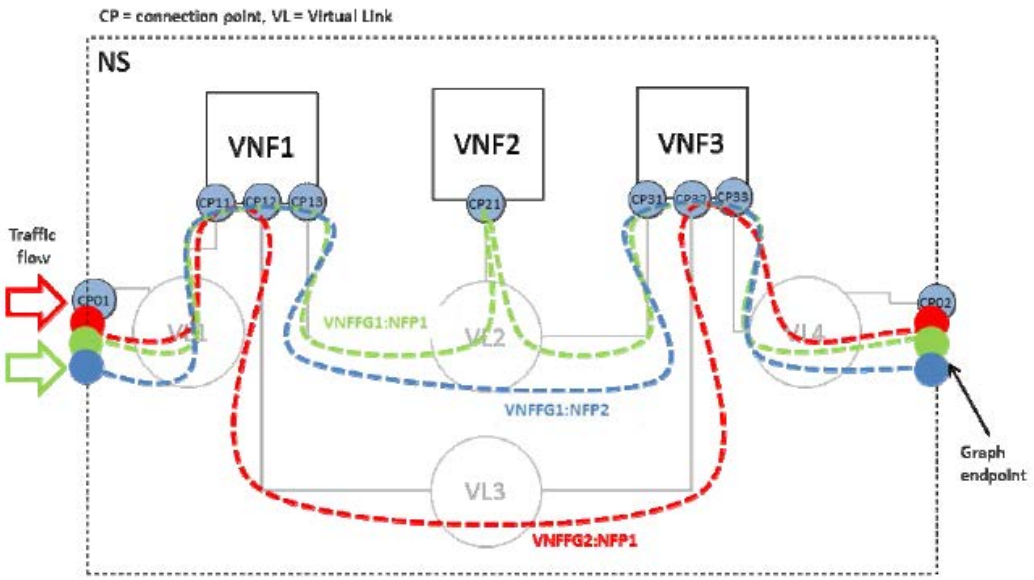



**Figure 11: Network Service as defined in figure 6.5 in MANO GS [i.2]**

#### 5.2.2.2 Chaining based on customer policy/service

The previous example describes a fairly static establishment of a VNF chain, where a pre-defined VNF-FG is used and there is nearly no consideration of additional context. However, a more dynamic and flexible delivery of VNF chains is also envisioned. Which VNFs are connected and in what order will be determined, at the time of service instantiation, based on a variety of factors that will include policies, network, compute and storage conditions, and customer based policies.

#### 5.2.2.3 Chaining based on VNF processing

VNF chaining with regard to a specific or aggregate traffic flow might be modified on the fly depending on the processing provided by a VNF. For example a security related VNF might want to change a traffic path. This might be done by providing that VNF with direct or non-direct access (e.g. via the NFVO or VIM) to the Application Control Interface of an SDN controller so that the change might be dynamically implemented.

#### 5.2.2.4 Load balancing across VNF's

A key capability of the NFV architecture is VNF elasticity. Elasticity enables scaling in order to grow or shrink traffic handling capacity either by adding or removing VNFC or VNF instances. The ETSI NFV architecture framework supports both of these options. For example, SDN might be used for automating the network so that when additional capacity is required, ingress traffic might be segregated per policy and forwarded to new additional elements for processing. Load balancing is considered as a L1-7capability.

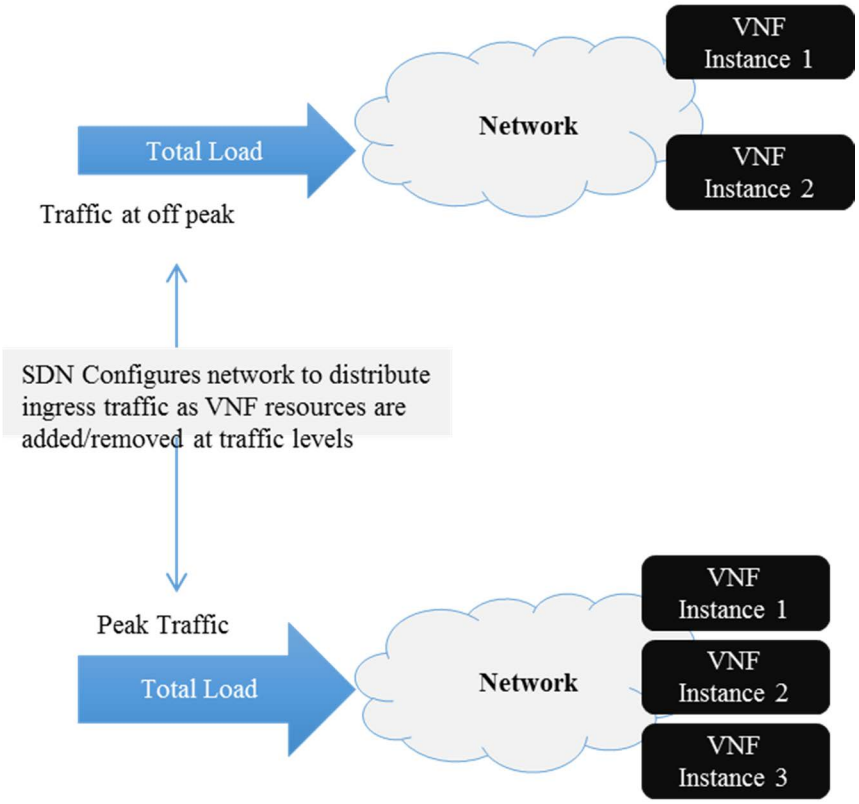



**Figure 12: Load Balancing Across VNFs**

For providing this capability SDN controller(s,) might receive input from the ETSI MANO functional entities so that they might keep track of VNF instances, availability etc. and traffic might be forwarded to the appropriate instance as per policy. This traffic segregation as per policy, has to be implemented at a line rate.

## 5.3 SDN across multiple VIM

### 5.3.0 Introduction

The clause 5.3 follows the general principles described in ETSI GS NFV-MAN 001 [i.2] (particularly clause 5.6.2), ETSI GS NFV-INF 005 [i.1] for network controllers in the NFV architectural framework [i.3] and the SDN architecture outlined in [i.6].

### 5.3.1 SDN controller interfaces

As described above in clause 4.2 (and taken from Recommendation ITU-T Y.3300 [i.8]), an SDN controller has got a:

- Northbound interface called the Application-Control Interface (ACI) - provides a northbound application programmatic control of abstracted network resources.

- Southbound interface, called the Resource-Control Interface (RCI) - controls network resources.

The possibility of defining different levels of abstraction for the network resources enables the definition of hierarchically recursive application/controller layers, with higher levels providing greater abstraction and broader scope. Here the Application Control Interface from one SDN controller presents abstract network resources to the resource-control interface of an SDN controller above it.

A number of the multi-VIM scenarios described below utilize hierarchical controllers and network abstractions. These enable the optimized utilization of network resources and the secure isolation of network resources across multiple trust domains.

## 5.3.2 Scenarios for SDN across multiple VIMs

To identify the challenges related to SDN deployment across multiple VIMs, two basic scenarios are considered separately:

a) VIM instances located in the same NFVI-PoP; and

b) VIM instances located in different NFVI-PoPs.

In case b, at least one and possibly multiple WAN domains will exist between the NFVI-PoPs. The WAN connection supports SDN connectivity between VNFs, as well as PNFs, present in different NFVI-PoP.

## 5.3.3 Challenges for SDN across multiple VIMs

The two scenarios listed in clause 5.3.2 pose different challenges, including (but not limited to):

- The crossing of administrative or organizational boundaries usually imposes requirements at various levels, e.g. connectivity, security, SLA fulfilment, information hiding, which constrains the choice of possible solutions.
- The characteristics of network protocols used in the scope of a NFVI-PoP and in the WAN are significantly different.
- The provisioning of network services across VIMs with requirements such as end-to-end low latency.
- The dynamic setup of paths between newly instantiated VNF, and existing PNF or VNF over WAN.
- Ensuring connection as well as SLA guarantee for VNF scaling over multiple NFVI-PoPs.
- The extension of virtual links outside the NFVI-PoP into the WAN.
- Transport network reconfiguration, e.g. during natural disasters, network congestion, etc.

## 5.3.4 Analysis of SDN across multiple VIMs

### 5.3.4.1 SDN across multiple VIMs located in a single NFVI-PoP

a) In this case, a single trust domain is supposed to exist between endpoints, therefore the administration of NFVI and network resources is supposed to be handled by a single entity, which considerably simplifies the problem. The interconnection between VIM instances is supposed to be provided by protocols typically used in data centers (e.g. VXLAN, NVGRE, LISP).

### 5.3.4.2 SDN across multiple VIM in different NFVI-PoPs

#### 5.3.4.2.0 Introduction

End-to-end connectivity between VMs hosted on different NFVI-PoPs is built on network resources in three different segments:

- the internal networks of the edge NFVI-PoPs;
- the gateway routers at the edges NFVI-PoPs;
- the WAN transport network in between.

The configuration of the intra-NFVI-PoP network resources is locally enforced, under the supervision of the respective VIM. One or more WAN domains are expected to exist between NFVI-PoPs.

When there are multiple WAN domains, an understanding of the capabilities and connectivity endpoints provided by these WANs is necessary, so that an appropriate WAN will be used at path setup, based upon the VNF locations and traffic service classes needed.

Different trust or organizational boundaries are expected to exist between the NFVI-PoP and the WAN domains. WAN operators and NFVI-PoP operators might be different, with the NFVI-PoP operator using the WAN services provided by the WAN operator. In the following, two basic sub-scenarios are briefly described.

#### 5.3.4.2.1 VNFs across multiple NFVI-PoP locations with a static bit pipe between them

In this case, an overlay network, crossing transparently one or more network domains and trust/organizational boundaries, built upon a pre-allocated static WAN connectivity service (e.g. E-Line, E-LAN, IP/MPLS VPN) and typically based on a long-term contract, is established between multiple termination points located at geographically remote NFVI-PoPs under the same trust domain. This scenario is rather simple, as it does not require coordination between NFVI-PoP and WAN resources, but also limited in the sense that the WAN resources do not follow agility and dynamism of the NFVI-PoP network environment.

Figure 13 below illustrates the simplest case (this basic scenario could be extended to any number of NFVI-PoPs), where a point-to-point connection between two NFVI-PoPs is transparently carried across the WAN over a transport tunnel by means of technologies such as MPLS or IEEE 802.1ad. Each endpoint is aware of the specific characteristics (e.g. bandwidth capacity) of the inter-NFVI-PoP connection, but the WAN network nodes are only aware of the transport pipe. Thus, the lifecycle of the inter-NFVI PoP inner connection (e.g. establishment, modification, release) is controlled by the intervening VIMs, but not the WAN domain(s) in between.

Changes to the outer tunnel, if needed at all, are expected to occur on a relatively large timescale and are not expected to require automated control mechanisms.

From the perspective of the NFVO, this solution, utilizing the static tunnel, appears identical to the first scenario with a single NFVI-PoP containing multiple VIM of the same trust domain. Establishment of the network connections among the NFVI is basically the same, regardless of whether the VNF are in the same or remote NFVI-PoP.

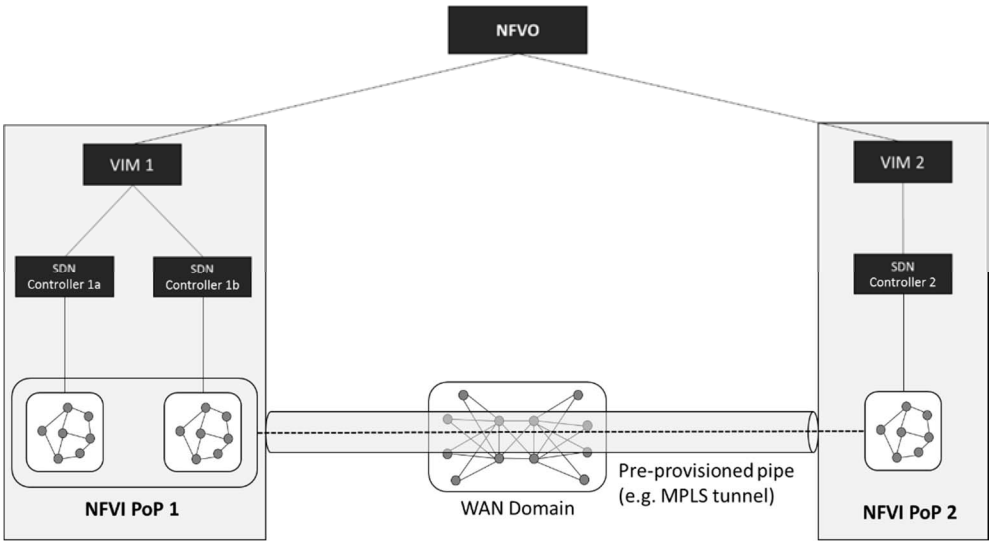



**Figure 13: NFVI-PoP Interconnection with Pre-Provisioned Static Pipe**

#### 5.3.4.2.2 VNFs across multiple NFVI-PoP locations, SDN-based NaaS between them

This scenario leverages SDN-based control in the WAN to build a richer networking environment. The WAN domain between NFVI-PoPs provides an "on-demand" connectivity service, NaaS (Network as a Service). This scenario is more complex than the previous one, in the sense that it requires coordinated control of NFVI and WAN resources. WAN connectivity services are explicitly requested by the NFVO through the WIM (WAN Infrastructure Manager), following the model described in [i.1]. Following the interface terminology proposed in [i.1], this corresponds to the Nd-Nd interface, in the case where the WAN domains interconnecting the NFVI-PoPs are SDN-enabled.

The connectivity service provided by the WAN domain could be either L2-based (e.g. following service models defined by the MEF – Ethernet Virtual Private Line, Ethernet Virtual Private LAN, Ethernet Virtual Tree [i.7]), or L3-based (e.g. IP/MPLS VPN).

From a functional point of view, there is a client-provider relationship between each NFVI-PoP and the respective WAN provider, even if they are under the administration of the same business entity.

The level of trust between the NFVI-PoP and the WAN domain(s) might have implications in terms of resource management policies, security and information sharing (usually, network operators are unwilling to reveal the internal details of their network infrastructure). Figures 14 to 16 illustrate three possible approaches for the interaction between the NFV orchestration and the WAN domains in this scenario. These three approaches might be mapped directly into 3 out of 4 scenarios described in [i.6] (clause 5, "Control functions and interactions") for the SDN control plane.

#### 5.3.4.2.3 VNFs across multiple NFVI-PoP locations - NFVI-PoP and WAN in a common trust domain (Option A)

In this scenario, also called Option A in the present clause, the WIM implements the application-control interface and uses directly the services exposed by the northbound interface of the WAN SDN controller. The WAN SDN controller, administered by the WAN operator, is in charge of managing the underlying physical resources. This is a good candidate for cases where NFVI and WAN resources are administered by the same entity.

NOTE: This is not a case of the WIM executing network controller actions over and above the WAN SDN controller. Here in Option A the WAN SDN controller and the NVFI network resources it controls are in the same trust domain as the NVFI-PoPs. In fact, how the WIM manages the WAN resources mirrors the VIM control of the NFVI-PoP resources. A 2-tier hierarchy of controllers, used when the WAN and NFVI-PoPs are in different trust domains, is described as Option B.

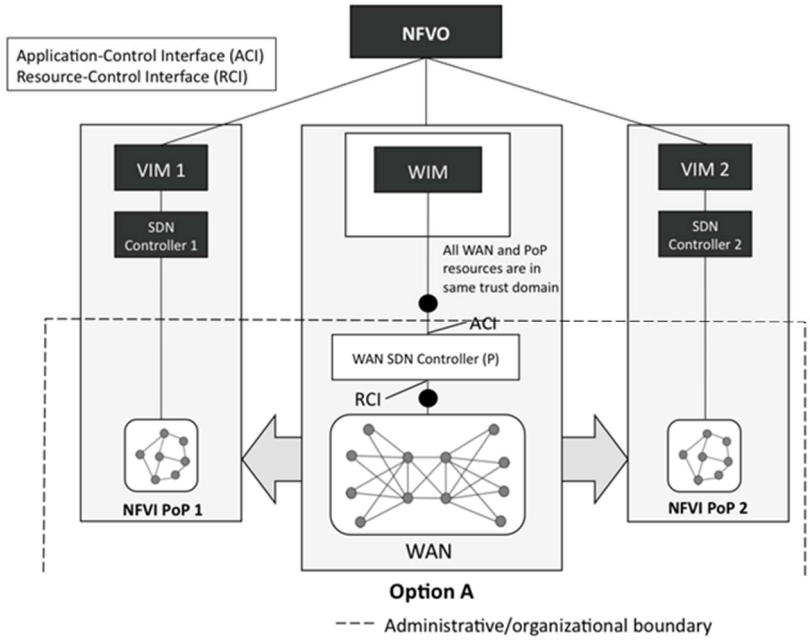



**Figure 14: A Single SDN Controller for the WAN Resources (Option A)**

#### 5.3.4.2.4 VNFs across multiple NFVI-PoP locations - NFVI-PoP and WAN in different trust domains - client access to virtual WAN resources (Option B)

Option B might be seen as an extension of option A by defining a 2-level hierarchy of SDN controllers - C (Client, administratively part of the endpoints' NFVI-PoP trust domain) and P (Provider, administratively part of the WAN).

As described in [i.6], the WAN SDN controller accommodates an agent for each client virtual network controller (C). This agent presents a distinct application-control interface to each client virtual network controller. Controller P, as described above, provides a northbound application programmatic control of abstracted network resources. To support multiple client networks, the WAN P controller provides a unique set of abstracted resources to each client. In this case, a client virtual network controller (C) manages only the abstract network resources the WAN P controller exposes to it. This capability to define different levels of abstraction of the network resources for different client virtual network controllers might be a particularly useful feature to establish different administration models of the WAN in a flexible way.

In the figure 15 below, the MANO components for multiple clients are presented. These include the clients' NFVO, WIM and SDN controllers that interacts with the WAN SDN controller. Each client controller is presented with a unique ACI / network abstraction.

This scenario, Option B, includes a 2-level hierarchy of SDN controllers. A detailed analysis of this SDN controller hierarchy is included in clause 5.4.

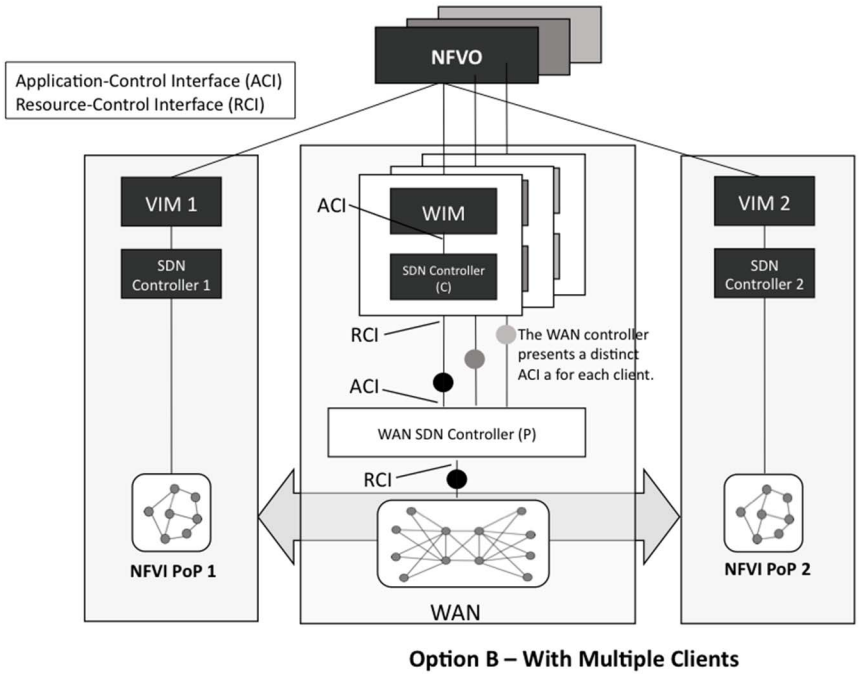



**Figure 15: WAN Controller Presenting Multiple ACI for unique abstractions per client (Option B)**

#### 5.3.4.2.5 VNFs across multiple NFVI-PoP locations - NFVI-PoP and WAN in different trust domains - client access to physical host resources (Option C)

In this scenario, Option C, the WIM requests network resources through an SDN controller, which interacts directly with the network nodes. This means the client SDN controller is being provided direct access to control a subset of the WAN network resources.

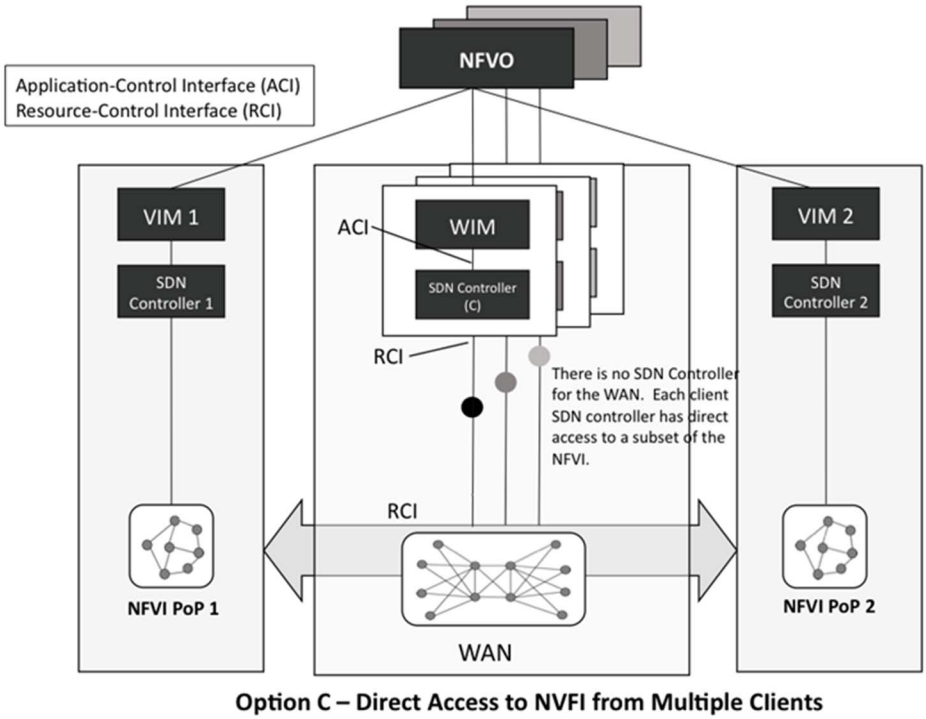



**Figure 16: No WAN Controller - each client SDN controller has direct access to WAN resources (Option C)**

The options indicated above might be mapped to the scenarios described in clause 5 of [i.6], "Control functions and interactions", as follows:

- Option A maps into scenario "Single player SDN provider" ([i.6], clause 5.1).
- Option B maps into scenario "SDN provider with Virtualised network, non-recursive" ([i.6], clause 5.3).
- Option C maps into scenario "SDN provider with SDN clients, with underlying network exposed" ([i.6], clause 5.2). This option, classified as "not recommended for deployment in practice", is included here for completeness.
- The fourth scenario described in [i.6], clause 5.4 "SDN provider with recursive Virtualised network" is considered to be beyond the scope of this clause.

Although option B, as described above, includes a 2-level hierarchy of SDN controllers, a detailed analysis of SDN controller hierarchy issues is not performed here (but rather in the present document clause 5.4 - "SDN controller hierarchy").

#### 5.3.4.2.6 SDN based NFV IaaS across multiple administration domains

An additional pair of scenarios to consider is when a network service expands its domain by creating a virtual PoP within the NVFI-PoP of another trust domain. Here, the expanding service is a client of IaaS / NaaS offerings from the host (or server) service.

These scenarios might occur, for example, when a service provider wants to expand geographically and either needs to do so quickly, temporarily, or simply manage its size dynamically, based on demand.

Two versions are presented here:

- The first is where the NaaS/PaaS service provides resources based on an abstraction provided by an SDN controller of the NaaS/PaaS provider (figure 17).

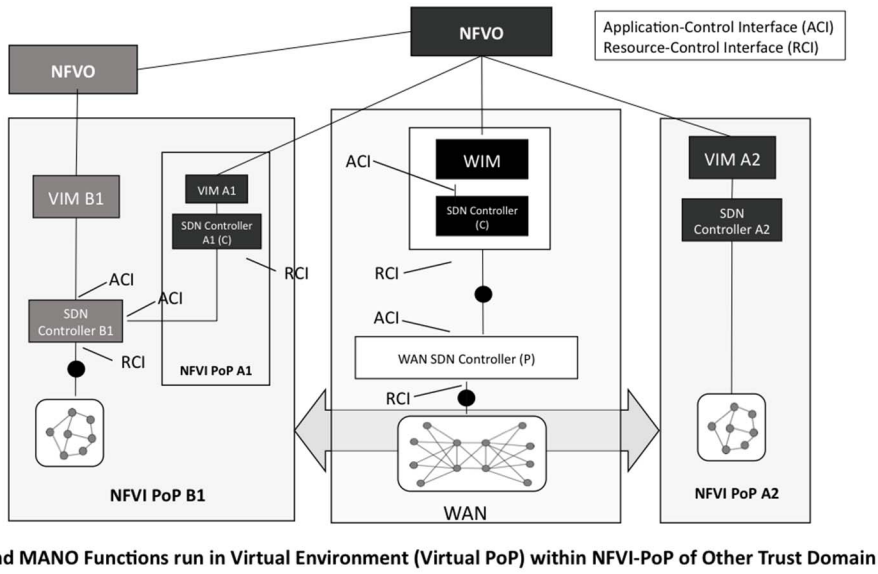



**Figure 17: Virtual NFVI-PoP of Provider A Running as Guest in NFVI-PoP of Provider B**

Here, network control is managed in a way similar to the Option B, described earlier. The SDN controller of the host NFVI-PoP presents a client specific network abstraction (via its Application Control Interface) to the controller of the guest virtual NFVI. In other words, the network of the virtual PoP is the abstraction the host service SDN controller presents to it.

Besides presenting the client specific Application Control Interface to the guest SDN controller, the host SDN controller also presents one to its own VIM.

- The second scenario is similar to what is in the Option C described above, where the NaaS provides the tenant provider PoP A1 direct access to the underlying networking infrastructure. This is shown in figure 18.

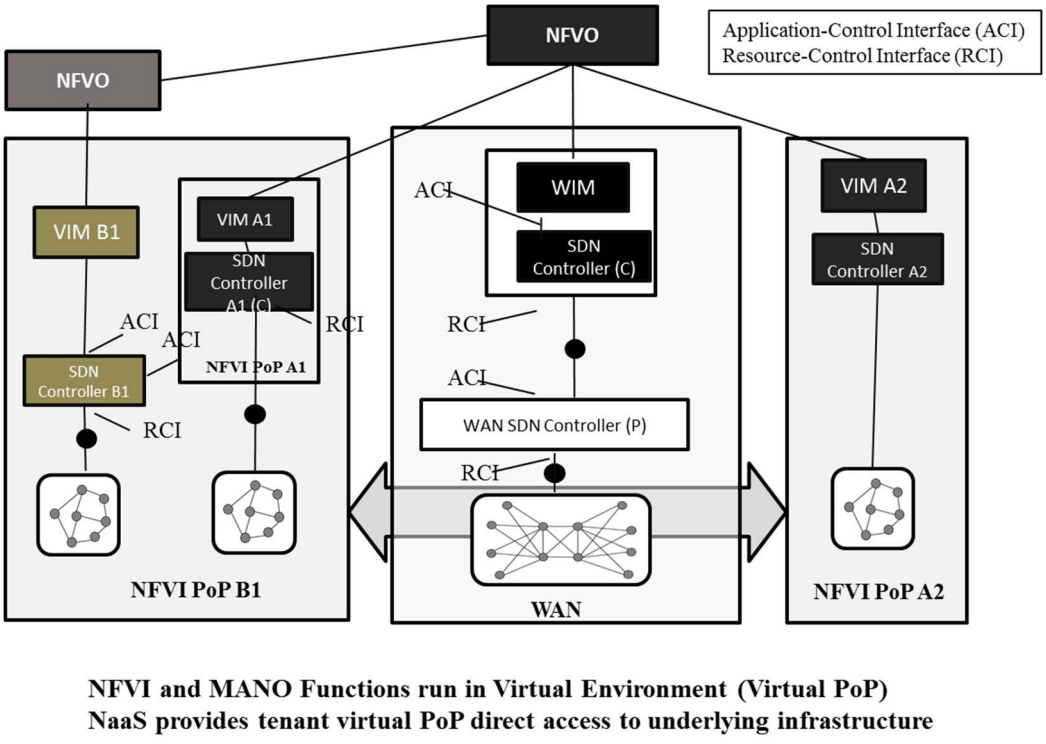



**Figure 18: Virtual NFVI-PoP of Provider A Running as Guest in NFVI-PoP of Provider B**

#### 5.3.4.2.7 Multiple VIM for a mobile network

Figure 19 provides a detailed inter-VIM SDN scenario for a mobile network which extends the option B illustrated in figure 15 of clause 5.3.4.2.4. The data plane is depicted more explicitly and the role of function blocks involved in inter-VIM connection management is described.

This scenario depicts a mobile network service provider that inter-connects its NFVI-PoPs utilizing other administration domains supportive for SDN controller and SDN data plane.

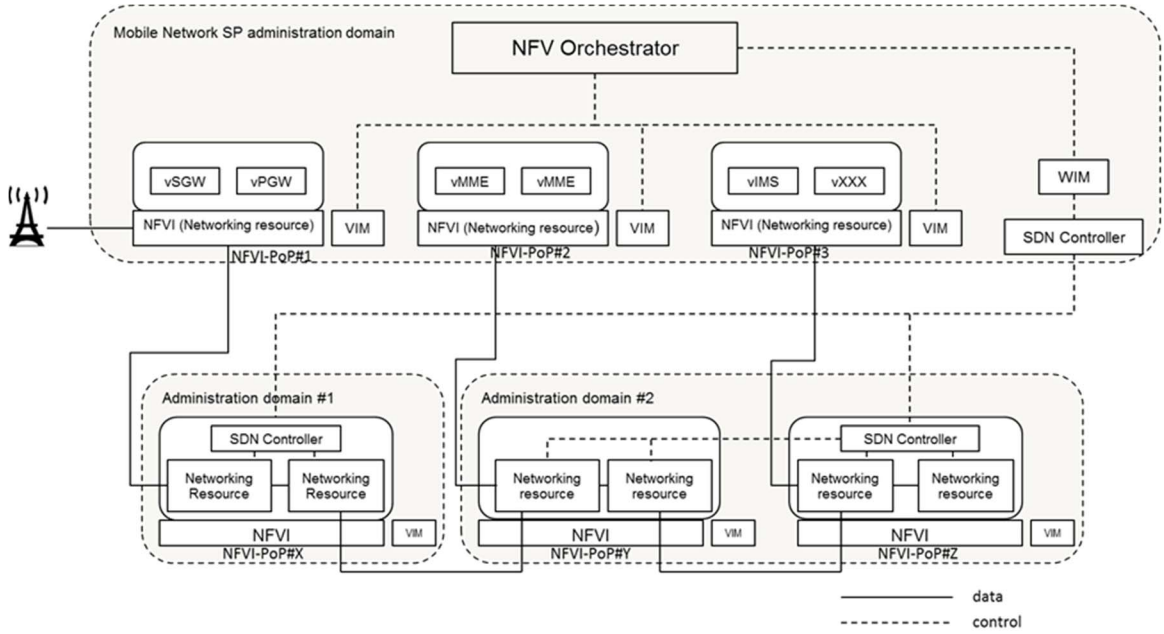



**Figure 19: The Mobile Network Scenario for SDN based NaaS Provided by Other Administration Domain**

As shown in the figure 19, the NFVI-PoP #1, #2 and #3 are inter-connected through administration domain #1 and #2 which deploys SDN architecture based network. The administration domain #1 and #2 host multiple SDN data plane elements and provide the Virtualised network resources through a SDN controller to the WIM managed by the mobile network service provider. The mobile network service provider might orchestrate the NFVI-PoP networking resources offered by the VIMs and cross NFVI-PoP networking resources by the WIM through the NFV Orchestrator.

As per ETSI GS NFV-MAN 001 [i.2], the inter NFVI-PoP connections are managed by NFV Orchestrator and WIM, i.e. inter-connection establish/release/change decisions are at the NFV Orchestrator level and operations conducted at the WIM level.

## 5.4 SDN controller hierarchy

### 5.4.1 Introduction

- A few key technical and business drivers typically drive the hierarchy of SDN controllers. These include: Improving performance, scalability and reliability by dividing the network.
- Supporting multiple administrative domains / tenant networks.
- Defining multiple network layers Virtualisation.

Mapped to the ETSI NFV architectural framework, this translates into a set of options for applying a hierarchy of SDN controllers. This clause describes some SDN controller hierarchies and possible scenarios for their application.

As described in figure 3 SDN controllers might be used in different functional blocks of the ETSI NFV architectural framework, such as in the NFVI or as a VNF. As such SDN Controller hierarchy might be used in WAN environment, inside Data Center, between WAN and Data Center PoP.

This clause describes SDN Controller hierarchies and scenarios for implementation.

## 5.4.2 Technical and business scenario

### 5.4.2.1 SDN controller hierarchy scenarios overview

The following SDN controller hierarchy scenarios are being described in this clause:

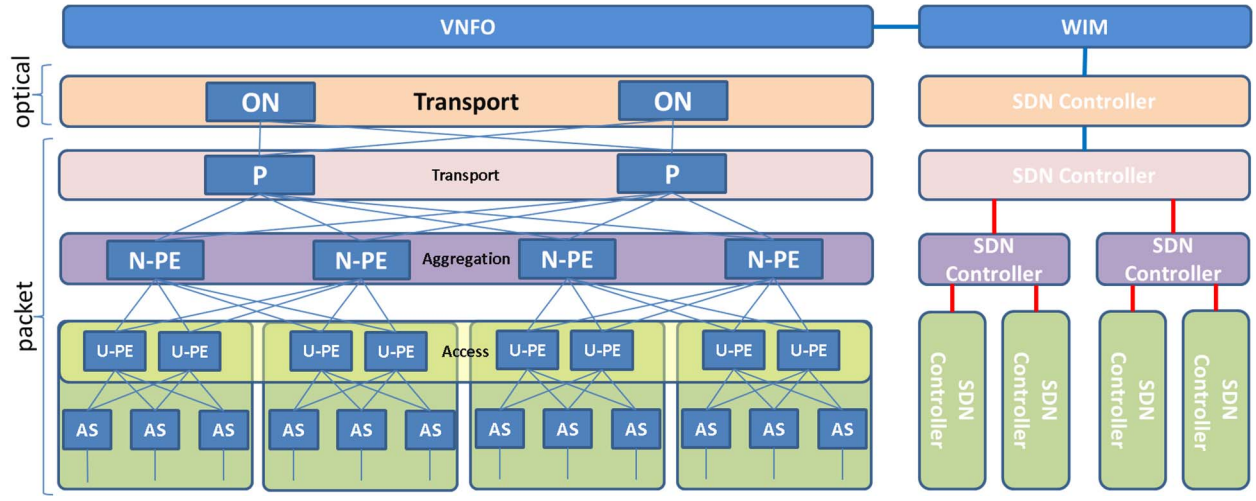
- Distributed performance, scalability and reliability
- Administrative domains interaction
- NaaS management
- Multilayer transport fault management

### 5.4.2.2 SDN controller hierarchy for distributed performance, scalability and reliability for multilayer and single-layer transport network

WAN and Data Centre networks typically have hierarchical structures with potentially a very large number of NEs. An SDN controller not only manages traffic handling rules for all of these NEs but also collects status notifications (OAM information). Therefore, an SDN controller hierarchy approach could be used to avoid performance, scalability and reliability issues. The top network controller will manage network resource abstractions provided by lower level controllers. Lower layer network controllers will manage sets of NEs associated with them. If one of these network controllers fails, another controller will serve as a backup.

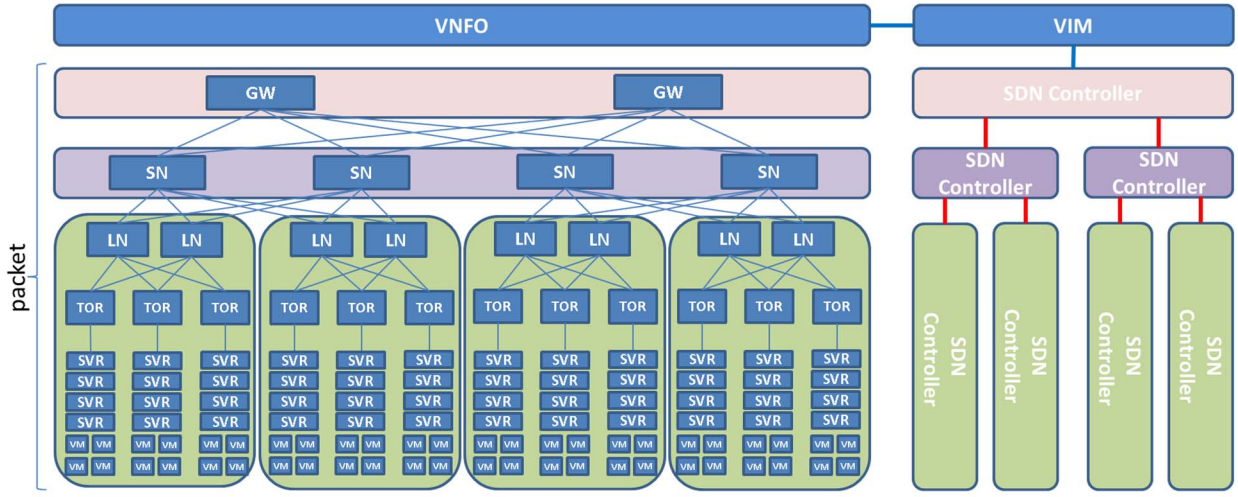
In the figure 20 below, there is an SDN hierarchy for a multilayer network where SDN controller of packet transport network might use the abstraction provided by SDN controller of the optical transport network.

Figures 20 and 21 present a Hierarchical SDN architecture (HSDN) for WAN and Data Center environments, respectively. Each top layer serves as a transport connecting other networks of systems. This approach helps to shrink FIB/LFIB transport tables at each layer. The packet transport like MPLS network might be divided into layers and one layer will be carried by another layers. FIB/LFIB information will be kept respectively on each layer SDN controllers, so it will be logical to use hierarchy approach on SDN controller layer.




NOTE 1: ON-Optical Node, P - Provider, PE-Provide Edge, N-PE - Network Provide Edge, U-PE - User Provider Edge, AS - Access Switch.
NOTE 2: The hierarchy of controllers on the right plane controls the forwarding rules of elements on the left plane.

**Figure 20: Typical WAN Packet Transport Structure**




NOTE 1: GW - Gateway, SN - Spine Node, LN - Leaf Node, TOR - Top of the Rack, SVR - Server, VM - Virtual Machine.
NOTE 2: The hierarchy of controllers on the right plane controls the forwarding rules of elements on the left plane.

**Figure 21: Typical Data Center Structure**

### 5.4.2.3 SDN controller hierarchy for distributed, cross-SP or cross-domain services

An SDN controller hierarchy might be used when one SP provides network transport resources to another SP.

This was described earlier in clause 5.3.4.2.2, and was also called Option B. Figure 22, shows 2 WAN domains belonging to SP3 and SP4, respectively. The WAN networks of SP3 and SP4 provide connectivity service for SP1 and SP2. The WIM and SDN controllers of SP1 and SP2, by using R-CI to A-CI interfaces, might make specific connectivity requests of SP3 and SP4. In the figure 22, the SDN controller "client1" is responsible for providing a northbound abstract view of WAN connectivity, while at the same time, choosing how to support that view via requests of the SP3 and SP4 SDN controllers below. SP2 also have SDN controller (client 2) connected to WAN SP4 to orchestrate NFV-PoP connections across WAN domain.

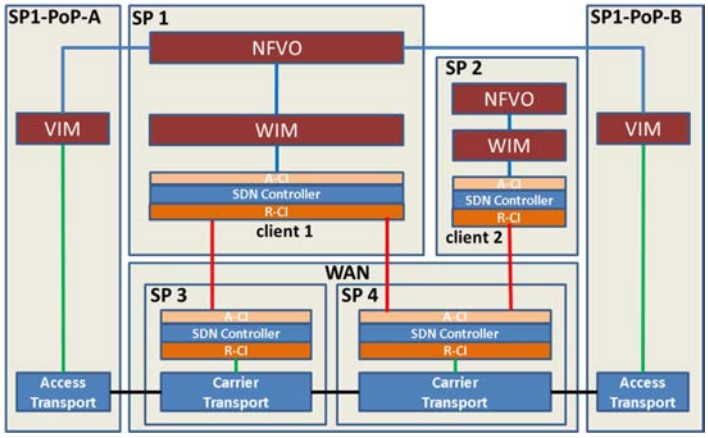



**Figure 22: SDN Hierarchy for Different SP**

In figure 23, a SP provides connectivity services based on its own WAN, (no other SP owns or manages it.) Here the layers of SDN controllers exist to support distinct virtual network abstractions for different organizations within the SP. There is a transport-level network controller to configure the transport connectivity. On top of it, there are separate controllers to coordinate the transport controller with VIM controllers of certain DCs, for example, for certain service types. PoP A / PoP B responsible for business services and PoP C / PoP D are responsible for generic services.

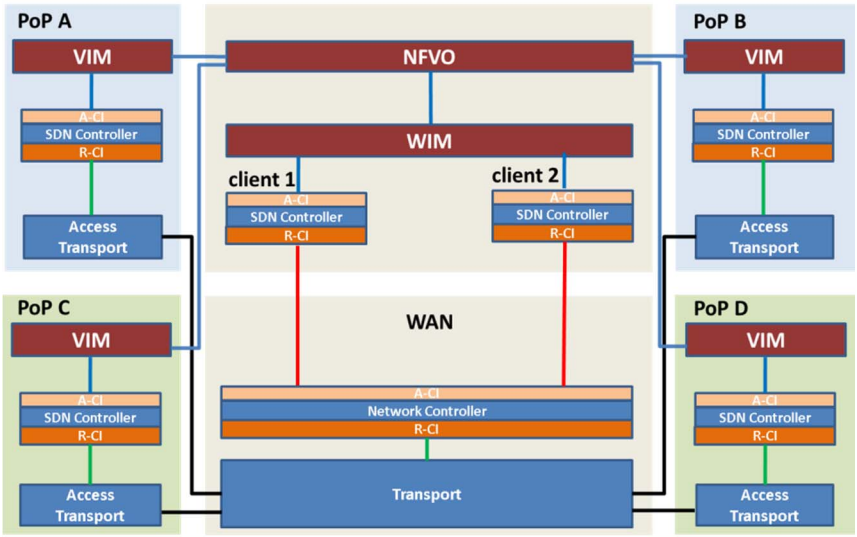



**Figure 23: SDN Hierarchy Within one SP for Service Differentiation Purpose**

### 5.4.2.4 SDN controller hierarchy for NaaS

In figure 24, a SP provides NFVI-B service inside his own NFVI-A. SDN hierarchy allows easy network resources management for NFVI-B within NFVI-A. Access Transport (some vSwiches) might be controlled directly by a tenant within his own NFVI-B however this network infrastructure might be connected to external carrier transport NFVI-B.

SDN controllers hierarchy will be used to maintain such boundary connectivity.

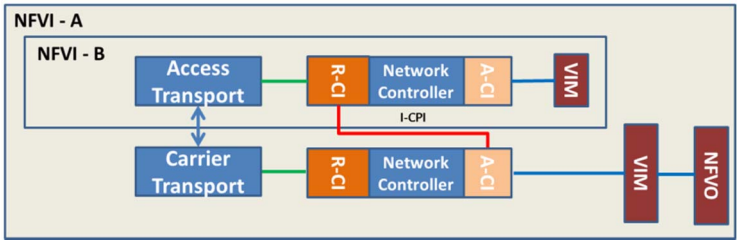



**Figure 24: SDN Hierarchy Within one SP - NaaS**

### 5.4.2.5 SDN controller hierarchy for multi-domain, transport network fast fault recover

Usually a SP network consists of multiple transport layers usually of distinct technologies, where each transport layer has dedicated management. All global service flow optimization, management and deployment actions are driven by the WIM and upper layer orchestration.

To configure automatic fail-over, an SDN controller hierarchy might be used. Faults within each layer might be fixed independently without impact to higher layers or services. Such an approach could remove the WIM and upper layer orchestration from being used to address many WAN faults, with the expectation of more rapid problem resolution.

Furthermore, layers across the WAN might be managed or owned by different SP. There might be two independent MPLS over one or more OTN over one or more optical networks, some subsets of these in different trust domains, managed by different SP. A transport failure might be detected directly and resolved directly in a layer, with communications between layers for optimization, as needed.

To pre-configure back-up options if some network fault occurs, the underlying SDN controller will inform the top SDN controller to sync the back-up configuration with the NE or networks that the top controller manages for back-up as shown in figure 25.

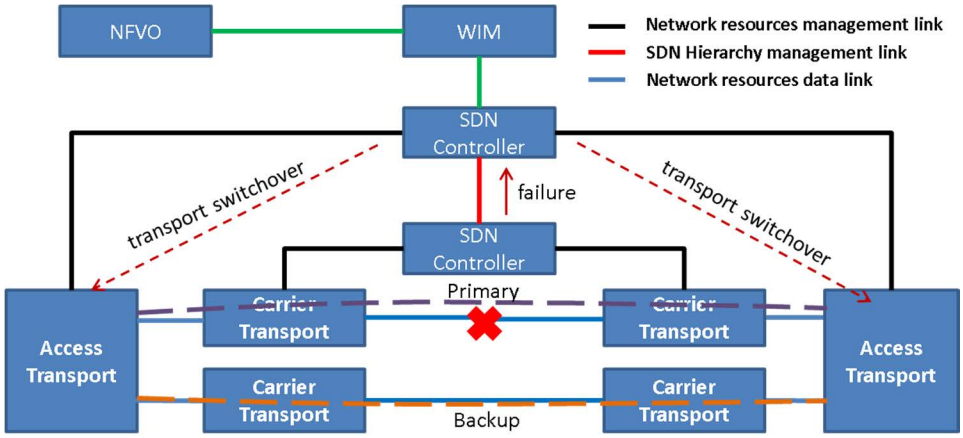



**Figure 25: SDN Controller Hierarchy for Fault Management in Multilayer Transport network**

The same approach might be used within a Data Center as illustrated in figure 26. The tenant virtual site NFVI-B might use a WAN transport which is different from the one used by the host site, NFVI-A.. In case of failure of the primary carrier transport within NFVI-A tenant, NFI-B SDN controller will be informed by NFVI-A SDN controller and NVFI-B access transport will be fast and safely switched to the backup carrier transport within NFVI-A.

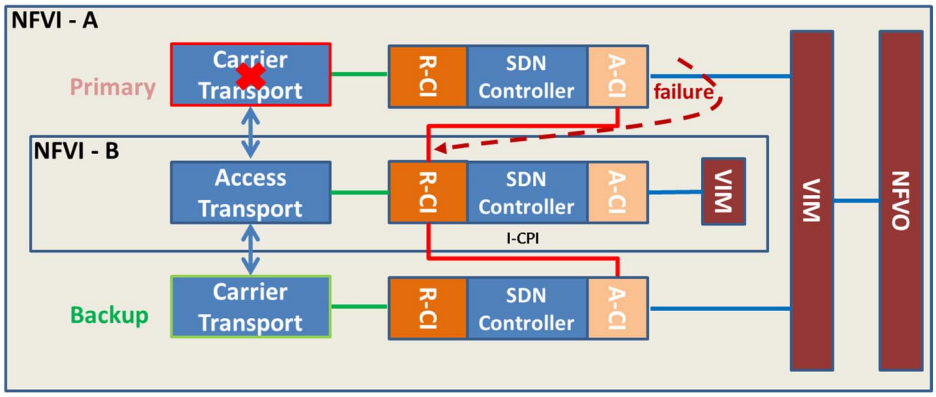



**Figure 26: SDN Controller Hierarchy for Fault Management in Multilayer Transport network**

SDN hierarchy information exchange in the two above scenarios also might be used for traffic management.

The carrier SDN controller might inform the access SDN controller not only about total transport failures but also about degradation issues. So the access SDN controller might decries traffic to a particular transport and perform load balancing between primary and backup transport.

## 5.4.3 Mapping of SDN controller to the ETSI NFV architecture

### 5.4.3.1 Introduction

An SDN controller hierarchy might be implemented within and across different functional blocks of the ETSI NFV functional architecture. The present clause describes a number of these implementation options.

### 5.4.3.2 Hierarchy of SDN controllers in the NFVI

The hierarchy of SDN controllers within the NFVI might fold under 3 cases:

- Hierarchy within a same domain, with underlay and overlay.

The NFVI might be composed of an underlay SDN-based network, managed by another SDN controller in the same layer.

The SDN might also be implemented in the overlay network, and have an SDN controller to manage this overlay.

When both cases are implemented there might be a hierarchy of SDN controllers within the same NFVI PoP and trust domain.

- Hierarchy within a same or multiple domain related to hierarchy of network layers.

In many telecom networks, there are layers of network technologies. Typically for optical networks, as illustrated in Clause C, the SDN controllers are connected to each other via northbound interfaces.

- Hierarchy due to business relationship sharing resources within a same NFVI.

In the final scenario, a Network provider who has an SDN enabled network, exposes some network resources to a guest NFVI-PoP. In this case, as illustrated in figure 17, there might be a hierarchy of SDN controllers, with one SDN controller in NFVI-PoP B1 host network, and one in guest NFVI-PoP A1. The SDN controller B1 is managed by Provider B1, while SDN controller A1 is managed by Provider A1. SDN controller A1 sits on top of the northbound API of SDN controller B1. Both SDN controllers might be Virtualised.

#### 5.4.3.3 Hierarchy of SDN controllers in a VNF

The other case is hierarchy of VNF, where these VNFs are SDN controllers. This might occur in an NFV environment where multiple applications use SDN controllers Virtualised, VNF, and have to exchange information across those VNF controllers. This interfaces would be the inter-VNF interface defined in ETSI GS NFV-SWA [i.5] as SWA-1. These relationships between SDN controllers VNF might be part of a Network Service defined by the NFVO.

#### 5.4.3.4 Hierarchy of SDN controller across functional blocks

The hierarchy of SDN controllers might occur across functional blocks.

For example, it is the case if an SDN controller co-located with the VIM invokes an SDN controller in the NFVI to allocate NFVI network resources. This case happens when a VIM SDN controller both has direct control of NFVI network resources, for example SDN enabled vSwitch, and indirect control of NFVI network resources, for example control of physical SDN switches through an SDN controller provided by a NFVI vendor. It is also the case when a VIM is across multiple NFVI and a WAN, and the VIM SDN controller defines connectivity across these different domains. That VIM SDN controller will interact with the NFVI SDN controller to allocate NFVI network resources or change network paths. This is sometimes combined with hierarchy of SDN controller within a functional block: typically the VIM SDN controller interacts with an NFVI SDN controller, that interacts with another SDN controller within that same NFVI.

#### 5.4.3.5 Hierarchy of SDN controllers below the WIM

While the WIM is not explicitly represented in the ETSI GS NFV 002 [i.3] architectural framework, there is the case of SDN controllers below the WIM in clause 5.3.4 and the hierarchy of SDN controller below the WIM in multiple figures such as figures 15, 17, 18, 19, 20, 21, 22 and 25. This hierarchy occurs when layers and regions of multiple trust domains occur in the Wide Area Network: with a C domain (Client, administratively part of the endpoints' NFVI-PoP trust domain) and P (Provider, administratively part of the WAN).

## 5.5 SDN controller in a Virtualised environment

### 5.5.1 Introduction

Clause 4.3.4 describes the case where an SDN controller is in an NFV architectural framework. In the previous clause 5.4.3.3, is described the scenario where an SDN controller might be Virtualised and be a VNF.

The present clause describes additional use cases, with some guidelines towards the design of an SDN controller to be deployed in a Virtualised environment.

### 5.5.2 Virtualisation of SDN Controller

A Virtualised SDN controller might be:

- A VNFC;
- A VNF; or

- A Network Service (NS) composed of multiple VNF that constitute the different sub-elements of the SDN controller.

This SDN controller to be deployed in an NFV architectural framework:

- would come with a software image and a description of resources required to deploy this software image.

If it is a VNFC, these resource descriptions would be part of the VNF Descriptor, VNFD or the parent VNF.

If it is a VNF, it would be the VNFD.

If it is a NS, it would be part of the NS Descriptor, NSD.

- would follow the architecture guidelines of ETSI NFV.

If it is a VNFC or VNF, these guidelines are primarily as described in ETSI NFV SWA [i.5].

If it is a NS, these are primarily described in ETSI NFV MANO [i.2].

This includes the lifecycle management of the Virtualised SDN controller, and its relationship with the other entities of the NFV architectural framework. Based on our review of opensource SDN controllers available today, none of the ones studied, provide a software image and descriptors as currently defined by ETSI NFV. This could be an area of enhancement on their side.

An SDN controller existing as a VNF will ultimately be controlling resources that are physical or virtual and in effect be controlling PNFs or VNFs.

### 5.5.3 SDN controller across multiple Virtualised environment

In the present document a number of use cases have been studied where SDN controller are used in the ETSI NFV architectural framework. In all those cases the SDN controller is considered within the same Virtualised environment, with one NFV Orchestrator.

However there might be cases where the Virtualised SDN controller sits on top of multiple Virtualised environment, i.e. multiple operators and controls an SDN network that defines end to end paths across multiple operators.

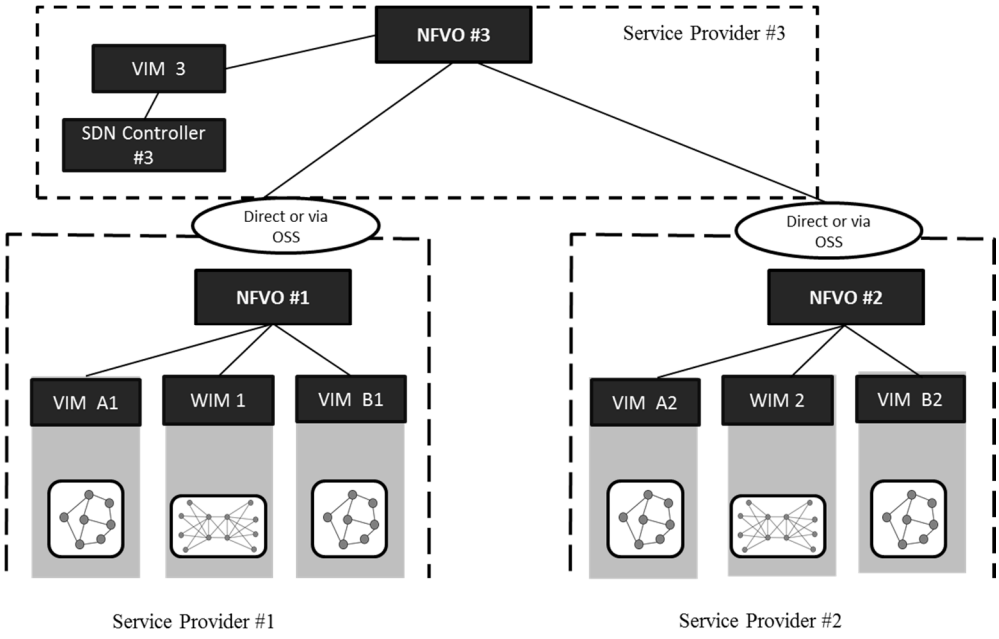



**Figure 27: SDN Controller of a Service Provider #3, Across Multiple Service Providers**

This use case which stretches beyond the current architectures covered by ETSI NFV raises a number of open questions such as:

- if SDN Controller #3 needs to control virtual resources deployed in SP#1 or SP#2, how this would be achieved needs to be addressed:
  - One option would be to go through the NFVO#3. In that case, the question is whether the NFVO#3 reaches NFVO#1 or NFVO#2 directly , or if there is an OSS in between the NFVOs.
  - Another option would be to go directly to the SDN resources or connect to an SDN controller deployed in the SP#1 or SP#2 NFVI PoP.

These use cases will become more and more popular as SDN is being deployed and combined with NaaS type of offering that will allow virtual network operators to define network services that span across multiple service providers and uses virtual resources deployed and reserved in the NFVI PoP of those service providers.

## 5.6 SDN and VNF forwarding graph

### 5.6.1 General

As described in ETSI GS NFV-MAN 001 [i.2], a VNFFG descriptor contains a list of Virtual Links to be established between its constituent VNFs and PNFs to form a Network Connection Topology (NCT) and might contain one or more Network Forwarding Path (NFP) elements associating traffic flows matching certain criteria to forwarding paths (i.e. the actual sequence of network functions to be traversed) within an NCT. The SDN technology might play a role for implementing both the NCT and NFP. The SDN controllers involved in the configuration of an NFP might be different from those involved in setting up the underlying NCT.

### 5.6.2 Static NCT

To implement a VNFFG, according to the architectural model described in clause 5.6 of ETSI GS NFV-MAN 001 [i.2], one or more SDN controllers would receive requests received from the NFVO, via one or more VIM or WAN infrastructure managers. Input parameters for these requests would be derived from the contents of the VNFFG descriptor.

An alternative implementation model, although not explicitly described in ETSI GS NFV-MAN 001 [i.2] would consist in omitting the NFPs from the VNFFG descriptor and relying on a service chaining application (hosted in the OSS or in a VNF) to request one or more SDN controllers to configure the appropriate routers and switches with a set of rules associating traffic flows matching certain criteria to forwarding paths. This approach might be relevant to implement the scenario described in clause 5.1.2.2.

The above cases assume that the definition of an NFP is relatively static while there are cases (e.g. see clause 5.1.2.3) where network forwarding paths are decided at runtime, e.g. by a VNF instance. This would require either:

- the ability for a VNF instance to request the NFVO to add or modify an NFP in a VNFFG;
- the ability for a VNF instance to access an SDN controller to provide NFP descriptions through means independent from NFV Management and Orchestration.

These procedures are not described in ETSI GS NFV-MAN 001 [i.2]. The first approach would require an evolution of the NFV architectural framework as it introduces a new reference point. In the second approach the communication between the VNF requesting the modification and the SDN controller could be seen as occurring at the SWA-1 reference point.

### 5.6.3 Dynamic NCT

Clause 5.5.2 assumes that the NCT is specified at the design time of a network service or is modified upon request of an OSS function. However, there might be cases (see clause 5.1.2.3) where an NCT would need to be modified at run time by a VNF instance. This would require the ability for a VNF instance to request the NFVO to modify a VNFFG. This procedure is not described in ETSI GS NFV-MAN 001 [i.2] and would require an evolution of the NFV architectural framework.

Once an NCT has been modified, modification of a related NFPs might be performed as per clause 5.5.2.

## 5.7 SDN controllers in the tenant and the infrastructure domains

There are two kinds of connectivity services in the NFV architectural framework, both of which require control capabilities and the orchestration and management of different types of resources for building and accomplishing a proper delivery, as NFV uses the network at two layers.

The first type of connectivity services are those provided by the NFVI to enable communication among VNFs and among their components, including the cases when those VNFs are instantiated in separated PoPs, reachable through a WAN connection. There is a clear understanding that SDN plays a key role to support the requirements on elasticity and Virtualisation for the infrastructural network to support the VNFs. This is the role of the *infrastructure controller*, managed by the VIM, and extensively discussed across the present document

The second deals with the network services provided at the service tenant layer (see note), and has to deal with the operation and management of the network service's constituent VNFs, and includes whatever semantics are related to the network service and that might be controlled by means of the SDN paradigm, through a programmatic, logically centralized control of the data forwarding plane (this would include functions like routing, filtering, service path selection, classification, etc.). The network services at this layer are suitable to be managed according to the SDN paradigm, and that is the proposed role of the *tenant controller*. It is important to highlight that applying SDN principles at this layer provides a concordance of the upper part of the NFV architecture with the general SDN proposal of a common control plane on top of which applications run. Those applications might range from enforcing intent-based policies provided by users to actually implementing OSS functions: the SDN model is general enough to accommodate all these uses.

NOTE: At the moment of this writing, the term "tenant" has no agreed definition within the ETSI NFV ISG. While such a definition is agreed, we use the term along this text in rather wide section, considering a tenant whatever entity (person, organizational unit, organization, etc.) using the NFVI to provide or manage network services.

While the first set of control actions are related to the semantics of the service itself (as intended by the deployment of the corresponding VNFs), the second set is related to the infrastructure resources. Despite of their different nature, all of these control actions have to be orchestrated in a consistent way to perform the expected service and to dynamically adapt to service changing conditions, as for instance the instantiation of new VNFs for that service. Furthermore, both kinds of control actions have to be comprehensively considered (i.e. synchronized and coordinated) at the same time.

The archetypal deployment of both controllers will allow to consider the infrastructure controller as providing the supporting underlay through the NFVI network Virtualisation layer, and the tenant controller providing an overlay composed of the tenant VNFs. From this point, relevant use cases are:

- Performing coordinated service function chaining by:
  - VNF composition, provided by the infrastructure controller through the NSD.
  - Dynamic VNF composition (according to user identity data, for example), provided by the tenant controller.
- Support for any hierarchical relationship between controllers running with awareness of VNF deployment and controllers running as part of the NFVI.

Figure 28 graphically presents the intended position of SDN controllers for both the infrastructure domain and the tenant domain within the NFV architecture framework.

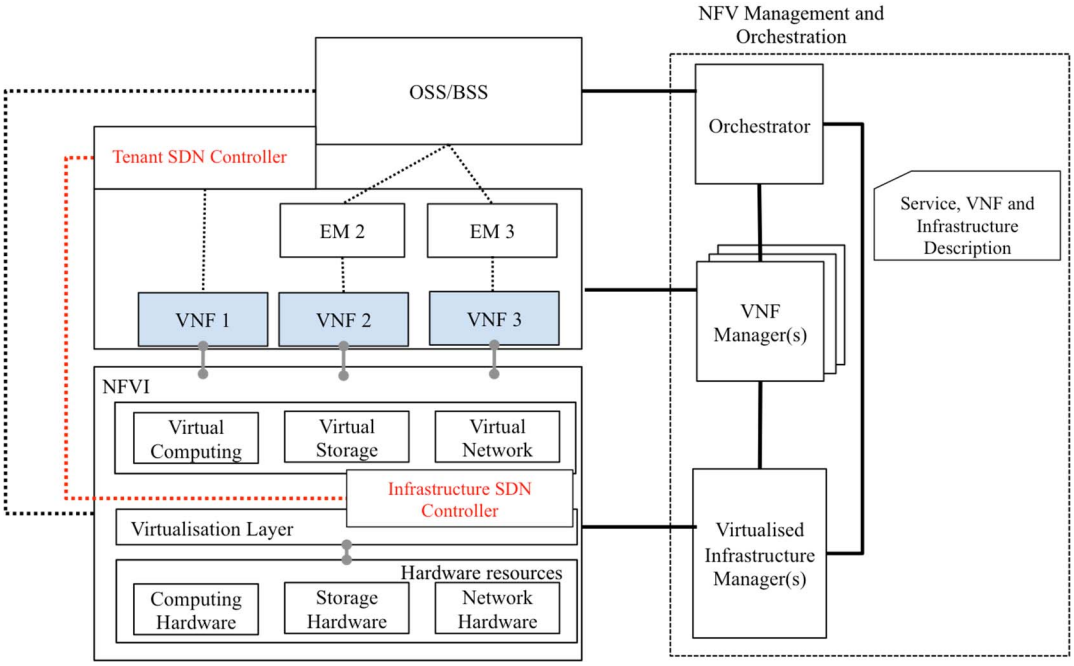



**Figure 28: Positioning Infrastructure and Tenant SDN Controllers in the NFV Architecture**

More detailed deployment options are shown in figure 29.

The SDN controller in the tenant domain, which might itself be a VNF, interacts with the different deployed VNFs in order to instruct those VNFs for taking actions on the traffic. On the other side, the SDN controller in the infrastructure domain is in charge of setting up the required connectivity (including the WAN) for communicating those VNFs. The SDN controller in the infrastructure domain supports the infrastructure network (as initially conceived by the original NFV proposal), while the SDN controller in the tenant domain provides a programmatic, and potentially simplified VNF management interface (note the intended direct connection to VNF 1 in the above figure 28, where this VNF 1 would be providing a direct SDN interface: OpenFlow, I2RS, etc.). Since the NFVO is in charge of orchestrating the NFVI resources and lifecycle management of network services, it might be completely orthogonal to the tenant SDN controller. NFVO is to be concerned precisely about the policy and rules being applied by the infrastructure controller.

There is a need for coordination between both controllers (if service chaining is performed at both layers it would be highly desirable, for example), though whether both controllers might interact directly through a new reference point as shown in figure 28, or indirectly via the MANO stack requires further study. Note this is coordination, not implying a direct control of the infrastructure controller by the tenant one, or vice versa. On the one hand, the MANO stack could provide the tenant controller abstracted information about the virtual infrastructure network and allow some degree of interaction in both directions. On the other hand, using the MANO stack would require some extensions to the interfaces being currently considered, and it could probably violate the decoupling between MANO and network service semantics. In any case the reference point depicted there would be out of the scope of NFV specifications, as it is the case of the reference point between the OSS/BSS and the infrastructure domain, and would require the availability of a specific ACI (Application Control Interface) for the infrastructure controller, as it is foreseeable that the abstraction supported through this interface would be different to the one provided by the infrastructure controller to the VIM.

Figure 29 shows two deployment options for the tenant controller, either as a VNF and therefore part of a network service deployment, or being part of the (new generation) OSS and therefore able to interact with several network services and even several tenant deployments if required. In both cases, the tenant controllers would be able to interact with the OSS either through a direct interface or by accessing data and policy repositories, such as HSS or PCRF.

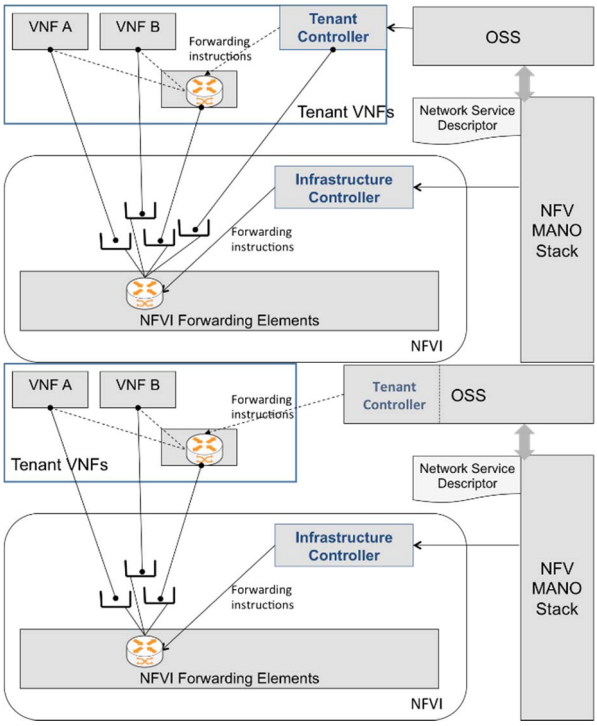



**Figure 29: Deployment Options for the Tenant Controller**

# 5.8 Service Function Chaining

## 5.8.1 Introduction

The following use case describes Service Function Chaining, SFC, combined with SDN and the ETSI NFV architectural framework.

## 5.8.2 Service Function Chaining (SFC)

Defined by IETF, the SFC architecture is described in [i.9] and has identified a number of components as shown in figure 30 such as:

- SFC aware network functions and SFC unaware network functions
- SFC Proxy
- SF Forwarder
- SFC Classifier

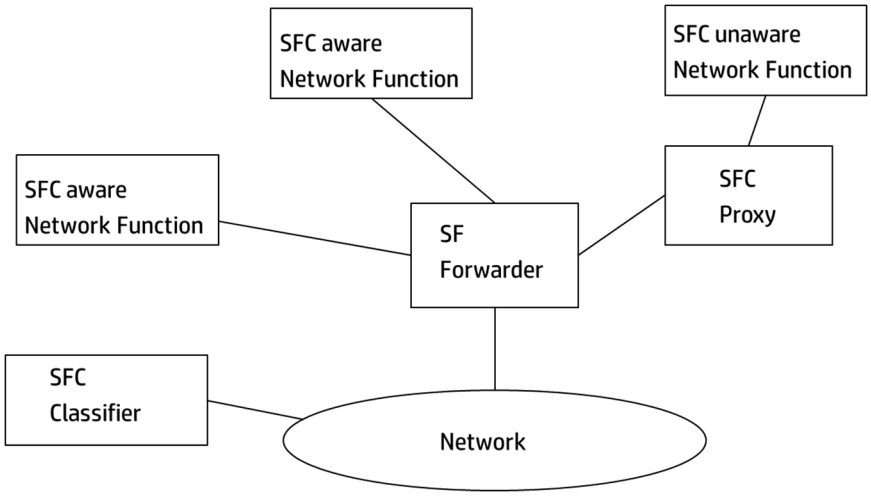



**Figure 30: SFC Architecture Components (as defined by IETF)**

The basic SFC architecture in IETF [i.9] does not consider SDN explicitly, neither NFV. IETF SFC is work in progress.

IETF has recently suggested a draft to address SFC with SDN/NFV [i.10]. The following clauses will propose some combination of SFC with SDN and NFV.

In the present document, the term SFC network function is used; however the IETF term 'service function' is used for what ISG NFV calls 'network function'.

### 5.8.3 SFC and SDN

In figure 30, and other figures in the clause 5.8, the term network is used as a generic term for networking resources carrying the traffic.

When SDN is being introduced in this network layer, that network is then composed of SDN resources, controlled by an SDN controller, as described in figure 31.

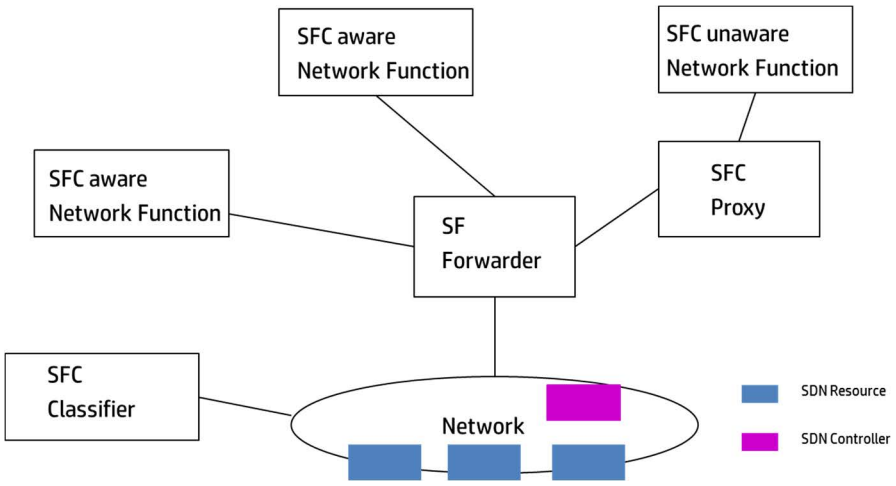



**Figure 31: Adding SDN to SFC Underlying Network**

The other case is to have SDN introduced in the SFC functions. In this situation, the SFC Classifier and the SF Forwarder could become SDN resources, and be controlled by an SDN controller, as described in figure 32.

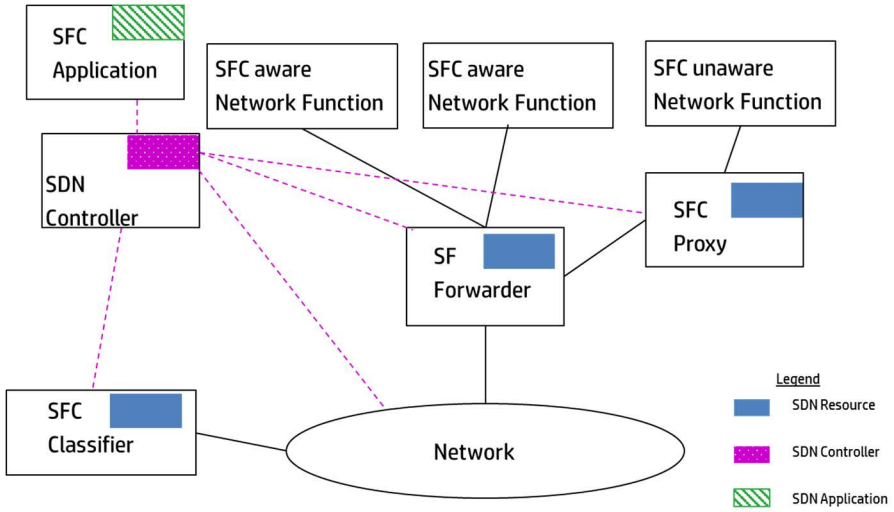



**Figure 32: Adding SDN to SFC Components**

This is typically the case in the diagram below, figure 33, where incoming traffic flows through an SFC Classifier being an SDN resource in that case, controlled by an SDN controller, then goes to a Service Function Forwarder to an SFC aware function, then back to the SF Forwarder to another SFC function but SFC unaware using an SFC Proxy.

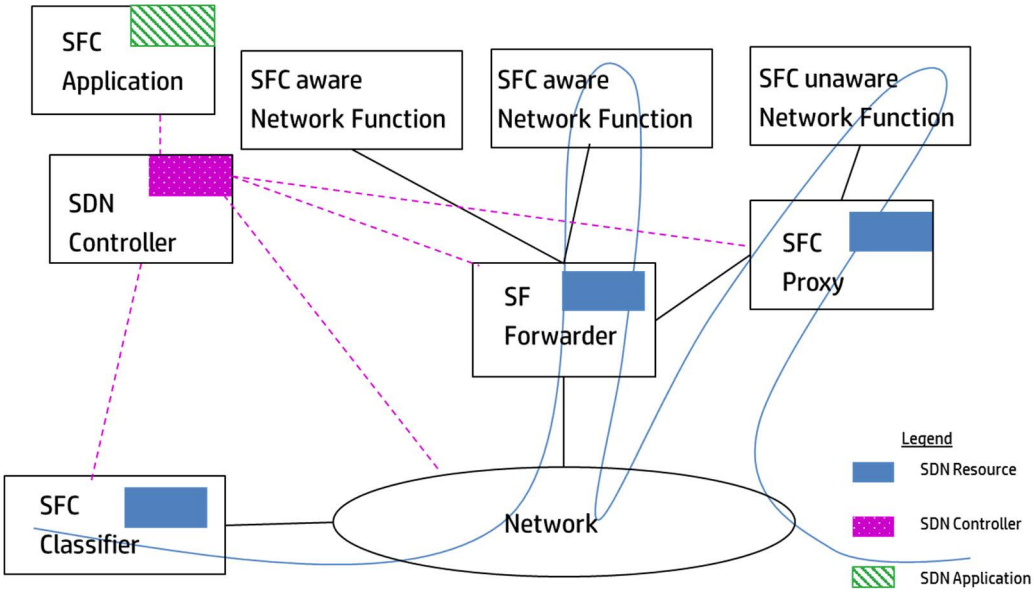



**Figure 33: Traffic Flow with SFC and SDN**

The entire SFC flow is defined by the SFC Application. This is an SDN application that implements the SFC service using the SDN controller. The flow might go through multiple SF Forwarder (SFF) typically, this diagram is a simplified view.

Also traffic might brand out after the SFC classifier or anytime within the path.

## 5.8.4 SFC, SDN and NFV with a single NFVI domain

### 5.8.4.0 Introduction

As NFV is being introduced, any of the components defined above might potentially be Virtualised and deployed on a Virtualized infrastructure managed by a Virtualisation management & orchestration environment, as described in figure 34.

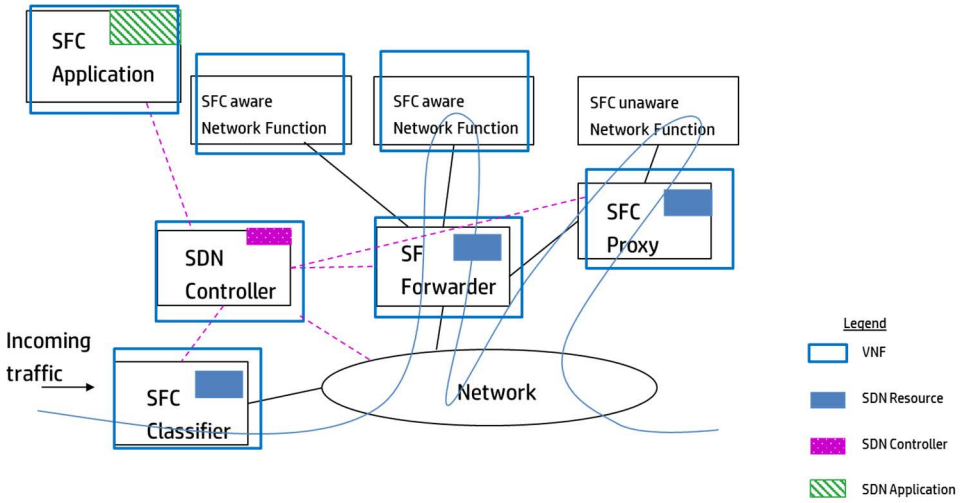



**Figure 34: Virtualisation of the SFC Functions with SDN Controller as a VNF**

Multiple cases might be envisioned:

- Case a: the Service Function Chain is defined in the NFVI. The SDN controller is deployed in the NFVI.
- Case b: the Service Function Chain is defined in the tenant domain. Some virtual resources are allocated to the tenant who deploys his service chaining functions, typically as an overlay network, with his virtual switching SDN enabled and his SDN controller.

### 5.8.4.1 Service function chain in the NFVI

In this case, SDN is used either in the NFVI underlay, meaning on native switches and routers, or as an overlay with virtual switches and routers SDN enabled on top of non SDN switches and routers. In both cases, an SDN controller is deployed in the NFVI, figure 35, connected to the VIM, and the NFPs are used to control the service chaining. Example of such configuration include using SDN switches or vswitch in the NFVI, with an OpenDaylight® (see note) SDN controller connected to an OpenStack Neutron, like in POC#8 , and in line with MANO [i.2] figure 5.2.as shown in figure 35. The SDN routers/switches of the NFVI, would play the SFF role.

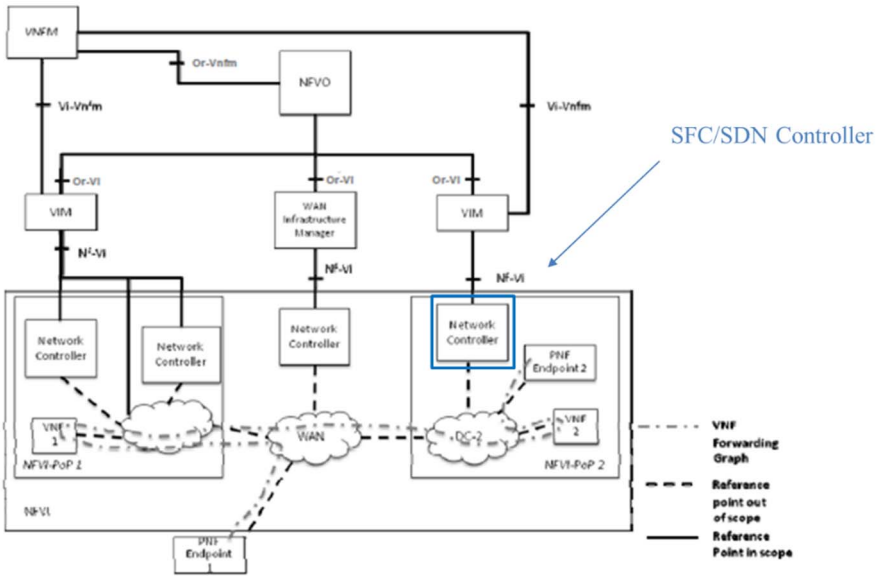



**Figure 35: SDN Controller in NFVI (based on MANO [i.2] figure 5.2)**

NOTE: "Opendaylight is the trade name of a collaborative open source project from The Linux Foundation. This information is given for the convenience of users of the present document and does not constitute an endorsement by ETSI of the product named. Equivalent products may be used if they can be shown to lead to the same results".

The Network Services (NS) as defined by ETSI MANO , as shown in figure 36, would then be deployed on this SDN enabled Virtualized infrastructure. Considering NS within a same VIM domain, a NS could go through multiple VNF and potentially PNF. Virtual Links (VL) would be defined between the VNFs as shown in figure 6.5 of ETSI GS NFV-MANO [i.2]. A set of VLs in a NS form a Network Connection Topology (NCT) as described in ETSI GS NFV-SWA 001 [i.5]. A VNFFG might reference other information elements in the NS such as PNFs, VLs and VNFs. A VNFFG also contains a Network Forwarding Path (NFP) element. In our case, SDN is used to setup a Network Connectivity Topology (NCT) and several Network Forwarding Paths (NFPs). Then depending on the type of traffic, different Network Function Path (NFP) will be used to carry the traffic through appropriate VNFs. As new elements, VNFs or PNFs, are introduced in the service chain, a new NCT and a new set of NFPs is being defined.

NSs are typically managed by the NFVO based on their NSD which defines the VNFFG and NFPs.

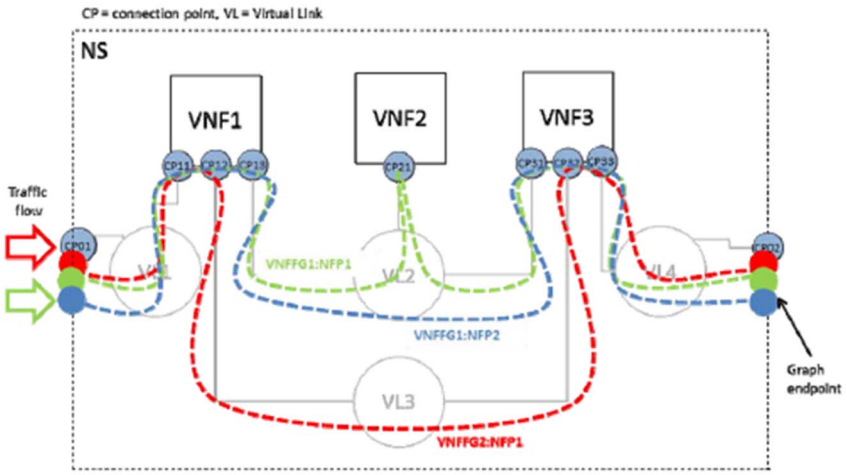



**Figure 36: VL between VNF (i.e. SFC VNF), and VNFFG/NFP as defined in ETSI GS NFV-MAN 001 [i.2]**

In summary, as SFC defined earlier is mapped with the ETSI NFV architectural framework in the case where the SFC is defined in the NFVI, the following elements are introduced:

- Some SFC service function might be Virtualized and become VNFC or VNF: SFC aware functions, SFC unaware functions, SFC Classifier, SFC proxy, SFC/SDN Controller and SF Forwarder.
- Others might remain non Virtualised.
- The SFC/SDN controller is deployed in the NFVI and typically connects with the VIM.
- The SDN routers/switches, Virtualised or not, underlay or overlay, play the role of SF Forwarder.
- An NFV Orchestrator is introduced to deploy the VNF and manage the NS that will define the NCT and NFPs used by the Service Function Chaining. It will interface with the VIM and the VNF Manager(s).
- Descriptors will be defined for VNF, VL, PNF, VNF-FG and NS.

This mapping is illustrated in figure 37.

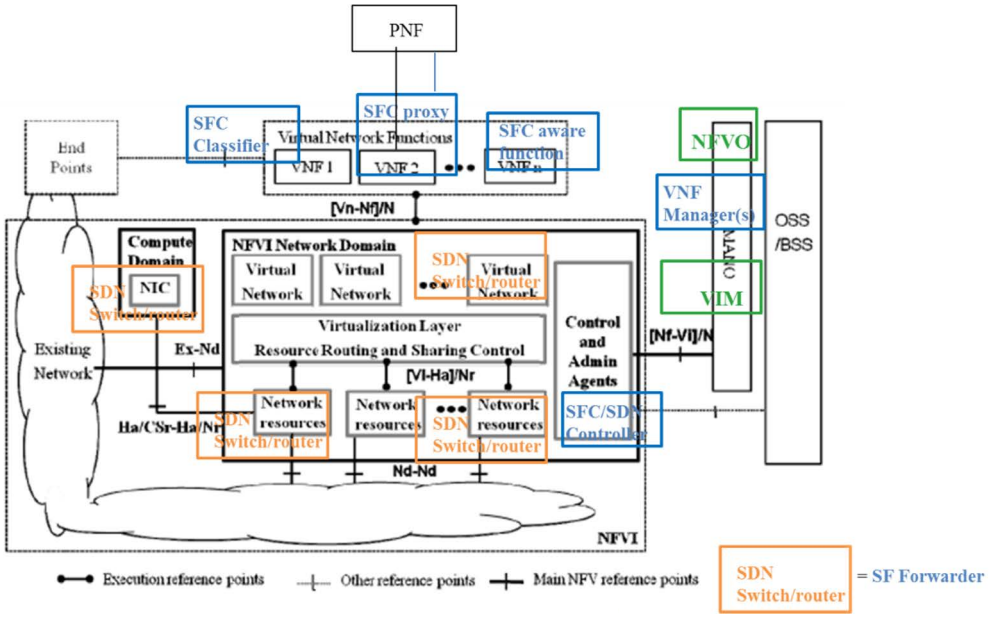



**Figure 37: SFC with SDN and NFV based on figure 3 ETSI GS NFV-INF 005 [i.1]**

### 5.8.4.2 Service function chain in the tenant domain

The second case, illustrated in figure 38, is to consider the SFC in the tenant domain. In this case the SFF and the SDN controller are VNFs and are part of a tenant NS. An appropriate NCT has been setup in the NFVI.

NOTE 1: SDN might be used by the NFVI to setup this NCT but this is internal to the NFVI and not visible to the SFC.

Once the virtual resources have been allocated to a tenant, he might deploy his VNF, SFF and SDN controller, but also SFC classifier and necessary SFC proxy, and define service chains depending on the classification of the traffic. These service chains might be uploaded on the SDN controller to be implemented on the SF Forwarder and evolve dynamically. Typically the service chains are often defined within a traffic steering/service chaining application that sits on top of the Application Control Interface of the SDN Controller.

NOTE 2: Some implementation use the NFV Orchestrator to define these service chains, leveraging the fact that the NFVO has the view of the end to end topology. In that case this might require the NFVO to have a direct link to the SDN controller.

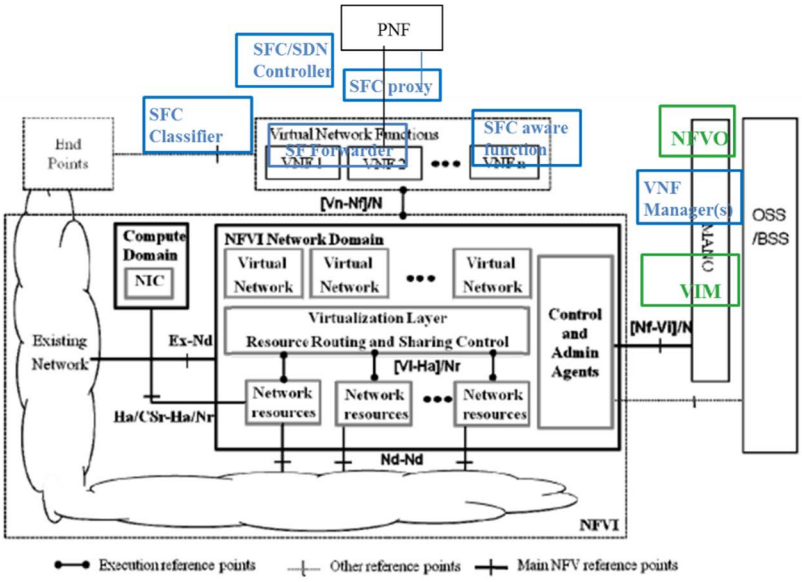



**Figure 38: SFC in the Tenant Domain**

### 5.8.4.3 Different options to control a dynamic service chain

The following figures 39 and 40 show how the two architectural options described earlier relate to the role of the optional NFPs in the NSD.

Figure 39 depicts a configuration where service chaining solely relies on switching and routing capabilities provided by the NFVI. The SDN controller in the NFVI receives all necessary instructions from the VIM. This includes instructions on the virtual links to be setup and on the forwarding rules to be applied to particular flows. This approach requires VNF Forwarding Graph descriptors to include one or more NFP descriptor.

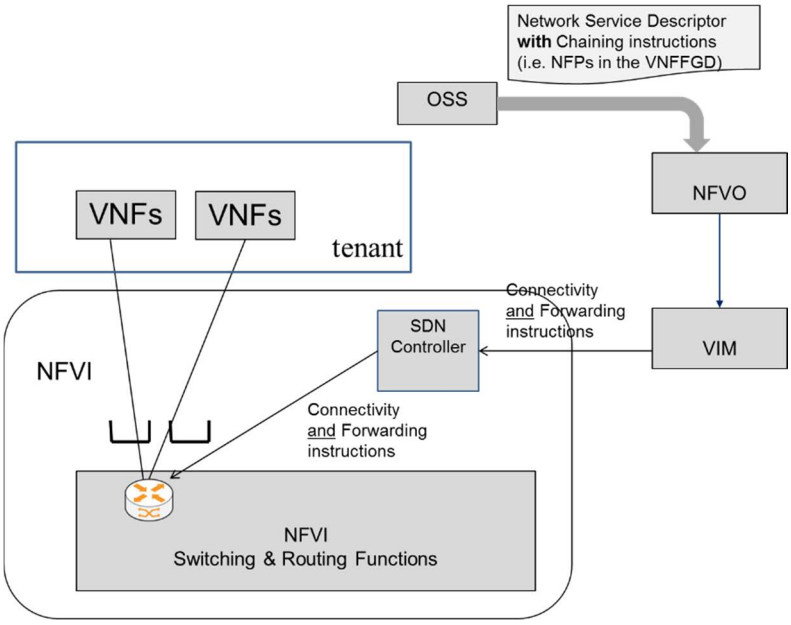



**Figure 39: NFV Service Chaining with Infrastructure Support**

Figure 40 depicts a configuration where service chaining relies on virtual switching and routing capabilities provided by VNFs as part of a Network Service (tenant). One of these VNFs is an SDN controller, which provides forwarding rules to one or more virtual router or virtual switch implemented in the form of VNF as well. These forwarding rules determine the paths that traffic flows between the other VNFs follow. These traffic flows are carried over a set of virtual links established with the support of another SDN controller in the NFVI. This approach assumes Forwarding Graph descriptors do not include any NFP descriptor.

NOTE: The SDN controller is here in the NFVI, but it could be elsewhere, i.e. in the VIM.

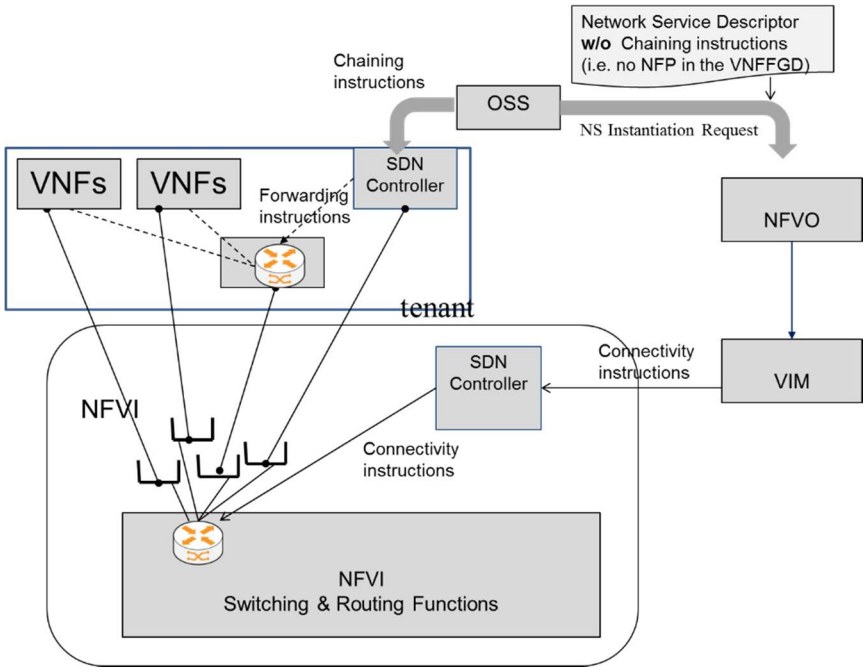



**Figure 40: NFV Service Chaining in the Overlay (Tenant) Network**

## 5.8.5 End to end carrier network with SFC, SDN and NFV

Within carrier networks, multiple access networks are being provided such as mobile, wifi, xDSL, cable, FFTH, etc.

As network and service providers move network functions into the cloud as shared resources in a data center, and want to optimize the usage of these resources, especially for video traffic for instance, with NAT, firewall, video optimization, TCP optimization etc., they introduce SFC and SDN in the data center for instance. Besides coping with traffic increase and getting more flexibility, they also introduce NFV in the data center.

Typically in the example below, figure 41, the different access networks leverage the service function chaining implemented in the NFVI of a common data center.

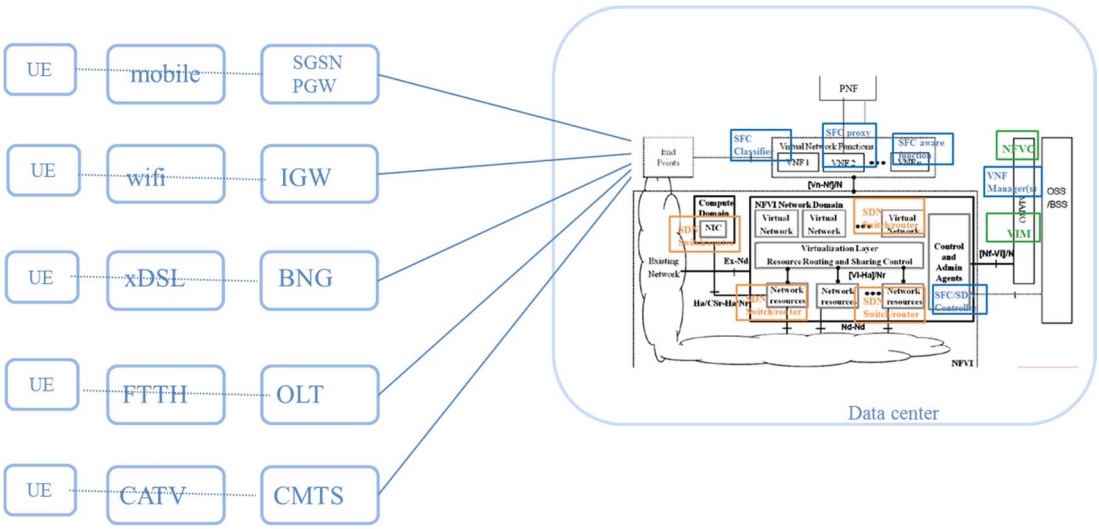



**Figure 41: End to End Carrier Network with SFC, SDN, NFV in the Data Center**

### 5.8.6 SFC, SDN, NFV with multi-domain, scalability, etc.

As the system evolves with multiple NFVI-VIM domains, scalability of the VNF, failover with geographic redundancy across multiple domains, the SFC environment has to become very dynamic, flexible and performant. In the case of the SFC in the NFVI, the NFVO has to define the service chains, NS, across multiple VIM with SDN controllers as illustrated in figure 42. As the service chain evolve dynamically, their implementation through the different layers of APIs, VIM API, SDN controller API, has to perform to avoid latency in the network. The NFVO might also need to control the WIM in order to define chains that span across WANs.

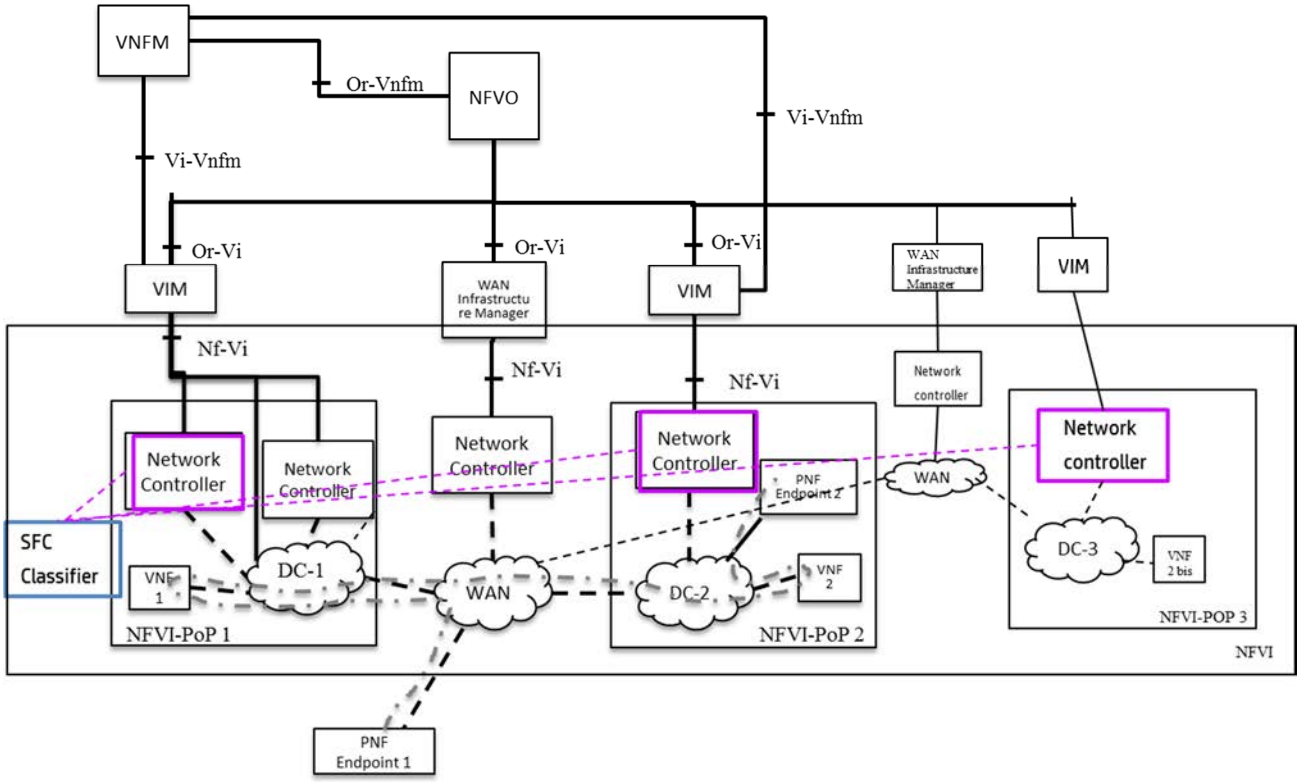



**Figure 42: SFC in the NFVI with SDN and NFV in a Multi-Domain, Scalable Environment**

After the VNF2 scalability request which creates a VNF 2bis under NFVI-PoP 3, or the VNF2 failover with geographic redundancy to NFVI-PoP 3, as illustrated in figure 42, new virtual links might have to be defined, with new VNF-FG as shown in figure 43.

The service chain then evolves not only by defining new paths among static Network Connection Topology (NCT), but also to adapt to dynamic NCT.

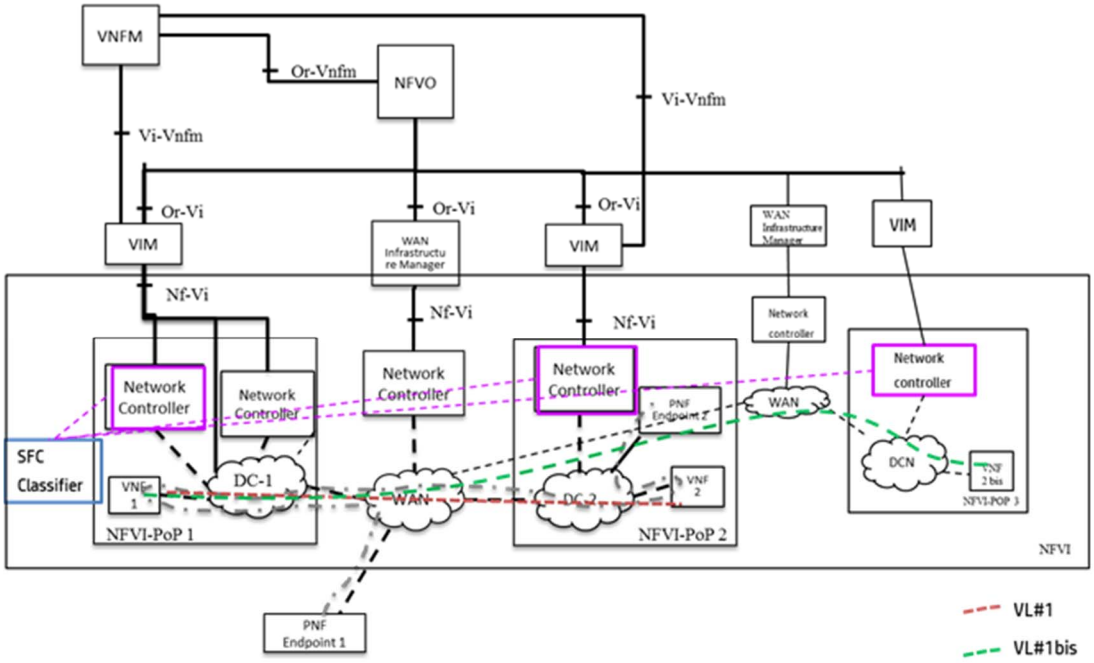



**Figure 43: SFC in the NFVI with SDN and NFV in a Multi-domain and virtual links**

Latency has been mentioned when provisioning a service chain through a layer of APIs. But requirements could also come with service chains themselves, such as guaranteed low latency over a given E2E network service, which brings some further enhancements of the ETSI NFV specifications in terms of resource capacity exposure and E2E path latency calculation before allocating and configuring NFPs, but also proper monitoring to measure this latency.

# 6 Functional recommendations

## 6.1 Introduction

The current list of recommendations is the following:

REC#1:

- Use case: SDN controller and associated SDN network resources need to be able to communicate, especially during scaling and migration of any of these components.
- Recommendation: it is suggested that a requirement be specified for the NFV architectural framework to enable a given SDN controller to always be able to communicate with its associated SDN resources and provide a mechanism to associate the two.
- Comment: this is particularly sensitive during scaling or migration of any of these components. This communication and the association procedure have to be made in a secure way. When the SDN technology is used in the tenant domain, the above requirement might be fulfilled by grouping the VNFs and/or PNFs corresponding to these components in the same NS and by specifying suitable requirements in the associated virtual link descriptors.
- Clause: 4.2.

REC#2:

- Use case: SDN controller is a PNF.
- Recommendation: it is suggested that a requirement be specified for the NFV architecture framework to support SDN controller being a PNF.

- Comment: further study might be conducted to analyse the impact of having an SDN controller PNF interacting with the NFV architecture framework to access NFV resources or SDN controller VNF for instance.

- Clause: 4.3.4.

REC#3:

- Use case: VNF might be deployed in Docker® containers. Virtualised SDN components (NW resource, controller or application) might be VNF or VNFCs, but also SDN controller might have to configure virtual links and network forwarding path across resources in Docker.

NOTE: "Docker is a registered trademark of Docker, Inc. This information is given for the convenience of users of the present document and does not constitute an endorsement by ETSI of the product named. Equivalent products may be used if they can be shown to lead to the same results".

- Recommendation: it is suggested to further study NFV management with SDN control & Docker container based VNF.

- Clause: C.3.2.

REC#4:

- Use case: interaction between tenant and infrastructure controller, or interactions between multiple SDN controllers as illustrated with East-West interface for instance that might be located in different building blocks on the NFV architecture.

- Recommendation: it is suggested to further study the controller-controller interfaces.

- Comment: for the tenant to infrastructure interactions, one of the key points of such a study will be to assess about using for this purpose dedicated, direct inter-controller interfaces (realized by any of the means already proposed, from protocols like BGP to specialized topology abstractions directly connected to the SDN model being used by the controllers), or to employ specific messages or parameters through the MANO stack.

- Clause: 4.4.1.

REC#5:

- Use case: intent-enabled interface is being used to interface with the SDN controller.

- Recommendation: it is suggested to further study the impact of intent-enabled interfaces on the NFV technologies.

- Comment: intent-enabled interfaces will constitute one of the essential mechanisms for providing network users access to network services, and therefore constitute a key aspect for interaction among VNFs and the NFVI, among VNFs themselves, and among NFV-enabled services and their users.

- Clause: 4.4.1.

REC#6:

- Use case: the NFVO might need access to an SDN controller or to an SDN application, in order to pass information between the 2 entities, such as topology information.

- Recommendation: It is suggested that a study be started to assess whether to support an SDN controller orchestration interface between the NFVO and an SDN controller in the cases where the SDN controller is implemented as a VNF or inside the NFVI. The study would also assess whether to support the same interface between the NFVO and an SDN application.

- Clause: 4.4.3.

REC#7:

- Use case: in case the SDN controller is a tenant SDN controller (case 4), an interface might exist with the infrastructure SDN controller (case 3).

- Recommendation: it is suggested that further study is conducted on the interactions and interface needed between a tenant SDN controller and an infrastructure SDN controller.
- Clause: 4.5.1.

REC#8:

- Use case: in case of paths set up in an NFV architecture with different service classes across multiple VIM and multiple WAN domains, the respective WAN domain capabilities and connectivity endpoints need to be available.
- Recommendation: it is suggested that WAN domain capabilities and connectivity end points requirements be specified when one or more WAN domains are involved via WIM.
- Clause: 5.3.4.2.0.

REC#9:

- Use case: in case NFVI-PoP interconnect via a WAN, a point-to-point connection between two NFVI-PoPs is transparently carried across the WAN over a transport tunnel by means of technologies such as MPLS or IEEE 802.1ad. Each endpoint is aware of the specific characteristics (e.g. bandwidth capacity) of the inter-NFVI-PoP connection, but the WAN network nodes are only aware of the transport pipe. Thus, the lifecycle of the inter-NFVI PoP inner connection (e.g. establishment, modification, release) is controlled by the intervening VIMs, but not the WAN domain(s) in between.
- Recommendation: it is suggested that further study is conducted to clarify how VIMs might request connectivity to the WAN domain in case of interconnected VIMs via WAN.
- Comment: more details can be found in the present document, clause 5.3.2.
- Clause: 5.3.4.2.1.

REC#10:

- Use case: the inner connection management controlled by the intervening VIMs might request direct connectivity across the WAN domain.
- Recommendation: it is suggested that further study is conducted on how VIMs might request direct connectivity across the WAN domain.
- Comment: VIM could connect to the WAN domain via OSS, via NFVO or directly.
- Clause: 5.3.4.2.1.

REC#11:

- Use case: NFVI-PoPs might be interconnected via one (figure 15) or multiple (figure 16) WAN domain, each being managed by the same or different providers. In case of different providers, they might be different NFVO (figure 16) managing different WAN domains.
- Recommendation: it is suggested that further study be conducted to analyse the relationship between each NFVI-PoP and the respective WAN domains/providers , in particular with regards of the role of the NFVO.
- Clause: 5.3.4.2.2.

REC#12:

- Use case: a VNF processing might trigger a modification of network path and require changes to a dynamic VNF service chaining. The VNF instance requesting that change might need to ask the NFVO to access to the northbound interface of an SDN controller to implement that change.
- Recommendation: it is suggested to further study this sort of direct access from a VNF to the NFVO to evaluate if a new interface is needed between VNF and NFVO in the ETSI NFV architecture.

- Comment: today interfaces exist between VNF and VNF, or a VNF and its VNF manager, but not to an SDN controller in the NFVI or VIM. The NFVO would validate the request, check that SDN controller location and authorize access or perform the change.
- Clause: 5.6.2.

## 6.2 Functional recommendations related to security

REC#SEC.1:

- Use case: attacks mounted via the Forwarding Plane. Examples are reconnaissance attacks, DoS and resource exhaustion attacks and vulnerability exploits.
- Recommendation: it is suggested that a requirement be specified to prevent attacks mounted via the Forwarding Plane against SDN switches and controllers.

REC#SEC.2:

- Use case: attacks from the control network. Examples are attacking the integrity and confidentiality of the controller-switch traffic, exploiting vulnerabilities of controller-switch interface implementations, DoS and reconnaissance attacks.
- Recommendation: it is suggested that a requirement be specified to mitigate attacks from the control network.

REC#SEC.3:

- Use case: attacks via the SDN Controller's Application Control Interface. Examples are vulnerable Application Control Interfaces, malicious or faulty applications running as part of the controller, unsecure protocols at the NBI, applications that are not authenticated and authorized, malicious or unexpected network control by (multiple) applications (rerouting flows, changing header fields to evade security policies also between different tenants).
- Recommendation: it is suggested that a requirement be specified to mitigate attacks via the SDN Controller's Application Control Interface.

REC#SEC.4:

- Use Case: Attacks against controllers and switches via the Virtualised environment. Compromising the Virtualised environment (e.g. hypervisors) might also compromise Virtualised switches, controllers and applications. Examples are unreliable isolation between the different switches or control of SDN controllers through errors in hypervisors and virtual machines.
- Recommendation: It is suggested that a requirement be specified to mitigate attacks against controllers and switches via the Virtualised environment.

## 6.3 Functional recommendations related to SDN controller

REC#13:

- Use Case: SDN controller located in the NFVI.
- Recommendation: it is suggested that a further study is conducted to clarify the exact location of an SDN controller in the NFVI according to NFV-INF architecture building blocks (ETSI GS NFV-INF 005 [i.1], figure 2).
- Clause 4.3.4.

REC#14:

- Use Case: the SDN controller might request the dynamic creation, deletion, modification of VNF Forwarding Graphs under its control.
- Recommendation: it is suggested to further study the NS lifecycle management request coming from SDN controller to the NFV Management & Orchestration.

- Comment: this use case only makes sense in the case of an SDN controller in the tenant domain requesting modification of a VNFFG in the same domain.

REC#15:

- Use Case: the SDN controller might use a semi-static resource inventory of various VNF types and instance as components in its own service offerings.
- Recommendation: it is suggested to further study the access or synchronization of NFV MANO repositories with SDN repositories, i.e. for VNF instance repository.
- Comment: this use case only makes sense in case of an SDN controller in the tenant domain. The NFV architecture holds metadata and instance repository. Interfaces to these repository are not clearly defined today.

REC#16:

- Use case: an SDN controller might relocate traffic to make better use of existing resources or might request additional resources. Rather than re-routing traffic, the NFV environment scales and migrates resources, driven by its own measurements or by request from another domain.
- Recommendation: it is suggested to further study the case where traffic steering or some capacity issue triggers actions and has to choose between rerouting traffic, i.e. asking SDN controller to reroute traffic, or scale resources, VNF or NS.
- Comment: the SDN controller has some understanding of the NW and the available options, NFV MANO has some understanding of the available resources and virtual functions, these 2 are complementary and could be coordinated to make best decision while optimizing resource usage & deliver best quality of service.

## 6.4 Functional recommendations on connectivity and interfaces

REC#17:

- Use case: MANO functional entity might request an SDN controller to change SDN traffic redirections dynamically at line rate.
- Recommendation: it is suggested that a requirement be specified for the interface between relevant MANO functional entities and SDN controller to provide low latency.
- Clause: 5.2.2.

REC#18:

- Use case: when presented with a service request, an SDN controller chooses instances of VNFs, FGs, and non-NFV resources, according to criteria that might include endpoint location, geographic or topological proximity of available resources, delay, aggregate or fine-grained load, monetary cost, fate-sharing, or other factors.
- Recommendation: it is suggested to further study the interface between NFV MANO and the SDN controller to address some of the SDN controller request such as monetary cost and delay for instance.
- Comment: some of these parameters are technical and might come from the infrastructure, i.e. delay, but others might be more business related, i.e. cost.

REC#19:

- Use case: MANO Network Forwarding Path provides classification and selection rules as attributes. These attributes and associated policies are passed to the SDN controller that has its own policies. Some correlation has to be performed.
- Recommendation: it is suggested to further study policy management between NFV MANO and SDN controller.
- Clause: B.4.4.

REC#20:

- Use case: some resources of an NFVI might be physically dedicated for use by NFV-MANO, but many resources, especially in the WAN, will be shared with other NFV network domain, or with other non- NFV network domain, and in particular with SDN. NFV-MANO and SDN coordinate their claims to the shared resources.
- Recommendation: it is suggested to further study the coordination of concurrent claims coming from SDN controller or NFV-MANO to shared resources in an NFV environment.

## 6.5 Functional recommendations on NFV Management and Orchestration

### 6.5.1 General

This clause describes functional recommendations on the NFV-MANO as following the general principles cited in ETSI GS NFV-MAN 001 [i.2] (particularly clauses 5.6.2 and 5.6.3) and in clause 5.2.2 of the present document.

The following list expresses the non-exhaustive recommended functionalities performed by the NFV-MANO. Those functionalities might be exposed by means of interfaces consumed by the MANO functional blocks or by authorized external entities:

- inter-administrative domain connectivity coordination;
- support of operations to an SDN controller below the VIM;
- support of ordering and charging across multiple administrative domains.

### 6.5.2 Inter-administrative domain connectivity coordination

The inter-administrative domain connectivity coordination is responsible for managing virtual links dynamically and consistently on top of the underlying NFVI network resources across multiple administrative domains by means of programmatic interfaces produced by the network controller in each administrative domain. In the setup and control of virtual links, the inter-administrative domain connectivity coordination requests each administrative domain to play any of the three roles: sender, receiver or relay domain.

- The sender domain is required to be informed of which addresses and/or identifiers are destined and which administrative domain(s) are the next.
- The receiver domain is required to be informed of which addresses and/or identifiers are received.
- The relay domain is required to be informed of which set of addresses and/or identifiers is relayed and which Administrative Domain(s) are the next.

The inter-administrative domain connectivity coordination informs them of their required information.

REC#21:

- Use case: the NFV-MANO functional blocks or authorized external entities might set up partial virtual link within each administrative domain dynamically and consistently across multiple administrative domains based on the result of the inter-administrative domain connectivity coordination; in such cases for inter-VIM SDN as described in clause 5.3.4.2.
- Recommendation: it is recommended that a requirement be specified for the NFV-MANO to ensure that administrative domain(s) are provided with enough information to ensure that the proper network connectivity role is performed by the SDN controller(s), including:
  - For SDN across VIM in different NFVI-PoP, with pre-allocated static WAN connectivity service (see clause 5.3.4.2.1), the inter-administrative domain connectivity coordination (which is played by VIM of a sender administrative domain) is informed about destination addresses and/or identifiers of virtual links to reach the receiver administrative domain.

- For SDN across VIM in different NFVI-PoP with SDN-based NaaS between NFVI-PoPs (clause 5.3.4.2.2), the inter-administrative domain connectivity coordination (which is played by WIM) is informed about origin and destination addresses and/or identifiers of virtual links so that WIM might determine the role of each administrative domain to configure the end-to-end connectivity.

### 6.5.3 Support of operations to an SDN controller below the VIM

REC#22:

- Use case: the VIM may need access to an SDN controller that is in the NFVI.
- Recommendation: it is suggested that a requirement be specified for the Nf-Vi interface to support operations going to an SDN controller.

### 6.5.4 Support of ordering and charging operations between multiple administrative domains

REC#23:

- Use Case: some use cases address interworking between multiple administrative domains (different trust domains), sometimes between competitive companies. Some NFV services span across multiple administrative domains with NFV/SDN interworking. These cases will require attention to many factors such as ordering and charging mechanisms, inter-administrative domain security considerations and the need for standardized service descriptions and identifiers.
- Recommendation: it is suggested to further study the requirements for interworking between multiple administrative domains using NFV and SDN, including ordering, charging, and inter-administrative domain security requirements.
- Clause: 5.5.3 figure 27.

## 6.6 Recommendations on operational aspects

The following are proposed operational recommendations based on IETF RFC 5493 [i.18] which discusses supporting conversions between distributed control and centralized control of network resources. These scenario are considered when introducing SDN in a network or evolving the existing SDN infrastructure: new SDN controller instance, migration, etc.

REC#24:

- Use case: transfer from/to management plane (or control Plane) to SDN control.
- Recommendation: it is suggested that a requirement be specified to transfer the ownership of resources from a management plane (or control plane) to SDN control (and vice versa).

REC#25:

- Use case: infrastructure consistency during control transfers from/to management plane (or control Plane) to SDN control.
- Recommendation: it is suggested that a requirement be specified for the infrastructure resources and the NFV environment need to stay in place throughout any control transfer or control update process. Services need to follow the same path through the network and use the same network resources.

REC#26:

- Use case: no Disruption of user traffic or alarms towards end users during control transfers from/to management plane (or control plane) to SDN control.
- Recommendation: it is suggested that a requirement be specified for control transfer process not to cause any disruption of user traffic. No alarms have to be generated towards the end users.

REC#27:

- Use case: no alarms to end users traffic during control transfers from/to management plane (or control plane) to SDN control.
- Recommendation: it is suggested that a requirement be specified for no alarms to be generated towards the end users during control transfers.

REC#28:

- Use case: synchronization of control state during conversion, transferring control from management plane to SDN control.
- Recommendation: it is suggested that a requirement be specified to assure that the control state of the service path is synchronized across the resources before the control conversion is considered complete.

REC#29:

- Use case: support for more than one control entity.
- Recommendation: it is suggested that a requirement be specified to be possible to segment a service path/resources under different control domains (co-existence).

REC#30:

- Use case: failure of transfer, transferring control from management plane to SDN control.
- Recommendation: it is suggested that a requirement be specified to be possible for a transfer from one control entity to another to fail in a non-destructive way, leaving the ownership unchanged and without impacting traffic.
- Comment: if during the transfer procedure issues arise causing an unsuccessful or unexpected result, it needs to be assured that: a) user traffic is not affected, b) the service path state is not impacted. Point b) assures that even in case of some failure during the transfer, the state of a service path/resources is brought back to the initial one and is fully under the control of the owning entity.

REC#31:

- Use case: security and policy considerations, transferring control from management plane to SDN control.
- Recommendation: it is suggested that a requirement be specified to support security and policy mechanism that would prevent from malicious intervention during transfer of control from management plane to SDN control.
- Comment: allowing control transfers introduces a possibility of malicious intervention. It is recommended that appropriate security and policy mechanisms are supported.

# Annex A (informative): SDN in ETSI NFV POC

## A.0 Introduction

This annex provides a summary of the ETSI NFV PoCs demonstrations involving the SDN technology. Any deviation between the solutions implemented for these PoCs and the ETSI NFV architectural framework is not to be understood as a proposal to modify this architectural framework.

## A.1 POC#1: Open NFV Framework Project

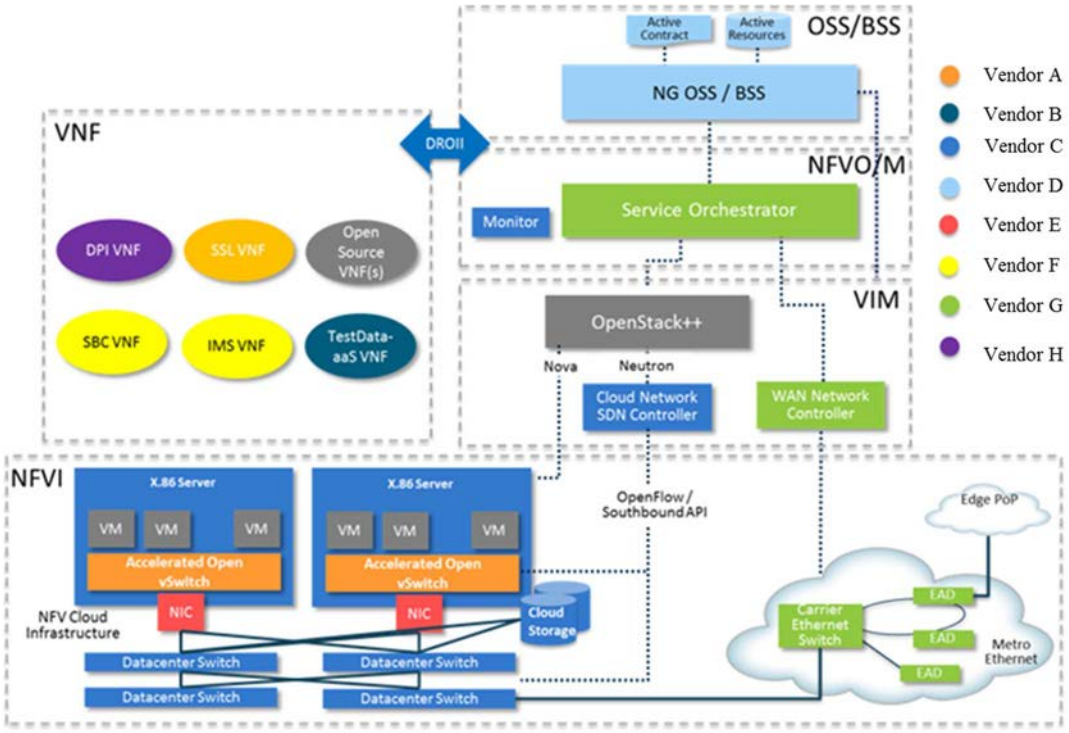



**Figure A.1: POC#1 - Open NFV Framework Project**

**Table A.1**

| | |
|---|---|
| Describe Application | This POC provides a high-level abstraction over both NFV and SDN for unified Management and Operations. It presents a Northbound API to OSS/BSS systems that exposes a Catalog of network capabilities and hides MANO complexity, providing the necessary abstractions for management to express its intent over infrastructure by policy without considering implementation details. It supports feedback loops for fully dynamic services, which might leverage active monitoring to drive re-evaluation of policies (real-time and batch analytics) for network operations to scale, heal, move and terminate functions to meet SLAs. It supports an end-to-end solution for a multi-vendor, multi-technology, multi-protocol and multi-PoP environments with dynamically composed service chains that are "network aware". System demonstrations extend from core to cRAN, where microcell LTE networks are instantiated using the same methods as core functions such as IMS. A single instance might support VM deployments via OpenStack or other mechanism, as well as Docker containers. |

| | |
|---|---|
| | The MANO middleware platform for this POC features an information-model based fabric for declaring connections and composing services, as well as a run-time with lifecycle management. It might perform the functions of NFV Orchestrator and VNF Manager for a unified architecture, but also serves as an Orchestrator of Orchestrators / Federated Controller.<br>It supports a catalog of VNFs provided by eco-system partners including: IMS, DPI, EPC and eNodeB.<br>This POC is deployed in a lab, with hardware and integration services, as well as switches, acceleration and OpenStack orchestration. |
| Role of SDN | DPI and cRAN aspects of the system require SDN level policies to be in place. In particular, the DPI VNFs requires that core functions such as IMS have their packets routed through and/or copied to its ports so that it might analyse the traffic. Likewise, with the cRAN, flow level policies need to be in place to ensure packets from connected devices are chained properly through the DPI components. |
| Role of NFV Orchestrator | The NFV orchestrator is used to translate declarative policies into dynamic processes, in this POC this involves multi-variant constraint satisfaction to calculate deployment manifests and then the correct steps for realizing a service. Part of this process involves evaluating the requirements for each component, so for instance, when a VNF for DPI is used the NFV orchestrator sees the requirement that packets have to be mirrored to one of its ports, it then scans the connected Infrastructure (or VIMs) to see what capabilities are available, and in this case settles on both the use of the OpenStack Open vSwitch plugin for instantiation, and then direction use of Open vSwitch through OpenFlow for subsequent configuration. From there the NFV Orchestrator is charged with generating an interface to each components, and determining the order and payloads of messages sent across each interface. |
| Network Controller details | Given OpenStack is the platform used for the ETSI NFV VIM, Open vSwitch plug-in is used for basic establishment of routers/networks, then the NFV orchestrator interfaces directly with Open vSwitch for the established networks. |
| Network controller to NFV Orchestrator interface and function | As detailed above, the NFV orchestrator interfaces both with the Openstack Neutron REST API and directly with the Open vSwitch NC using its REST-based OpenFlow API. |
| Data Forwarding Plane | As provided by the Open vSwitch. |
| Data Control Plane (Programming plane) | Our focus is on the determination and generation of proper SDN level policies based on the overall service requirement. Here the control plane issues FlowMod commands to affect the changes in packets in the Data Forwarding Plane, for instance to ensure chaining of IMS packets to DPI. |

| | |
|---|---|
| Lessons learned and (new) requirements | The overall goals of our POC are focused on high-level orchestration of VNFs, but it became evident early on that the SDN level would need to be included even for simple applications. The out of the box OpenStack does not have the capabilities necessary for advanced use cases. To overcome this, Neutron Plug-ins might be used to add SDN capabilities to the OpenStack, however the two paths this offers each have limitations. People would either create your own Plug-in, at best creating a customized software extension with limited reuse, or they might re-use one of the many existing Plug-ins (Open vSwitch, OpenDaylight, ML2, etc.). In any case, concepts of SDN will be a requirement of any orchestrator, however writing them to any specific plugin will limit their applicability, and it is unclear version to version which plug-ins will survive in their current form. The API evolution in OpenStack itself is problematic; the flow through to the semantics involved with using the plugins compounds this. For this POC, a more robust solution was to utilize a plugin such as the Open vSwitch minimally to simply instantiate the underlying structure, then to communicate with it directly using the underlying standard, establishing a pattern that is decoupled from the APIs of any specific plugin and portable to other NFV VIM. This represented a natural extension to our service model to include base concepts of flow-based requirements to our common set of abstractions. Given our declarative approach, this meant extending our information model with a few additional SDN/OpenFlow-like commands/constraints. Our generalized approach to synthesizing interfaces allowed to generate an OpenFlow interface using the same techniques as are used for communicating with OpenStack, further demonstrating the value in a holistic system for orchestration of both NFV and SDN layers involved in cloud solutions. |
| Gaps Identified | As this POC has demonstrated, it wraps and extends emerging protocols and standards to provide early worked examples that in turn accelerate protocol/standard evolution and adoption, in a positive self-reinforcing loop. It allows minimum viable protocols/standards to be promoted early and iterated quickly in a live testbed/system.<br>The problem is that OpenStack is low-level tooling with limited abstractions.<br>The gap is that ETSI NFV does not describe a MANO application architecture, without which implementations, as has been evidenced by many NFV vendor demonstrations and Cloud deployments generally, devolve into a sea of static scripts.<br>The implementation gap is a historic challenge in SDO community.<br>This POC provides a generalized platform (MANO middleware), a high-level abstraction over NFV and SDN, which provides flexibility in a diverse and evolving environment. It provides a bridge for implementing ETSI NFV specifications along with other related protocols and standards to achieve desired industry goals. |

## A.2 POC#2: Service Chaining for NW function selection in Carrier Networks

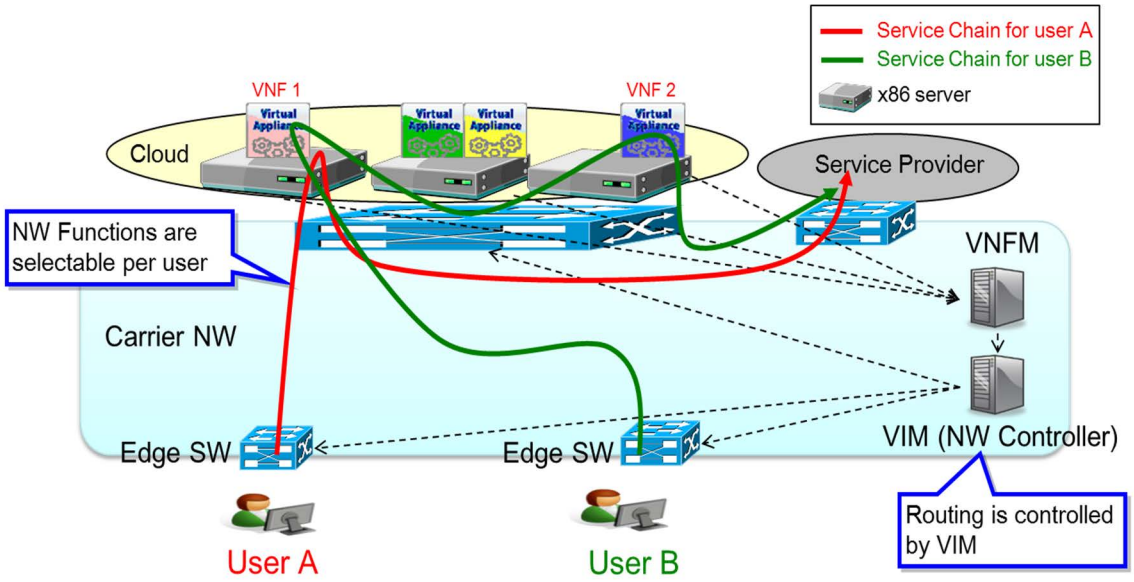



**Figure A.2: POC #2 Service Chaining for NW function selection in Carrier Networks**

**Table A.2**

| | |
|---|---|
| Describe Application | This PoC demonstrates the concept of Service Function Chaining (SFC). This PoC verify a new SFC method which works on carrier networks with Virtualised NW functions (VNFs). VNFs are provided by several vendors and some subjects will be clarified to consider in terms of interoperability. |
| Role of SDN | SDN is used for classification of user traffic, adding labels to packet and packet forwarding with tunnelling protocol. Our developed method adds labels to each packet to achieve SFC. In this method, an SDN enabled edge switch classifies packets and adds labels that show the Service Function (SF) the packet to be forwarded. |
| Role of NFV Orchestrator | The NFV Orchestrator has roles to manage user's service and VNFs, create service chain of each user. In this POC, an original NFV orchestrator was created and customized for this application. |
| Network Controller details | In this PoC In this POC, the SDN Controller (Ryu) is used to manage SDN enabled switch. Ryu [i.16] |
| Network Controller to NFV Orchestrator interface and function | In this POC, REST APIs are used between NC and NFV orchestrator, although the REST message description is our original. |
| Network Controller to Data Plane Control interface and function (Data Forwarding Plane) | In this POC environment, MPLS label is tentatively used as an SFC (service Function Chain) label and OpenFlow vswitch as a forwarding function in edge switch in order to implement the SFC method and evaluate its efficiency. |
| Data Control Plane (Programming plane) | The NFV Orchestrator exposes to users the interfaces necessary to contract service functions. The contracted services are translated into OpenFlow flow entries by the NFV Orchestrator. |
| Lessons learned and (new) requirements | The network functions working as routers require virtual interfaces to belong to various network segments. In SFC, VNFs might be connected to/from various VNFs; thus, designing IP addresses will be difficult. an OpenFlow controller has been temporally customized to reply to the dummy ARP packet to make neighbour VNFs work as default gateways. |
| Gaps Identified | The important point of our PoC is that new method is suggested to create a VNFFG and service function chain inside the NFVI. Therefore, intelligent load balancing is not considered. When performing load balancing of SFC in realistic environment, the service chain will be controlled dynamically depending on the resources of the VM or demand of VNFs. In that case, service chains are reconstructed in real time. In order to control the network in this way, taking advantage of the SDN is important. Therefore, the control mechanism in conjunction with load balancing of VNF is required. |

# A.3 POC#8: Automated Network Orchestration

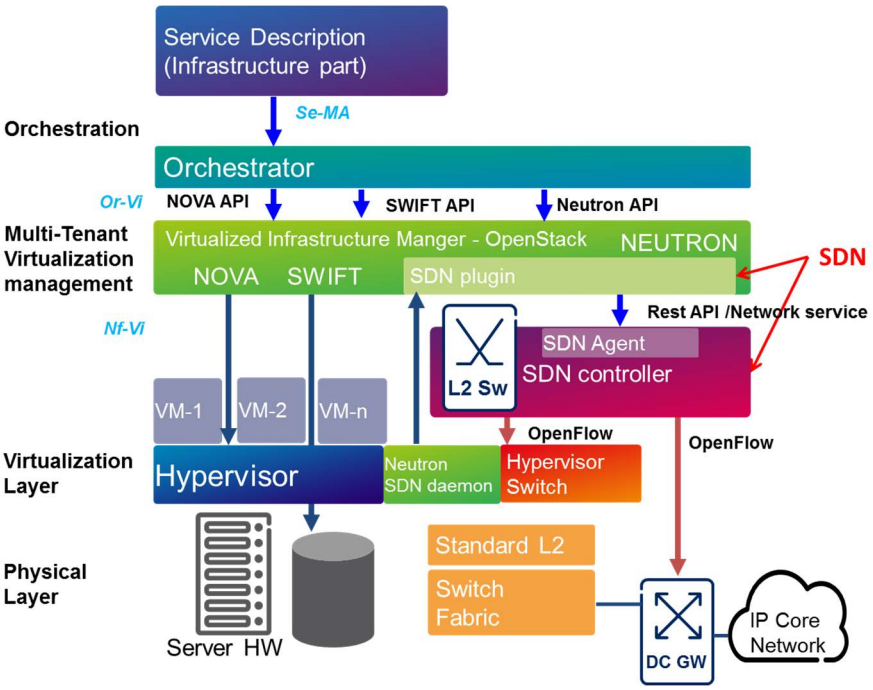



**Figure A.3: POC#8 Automated Network Orchestration**

**Table A.3**

| | |
|---|---|
| Describe Application | This PoC demonstrates the use of an open source orchestration system to automate the instantiation and scaling of a typical telecom messaging application as VNF, consisting of multiple VNFC. The application is not NFV-adapted and requires internal layer-2-connectivity for non-IP traffic and pre-programmed MAC addresses on the interfaces. |
| Role of SDN | SDN technology is used to implement the MAC forwarding required to connect the clustered VNFC components within the VNF and to connect the VNF to the network. The northbound abstraction of a L2 network is provided by an SDN agent function in the SDN controller and consumed by the SDN plugin component in Neutron. The required forwarding in the virtual switches is implemented using OpenFlow as a south bond protocol. |
| Role of NFV Orchestrator | |
| Network Controller details | The controller used in this PoC is an own development written in Erlang. At the time this PoC was planned, there were no sufficient open source alternatives available. In the meantime functional equivalents based on OpenDaylight are used in similar PoC work. |
| Network Controller to Orchestrator interface and function | The SDN agent component in the controller provides a Rest API to the SDN plugin in Neutron, which in turn provides a Rest API to the NFV orchestrator (the un-modified Openstack Networking API). |
| Data Forwarding Plane | In the PoC environment, forwarding has been provided using OpenFlow-1.3 capable virtual switches. The PoC used the CPqD reference implementation of a software switch (http://cpqd.github.io/ofsoftswitch13/), since switching performance was not the primary focus of this PoC. |
| Data Control Plane (Programming plane) | All needed network services, such as MAC learning, broadcast, flooding, or multicasting, are implemented and controlled via the OpenFlow interface. No native L2 switching function is used in the virtual switches. An SDN daemon in the host connects newly created virtual NICs to the OpenFlow switch and maps their ID's to the Openflow ports. |
| Lessons learned and (new) requirements | The OpenFlow implementation provides an extremely flexible control over the implemented services based on the OpenFlow 1.3 forwarding abstractions. While this could be seen as overly complex for plain L2 services, it allows flexible integration of distributed routing, load balancing, or forwarding graphs without changes in the control architecture or vSwitch implementation. |
| Gaps Identified | The approach shown requires a highly performant Openflow-capable dataplane in the vSwitch, which is not available in open source products today. The requirement is line-rate for multi-table Openflow pipelines using one or maximum two cores per server.<br>In order to make use of the aforementioned flexibility of the southbound resource control interface, more work is required to provide standardized northbound abstractions of the required functionality for OpenStack, OpenDaylight, and the NFV orchestrator in order to assure exposure of networking capabilities all the way from OSS/BSS to the actual forwarding. |

# A.4 POC#13: Multi-Layered Traffic Steering for Gi-Lan

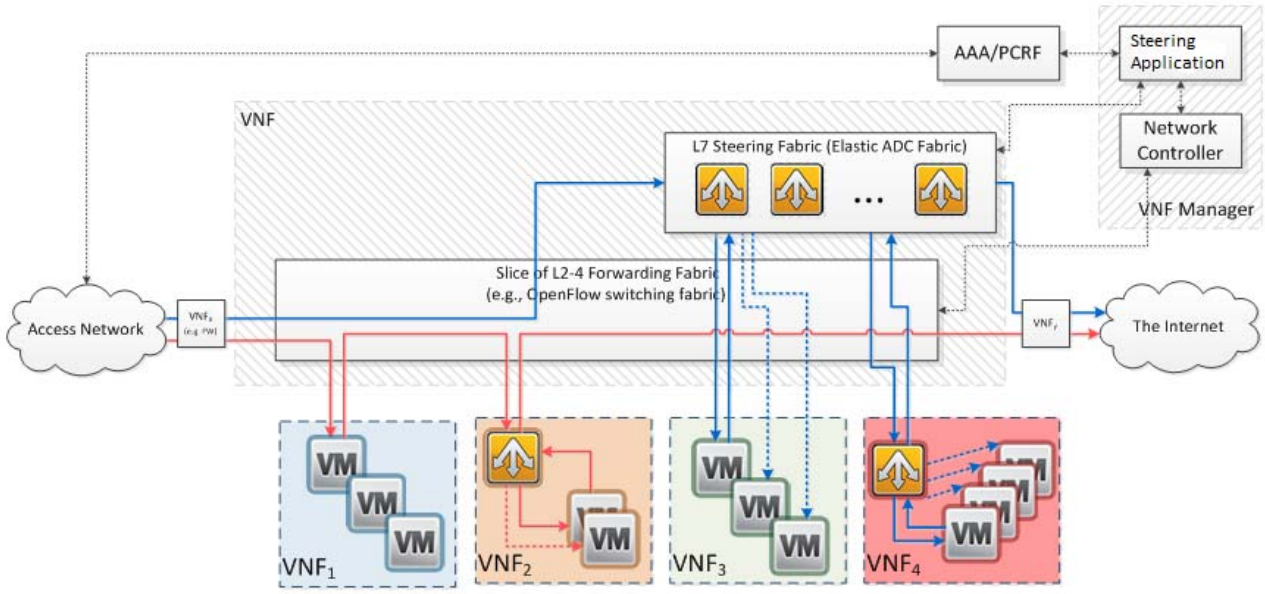



**Figure A.4: POC#13 Multi-Layered Traffic Steering for Gi-Lan**

**Table A.4**

| | |
|---|---|
| Describe Application | This POC demonstrates the concept of multi-layered traffic steering system, i.e. support the definition of steering rules at different layers of the networking stack and enforce those decision at the most appropriate (Virtualised) device hence optimizing the utilization of network components and, accordingly, reducing the need for high capacity high level network components. In addition, this PoC, demonstrates the Infrastructure and VNF-external Load Balancing models as described in ETSI GS NFV-SWA 001 [i.5]. |
| Role of SDN | SDN serves two roles: 1. An SDN enabled forwarding fabric is used for L3-L4 traffic steering; and 2. The SDN enabled forwarding fabric is also used to support elasticity of the L7 traffic steering fabric, i.e. traffic that might require L7 analysis is distributed, by the SDN enabled fabric, among the different instances of the L7 traffic steering virtual appliance. The traffic steering control application sits on top of an SDN controller and programs thru it both the L3-L4 traffic steering flows as well as the L7 traffic distribution flows. |
| Role of NFV Orchestrator | The NFV Orchestrator is responsible for the provisioning of all the components. The goal is to integrate the steering functionality with the NFV Orchestrator , but initially the PoC will be based on vanilla OpenStack |
| Network Controller details | In this POC, a vendor SDN controller is used as a virtual appliance (a VNFC). See gaps |
| Network controller to NFV Orchestrator interface and function | Not applicable (see gaps) |
| Network Controller to Data Plane Control interface and function<br>(Data Forwarding Plane) | In this POC, OVS is used as the L3-L4 steering component as well as the load balancer for the L7 steering elastic fabric. In a more realistic environment, the recommendation is to provide these functionalities with physical switches both for performance and for flexibility in the deployment of the different VNFCs that comprise the network service (see gaps). The interface between NC and the OVS is OpenFlow (1.3 or higher) |
| Data Control Plane (Programming plane) | The Steering Control application exposes to users the interfaces necessary to define service chains and steering rules. These rules are translated into OF flow entries and/or ADC steering rules |
| Lessons learned and (new) requirements | On-going - TBD |
| Gaps Identified | In its current incarnation all the components in the PoC, including the NC, the OVS and the value added services (VAS) VMs, are packed together as a single VNF. In more realistic environment the VASs are not likely to be part of the steering VNF and might be distributed over the entire NFVI, hence a single OVS is not likely to be the proper solution. Instead, the steering VNF needs to be able to create and manage the steering flows on the physical network or at least in a partition of the physical network. Whether this is done directly by the controller embedded in the VNF or by an infrastructure controller in the NFVI the interfaces from VNF to infrastructure switch and/or controller are not defined. Moreover, then mechanisms (partitioning, prioritization, etc.) to enable VNFs to influence traffic flows in the physical infrastructure are not defined. |

# A.5 POC#14: Forces applicability for NFV and integrated SDN

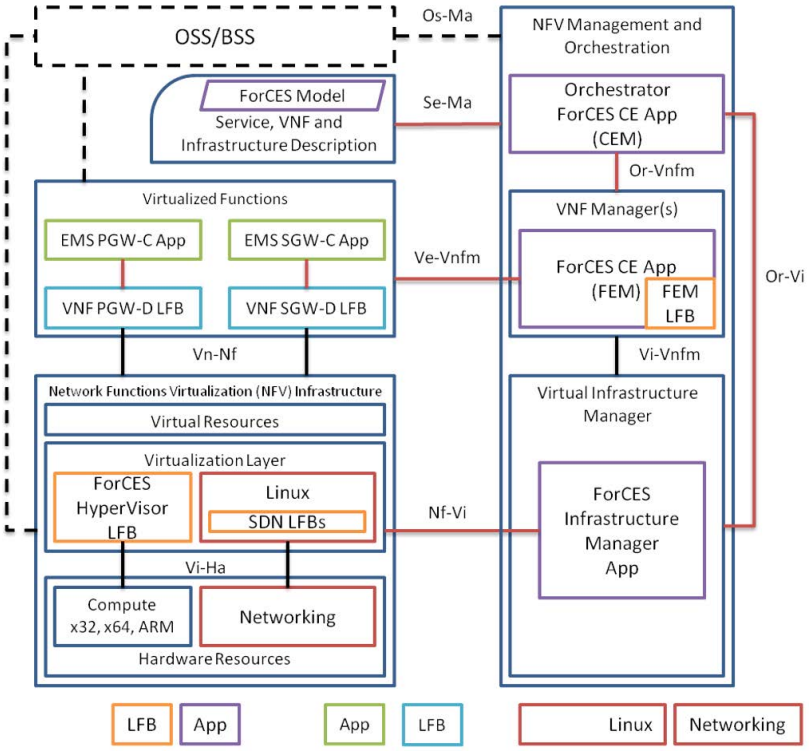



**Figure A.5: POC#14 Forces applicability for NFV and integrated SDN**

**Table A.5**

| | |
|---|---|
| Describe Application | Demonstrate the applicability of the ForCES framework, both in terms of modelling and interfacing. Show how the ForCES modelling language might be used to model the infrastructure as well as the network functions. Utilizing only the ForCES protocol on modelled entities, to demonstrate control of both the infrastructure and the NFs. Provide a view on how to use a single framework (model and protocol) for the whole NFV architecture. |
| Role of SDN | SDN is applied in two specific cases in this PoC. The first is to control and manage the networking resources of the infrastructure, connect containers and VMs. The second is the VNF itself, which is modelled and abstracted using the ForCES model and is controlled by an EM application. |
| Role of NFV Orchestrator | The NFV orchestrator is an application on top of the VIM controller requesting to set up the VMs and connect them together. The NFV orchestrator in this PoC is a combined SDN & NFV orchestrator |
| Network Controller details | There are actually two different groups of controllers and applications using the controller. The first group is the EM CEs and applications controlling the VNFs. The second group is the VIM which is acting as the controller for the networking between VMs. It is important to highlight that the controller is in fact both and SDN & NFV controller, as the Hypervisor is also being modelled using the ForCES model, and as such the controller is both an SDN & NFV controller. |
| Network Controller to NFV Orchestrator interface and function | The NFV orchestrator being an application on top of the controller is using ForCES specific APIs to request, provision and control the network resources. |
| Data Forwarding Plane | All data plane forwarding is performed by the VNFs. |
| Data Control Plane (Programming plane) | Both, the EM control applications program the VNFs and the VIM program the network infrastructure through the ForCES protocol interface. |
| Lessons learned and (new) requirements | The ForCES framework is applicable to both SDN and NFV. |
| Gaps Identified | The interface between EM and NFV has not been specified and might lead to vendor lock-ins for complete Virtualised solutions. Opening up such an interface using SDN techniques will allow more stakeholders to develop solutions and thus provide more variety and flexibility for operators to select the best possible solutions. |

# A.6 POC#15: Subscriber Aware Sgi/Gi-lan Virtualisation

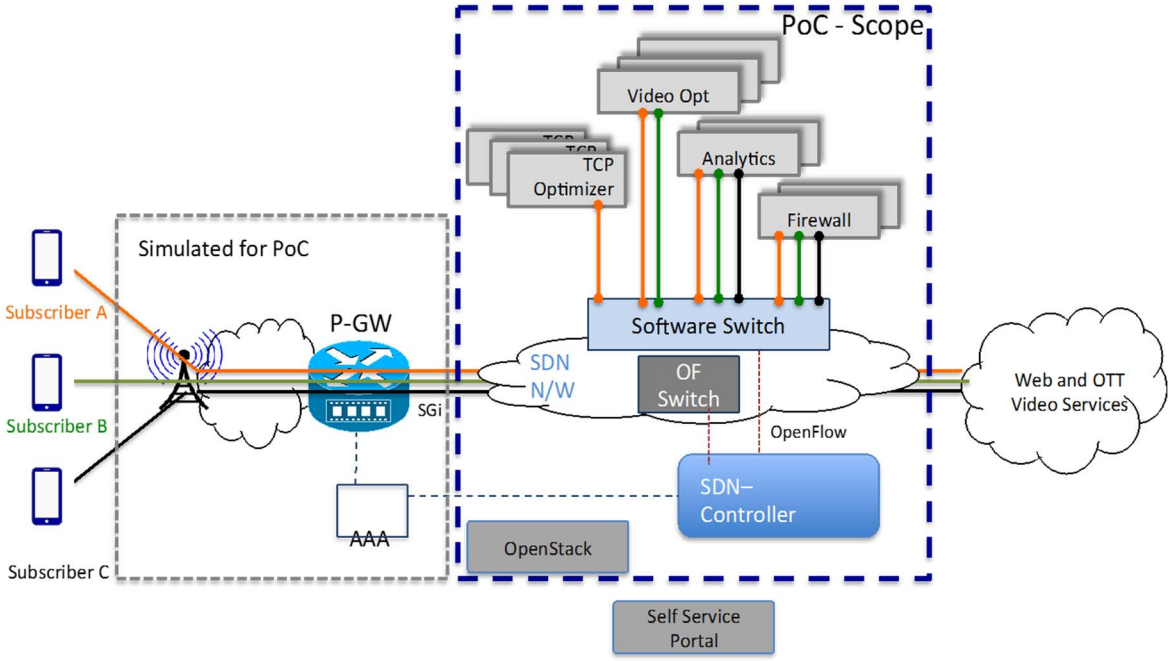



**Figure A.6: POC#15 Subscriber Aware Sgi/Gi-lan Virtualisation**

**Table A.6**

| | |
|---|---|
| Describe Application | This POC demonstrates:<br>1. A subscriber aware method for service chaining intrinsically provided by the NFVI.<br>2. How a subscriber aware service chaining method applied in the NFVI might enable Virtualisation of functions on the Sgi/Gi interface and provide elasticity to VNFs.<br>3. How a subscriber's specific service function element chain is selected and is composed from a catalog of individual functions, where these functions might be hosted within a NFVI-PoP or across NFVI-PoPs.<br>4. How Virtualisation might enable programmability of network driven functions that are selected by subscribers on a self-service portal. |
| Role of SDN | SDN provides the fabric through which the traffic from/to the P-GW to/from PDN passes. It creates steering so that traffic might be delivered to/from VNFs. While VNFs interconnect on "an Ethernet interface" to the fabric, the next hop VNF is chained on a per subscriber basis.<br>The PoC uses an SDN controller to dynamically configure the Ethernet switches using OpenFlow. In the PoC subscribers are added to the network. The SDN controller configures the end-to-end service chaining path for each new subscriber. |
| Role of NFV Orchestrator | The VIM is OpenStack. It creates/manages the virtual machines associated with VNFs.<br>The networking component of OpenStack (Neutron) assigns IP/Mac addresses, but is not responsible for setting up interconnectivity for the VM, beyond configuration of vNIC.<br>The NFV orchestrator is not yet used in this POC. |
| Network Controller details | Distributed controller that is federated using a distributed mapping service (IETF LISP framework and ODL based). Controller sets up connectivity from vNIC, vSwitch, to TOR and Subscriber aware switch. Interconnectivity between TOR is via. Overlay that in PoC is beyond scope of NC.<br>The Network Controller used is an SDN Controller. It is based on the OpenDayLight controller. |
| Network Controller to NFV Orchestrator interface and function | The SDN network controller interfaces with Openstack Neutron to keep track of VNFs. This knowledge is global and distributed.<br>The integration of the Network Controller to the NFV orchestrator is planned for the 2nd stage of the PoC. |
| Data Forwarding Plane | Open Flow<br>The data plane uses OVS switches. The switches are used to establish per subscriber service chains.<br>The switches are configured using OpenFlow. |
| Data Control Plane (Programming plane) | Interface with policy engine so that per subscriber flows might be setup when subscriber connects. This helps make the network programmable on a per subscriber basis. |
| Lessons learned and (new) requirements | |
| Gaps Identified | The PoC identified a gap where the SDN controller, which is a VNF, does not have a way to configure the infrastructure, according to the current NFV reference architecture.<br>Another gap identified is the lack of a standard North Bound API to the SDN controller. There was not a standard way to connect the SDN controller to the orchestrator. |

# A.7 POC#16: NFVIaaS with Secure SDN-controlled WAN Gateway

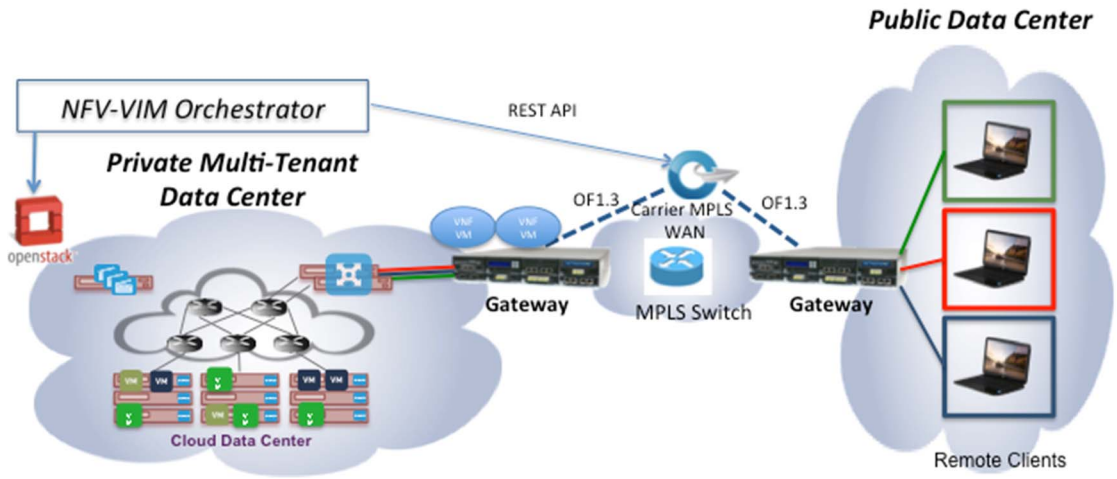



**Figure A.7: POC#16 NFVIaaS with Secure SDN-controlled WAN Gateway**

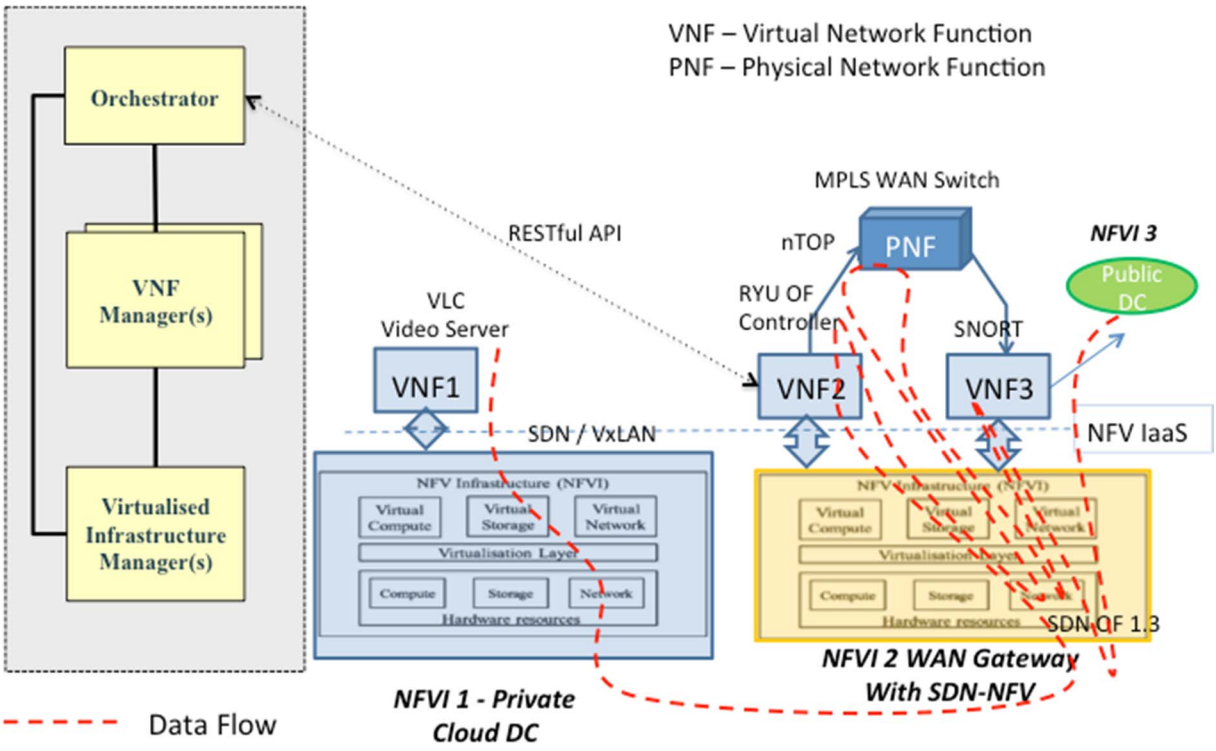



**Figure A.8: POC#16 Mapping to NFV end-to-end architecture framework**

**Table A.7**

| | |
|---|---|
| Describe Application | This POC evolves around using a cloud orchestrator, with OpenStack integration, to provision a Virtualised network topology (VMs & VNFs) while monitoring and enforcing SLAs via SDN-OpenFlow, including across legacy MPLS WAN.<br><br>This PoC demonstrated the following:<br><br>1. The orchestration of a software-defined data centre, or NFVI<br>2. Deploy an enterprise service with SLAs (private –loud - NFVI) enforced by monitoring real-time analytics through the Open Source nTOP application<br>3. Provide VNF services such as firewall, IPS/IDS and load balancer to the enterprise, or to remote users across the WAN<br>4. Ensure seamless connection over the WAN by having an accelerated COTS server-based Gateway, hosting IDS/IPS, firewall and load balancing VNFs<br>5. Demonstrate SLA across the WAN using orchestrator capability to control the Open Source SDN-OpenFlow controller (RYU) and the OpenFlow-capable Gateway while monitoring WAN bandwidth using traffic monitoring application (e.g. nTOP) and OpenFlow protocol statistics<br>6. Demonstrate interoperability with a legacy MPLS WAN switch-router<br><br>In addition, the PoC demonstrated the co-existence of physical and virtual functions (PNFs and VNFs) within a data centre, or between data centres, across a legacy MPLS WAN. |
| Role of SDN | This PoC uses an open source SDN controller (RYU) controlling a SDN gateway. The gateway interfaces a multi-tenant data centre to a legacy MPLS WAN. The PoC uses a full implementation of OpenFlow 1.3 with multi-table and metering capabilities – all implemented on the SDN gateway. It is important to note that this Gateway combines the functions of SDN controller, WAN controller, and SDN gateway, and that of NFVI, all in one physical, Virtualised platform. As a result, The gateway is capable of hosting multiple VNFs. In fact the SDN controller is one such VNF. |
| Role of NFV Orchestrator | The NFV Orchestrator, acting as both NFV and VIM orchestrator, is used to instantiate the software-defined data centre (SDDC), based on requirements in a user-defined service template. In addition to controlling the NFVI entities, it also interacts with the SDN WAN gateway to control the SLA over the WAN. |
| Network Controller details | The network controller is the open source RYU SDN controller. This is implemented as a VNF, hosted by the gateway. |
| Network Controller to NFV Orchestrator interface and function | The Network, or SDN, controller interfaces with the NFV/VIM orchestrator through a RESTful type API. The gateway gathers flow statistics. The orchestrator polls the gateway to obtain the traffic statistics. An application hosted by the orchestrator computes the bandwidth (BW) utilization, based on the gathered statistics. Based on the calculated BW, the orchestrator sets the OpenFlow metering parameters on the gateway, and as a result controls the amount of BW allocated to the video stream application, transported over the WAN. |
| Data Forwarding Plane | Data plane forwarding is implemented as a VNF hosted by NFVI compute nodes, or by the gateway. The forwarding plane might be based on physical or virtual network functions (PNFs and VNFs), including security, management and monitoring functions. |
| Data Control Plane (Programming plane) | In this PoC, OpenFlow was used to control the data plane network elements for traffic forwarding, firewall, load balancing and other applications. |
| Lessons learned and (new) requirements | This PoC controls SLA over the WAN through the interaction between the NFV/VIM orchestrator and the SDN controller. A need was identified to control the SLA within the DC, in addition to the WAN. |

| | |
|---|---|
| Gaps Identified | Although this might be a common knowledge, there is a stronger sense to continue to highlight that SDN might control both physical and virtual functions in the various GS documents. In addition, there is a need for tight integration between the orchestrator and the SDN controller. |

## A.8 POC#21: network intensive and compute intensive hardware acceleration

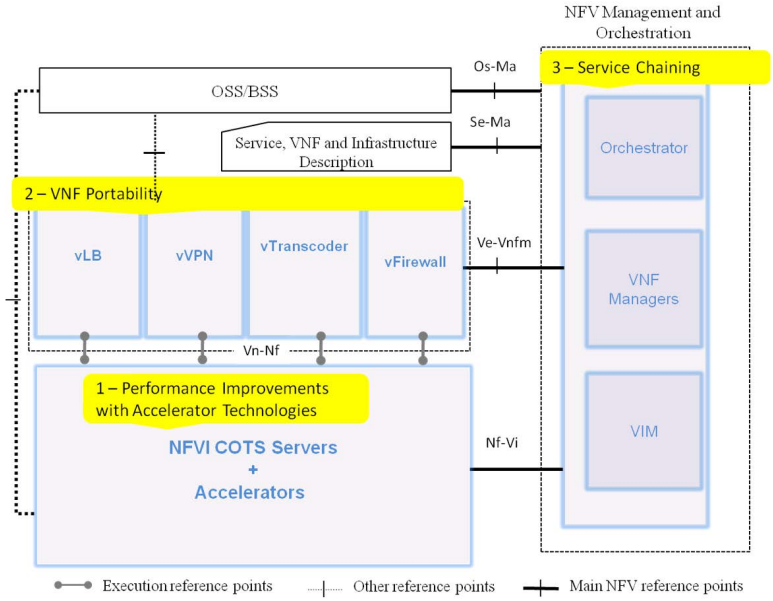
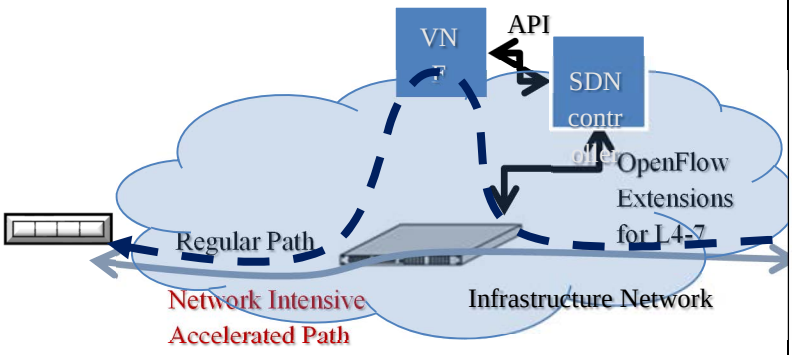



**Figure A.9: POC#21 network intensive and compute intensive hardware acceleration**

**Table A.8**

| | |
|---|---|
| Describe Application | Demonstrate the benefits or Hardware Acceleration in NFV environments for compute and network intensive functions. The first stage shows how rerouting of flows from the VNF to the Infrastructure Network might improve performance. A Layer 7 Load balancer VNF is used as an example VNF for this PoC. |
| Role of SDN | SDN is used to program the physical network switches and NIC cards to process and reroute traffic on behalf of the VNF based on the Dynamic Rerouting of Packet Flow Routing as described in GS NFV-INF 005 [i.1].   |
| Role of NFV Orchestrator | Not applicable - the first stage of the PoC focused on the infrastructure only and relied on manual provisioning. |
| Network Controller details | The controller is a Protocol Oblivious Forwarding (POF) controller (http://www.poforwarding.org/) that is based on Project Floodlight® (see note) and enhances OpenFlow to be oblivious to the dataplane forwarding protocol. |
| Network Controller to NFV Orchestrator interface and function | Not applicable - the first stage of the PoC focused on the infrastructure only and relied on manual provisioning. No NFV Orchestrator used. |
| Data Forwarding Plane | Initially, all data plane forwarding is performed by the VNF that performs the load balancing decision. Once it has made its load balancing decision, the VNF requests some rules be added to the infrastructure network (physical switch, NIC). It makes those requests through an Optimization Interface that converts those requests into NBI calls to the network controller. |
| Data Control Plane (Programming plane) | The VNF programs the SDN switch/NIC through an optimization interface. |
| Lessons learned and (new) requirements | Scalability of the SDN controller and SDN agent/switch need to be taken into consideration. |
| Gaps Identified | • Need adoption of VNF API to SDN controller for HWA through standards or open source.<br>• OpenFlow extensions are needed to support dynamic rerouting based on L4-L7, preferably using protocol agnostic extensions to provide the necessary flexibility and extendibility. |
| NOTE: "Floodlight is a registered trademark of Big Switch Networks, Inc. This information is given for the convenience of users of the present document and does not constitute an endorsement by ETSI of the product named. Equivalent products may be used if they can be shown to lead to the same results". | |

## A.9 POC#23: E2E orchestration of Virtualised LTE Core-Network functions

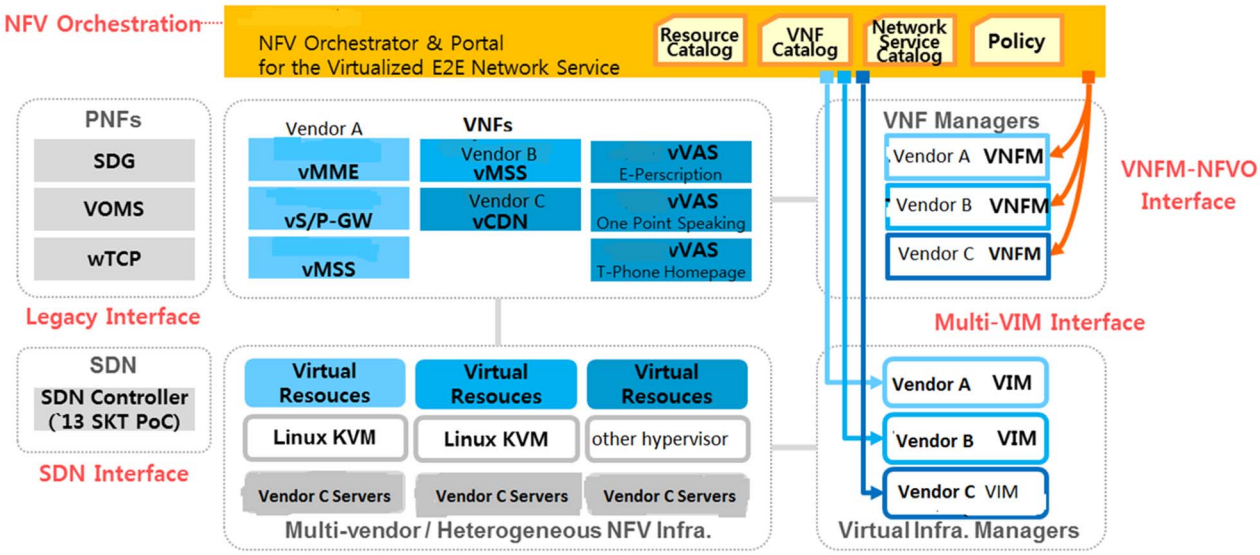



**Figure A.10: POC#23 E2E orchestration of Virtualised LTE Core-Network functions**

**Table A.9**

| | |
|---|---|
| Describe Application | Our NFV orchestrator defines an interface between the NFV orchestrator and the SDN controller for dynamically chaining various in-line & value-add service functions (e.g. video optimization, content caching, etc.) in the SGi LAN in the LTE core network |
| Role of SDN | As briefly mentioned above, SDN is used within the SGi LAN for dynamic service chaining based on the policy.<br>The following describes an example use-case. Assume User-1 subscribed to a video optimization service, whereas User-2 did not. In such case, User-1's packets will be directed to the video optimization service node (for a better user experience), whereas User-2's packets will not be directed to the particular service node. |
| Role of NFV Orchestrator | In our PoC architecture, the NFV orchestrator has the global view of the Virtualised LTE network. Furthermore, all the dynamics on the network service topology are managed through the NFV orchestrator using VNF-FGs.<br>The topology dynamics information as defined in the ETSI NFV specification get passed to the SDN controller (via our internally defined interface between the orchestrator and the SDN controller) and used as part of dynamic service chaining of the user traffic. |
| Network Controller details | Currently in the PoC, a proprietary SDN controller is use. An earlier version of this controller was used in a demo during MWC 2013.. |
| Network Controller to NFV Orchestrator interface and function | There is an interface between the SDN controller and the NFV orchestrator to exchange the VNF-FG dynamics information. Whenever there is a VNF-FG change, the NFV orchestrator sends the updated VNF-FG information to the SDN controller. |
| Data Forwarding Plane | The dynamic service chaining feature is transparent to both the data packets and the in-line value-add network functions in the SGi LAN. The value-add functions simply process the incoming packets in-line as they receive them. |
| Data Control Plane (Programming plane) | Data forwarding plane consists of SDN switches. The SDN controller sends the data forwarding (i.e. OpenFlow) rules to the switches in the SGi LAN, and the switches act as data packet forwarders based on the rule received. |
| Lessons learned and (new) requirements | • The number and variety of in-line and value-add functions in the SGi LAN do not grow as fast as fast as initially anticipated<br>• On the other hand, if the number of functions happens to increase dramatically, there might be scalaibility issues, especially in OpenFlow (e.g. how many rules might be stored, how fast has the rule-updates to be done, etc.). |
| Gaps Identified | • Since the NFV orchestrator sends very primitive information (i.e. VNF-FG) to the SDN controller, most complexity will be residing in the implementation of the SDN controller and the applications sitting on top of the SDN controller.<br>• There might also be scalability issues in the number of OpenFlow rules to be stored as briefly mentioned above.<br>However, the above concerns do not belong to the scope of the NFV orchestrator. At least from the scope of our PoC, no major gaps were identified. |

# A.10 POC#26: Virtual EPC with SDN functions in Mobile Backhaul Networks

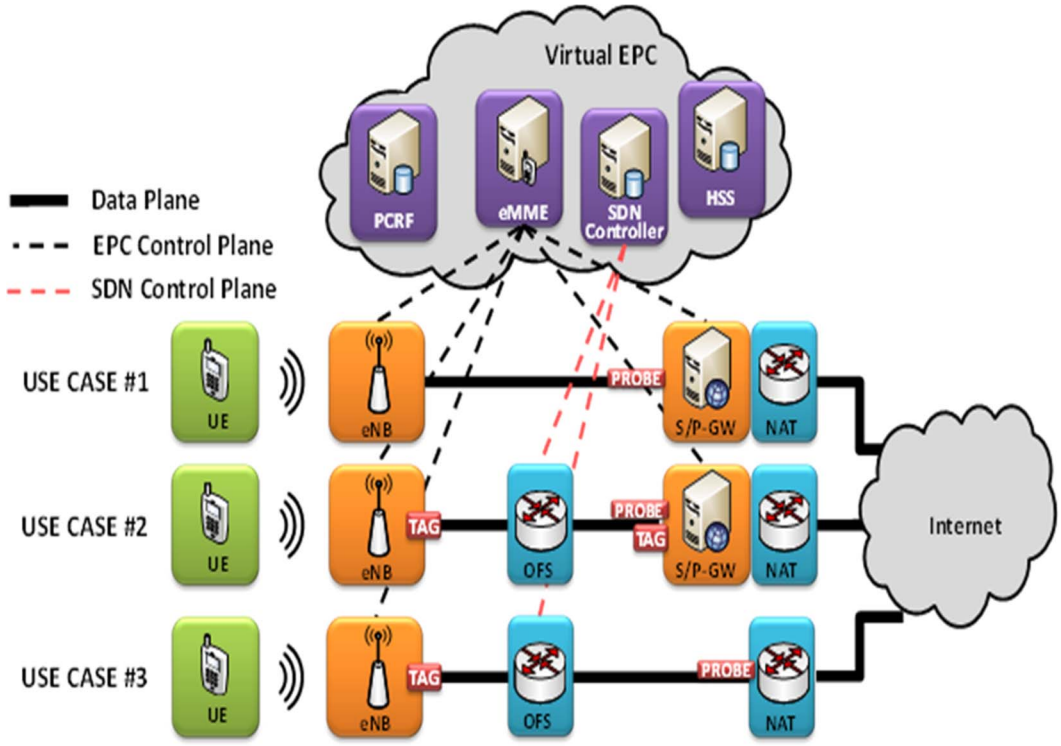



**Figure A.11: POC#26 Virtual EPC with SDN functions in Mobile Backhaul Networks**

**Table A.10**

| | |
|---|---|
| Describe Application | This PoC will verify the entire Virtualisation of EPC (vEPC) and the benefits that Network Function Virtualisation (NFV) and Software Defined Network (SDN) bring to EPC. The Orchestrator interacts with virtual EPC network elements such as the Mobility Management Entity (MME) to receive mobility information and trigger the action in the SDN network controller to change the required data plane flows to perform mobility while maintaining the QoS assigned to the mobile devices across network events. |
| Role of SDN | In this PoC, several use cases of vEPC are demonstrated where progressively the scenario moves from the usage of VNFs of existing EPC network elements (scenario 1) to the integration of SDN as part of the transport network used in the mobile backhaul (scenarios 2 and 3). SDN is used to replace mobile network stack such as GTP replaced by off the shelf switches e.g. Ethernet, VLAN or MPLS. |
| Role of Orchestrator | The orchestrator relies on the information received by the vEPC to configure the edge nodes of the SDN network to forward the specific user data accordingly to the backhaul provisioned paths. |
| Network Controller details | The SDN network controller is compliant with OpenFlow 1.3. The PoC uses Ryu SDN network controller, which is available as open source and developed in Python® (see note). |
| NC to Orchestrator interface and function | The orchestrator application is implemented in Python, running directly on top of the SDN network controller. The MME uses a REST-like API based on JSON messages to interact with the SDN network controller to notify mobile events. |
| Data Forwarding Plane | The data forwarding planes are based on L2 switches with Ethernet, VLAN or MPLS forwarding capabilities managed from the SDN network controller. |

| | |
|---|---|
| Data Control Plane (Programming plane) | There are different control planes in use:<br>• OpenFlow 1.3: TCP-based Between SDN network controller and OpenFlow datapaths.<br>• 3GPP S1AP: SCTP-based between the eNB and the MME.<br>• 3GPP S11: UDP-based between the MME and the S/P-GW. |
| Lessons learned and (new) requirements | The choice of Ryu for the SDN network controller application seems to be appropriate as it allows fast development of Python applications for realizing the orchestration. Network provisioning is performed proactively, upon initialization of the setup, which does not introduce any operational delays from the controller's perspective. The controller reacts to mobile specific operations, i.e. attach, detach, handover setting up the appropriate flows proactively. It also receive network events such as link down, link up, congestion and acts to correct and adapt the network to the new conditions to obtain the optimal performance and user experience.<br>The provisioning of the backhaul, either VLAN-based or MPLS-based seems suitable for our technology. In more extreme cases where no dedicated Layer 2 paths are available, it would be possible to fall back to standard layer 3 IP GRE tunnelling between endpoints, however it is not preferred. Furthermore, it was demonstrated that mobile events require only changes at the edge of the network, signalled from our SDN controller, and would not affect by any means the existing backhaul provisioning. These 2 functions might ran independently from each other the moment there is an agreement on the communicating endpoints based on specific packet fields (VLAN tags, MPLS labels or IP addresses).<br>The 3GPP signalling mechanisms for handovers were executed successfully, albeit further testing is required to make a more comprehensive analysis of the delays in play and devise possible optimizations.<br>The PoC shows the feasibility of adapting the standard 3GPP GTP tunnelling to different use cases. Encapsulating the GTP in an MPLS tunnel, or completely removing the GTP tunnelling, encapsulating the user data in an MPLS tunnel, which then provides the scope previously offered by GTP.<br>In addition, one of the lesson learnt is that SDN capable switch implementations differ from one another, resulting in unexpected behaviours that have to be specifically dealt with. In our setup, this meant having to add middleboxes to perform MPLS Label Edge Router (LER) functions was not originally conceived and contributes to increase the delay, especially of the signalling events. Furthermore, the original network design had to be reworked replacing the single broadcast domain with routing techniques to connect the separated networks.<br>During the realization of the POC, a standalone policy and charging function was realized using OpenFlow. This function brings together MPLS encapsulation/decapsulation, Evolved Packet System (EPS) bearer metering as well as Service Data Flow (SDF) metering that enforces the Traffic Flow Templates (TFT) defined for a user. The charging might be performed by collecting the flow/meter statistics from the datapaths. |
| Gaps Identified | ETSI NFV has not defined an interface between NFV Orchestrator and SDN controller north bound interface in the form of a standard API. Therefore, it is recommend that ETSI NFV pays attention to:<br>1. Interoperability and certification of OpenFlow or SDN enabled switches to ensure the required functionality is available.<br>2. Abstraction layers between HW and Virtualised components. The SDN controller needs to access directly to information in packet headers to determine new locations of the packet forwarding during mobility events. Therefore, it is recommend that applications (e.g. vMME) having a direct access to the SDN controller instead of API to the VIM.<br>3. Having NFV APIs built to access only virtual instances of the network which do not require time critical actions. |

| | |
|---|---|
| | A test framework with a broad number of tests has to be defined to explore each possibility that the OpenFlow protocol supports. In this direction, there is already some work developed by Ryu SDN Controller, what they call the "Ryu Certification". However, additional effort is required to complete the uses cases that evaluates the datapaths implementation and their conformance with the standard OpenFlow. For instance, the switches do not behave as expected what comes to the internal treatment of the actions and matching functionality specified in the Openflow specifications. For instance, the MPLS packet processing differs in OpenvSwitch and OfSoftSwitch13 in terms of removing the MPLS label and reprocessing the packet in a different table.<br>Our PoC considered the SDN controller as an additional VNF in addition to other functions as MME and control plane S/P-GW. In the present testbed, there is no interface with the cloud orchestration, but such interfaces might simplify the connection of mobile specific VNFs as these might control the underlying SDN network. Special abstractions need to be defined for that purpose.<br>The EPC used in the PoC used single tenant but the multitenancy support of EPC infrastructure was considered during the network provisioning. |
| NOTE: | "Python is a registered trademark of the Python Software Foundation. This information is given for the convenience of users of the present document and does not constitute an endorsement by ETSI of the product named. Equivalent products may be used if they can be shown to lead to the same results". |

## A.11 POC#27: VoLTE Service based on vEPC and vIMS architecture

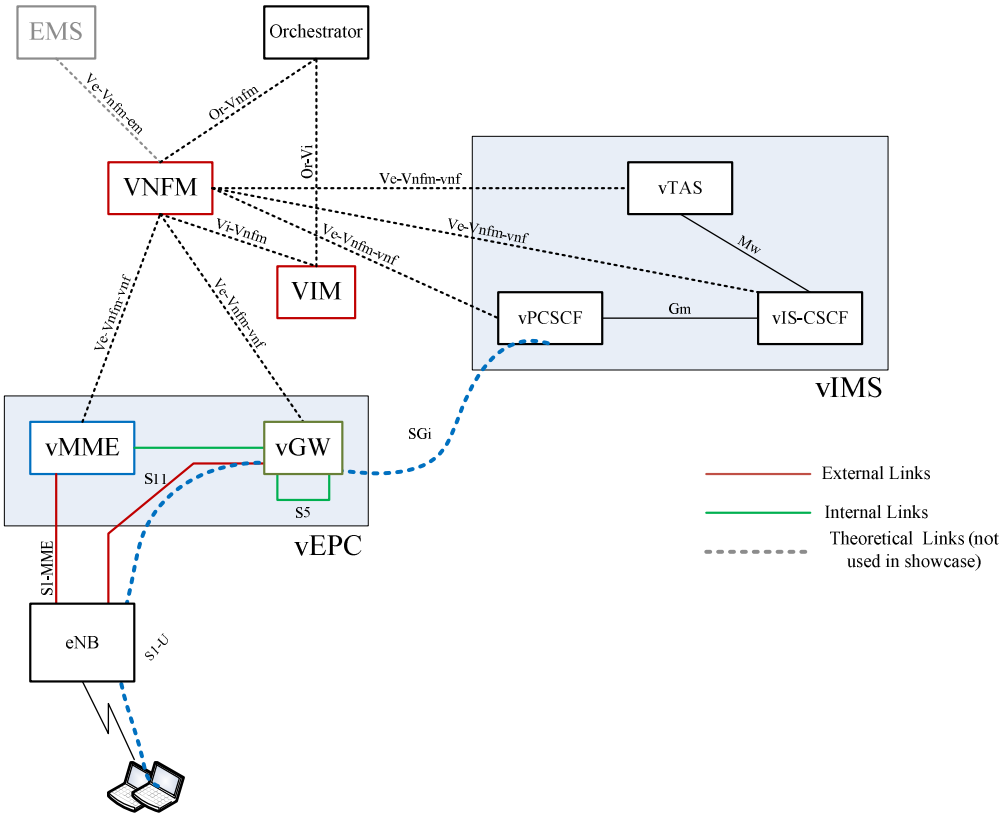



**Figure A.12: POC#27 VoLTE Service based on vEPC and vIMS architecture**

**Table A.11**

| | |
|---|---|
| Describe Application | The PoC demonstrates the end-to-end VoLTE service that might be constructed based on vIMS and vEPC VNFs running on a multi-vendor NFV environment. |
| Role of SDN | SDN technology is used to separate the control plane and user plane in vEPC xGW (SGW and PGW), that is, the control plane implementing SDN controller deployed as VNFs runs in Virtualised environment, and the user plane runs in SDN-based forwarding equipments.<br>On the other hand, SDN technology is introduced into NFVI to enhance the capability of network management in telecom cloud. |
| Role of NFV Orchestrator | The NFV Orchestrator has the role to manage the whole resources in multi-vendor NFV environment, the instantiation of the VoLTE service and all constituted VNFs, and their lifecycle management. |
| Network Controller details | There are two kinds of Network controllers applied in different layers in this PoC.<br>Type I, the Network controller is deployed as part of VNF and provides the interaction with user plane of the EPC GW using SDN-based protocols.<br>Type II, the Network controller is introduced into VIM to enhance the capability of network management in telecom cloud. |
| Network Controller to NFV Orchestrator interface and function | In this PoC, the Type I Network Controller has no direct interface with the NFV Orchestrator. The Type II Network Controller exposes interface through VIM to NFV Orchestrator, through which the network resource might be managed efficiently. |
| Data Forwarding Plane | In this PoC, the SDN-based data forwarding equipments might be Virtualised or non-Virtualised. The SDN-based data forwarding equipments interact with Type I Network Controller based on OF-epc protocol (see note), and interact with Type II Network Controller based on OpenFlow implementation. |
| Data Control Plane (Programming plane) | The SDN controller sends the data forwarding rules (i.e. defined in OpenFlow, or OF-epc) to the forwarding equipments, and the forwarding equipments act as data packet forwarders based on the rule received. |
| Lessons learned and (new) requirements | |
| Gaps Identified | With the separation of control plane and data plane in EPC GW, the internal private control-forwarding interfaces need be open and standardized, and supported by the SDN-based interface. This gap now is considered in ONF.<br>More gaps will be identified and provided with the progress of this PoC. |
| NOTE: OF-epc is extended by ZTE based on the OF-mpc interface that is in progress in ONF. | |

# A.12 POC#28: SDN Controlled VNF Forwarding graph

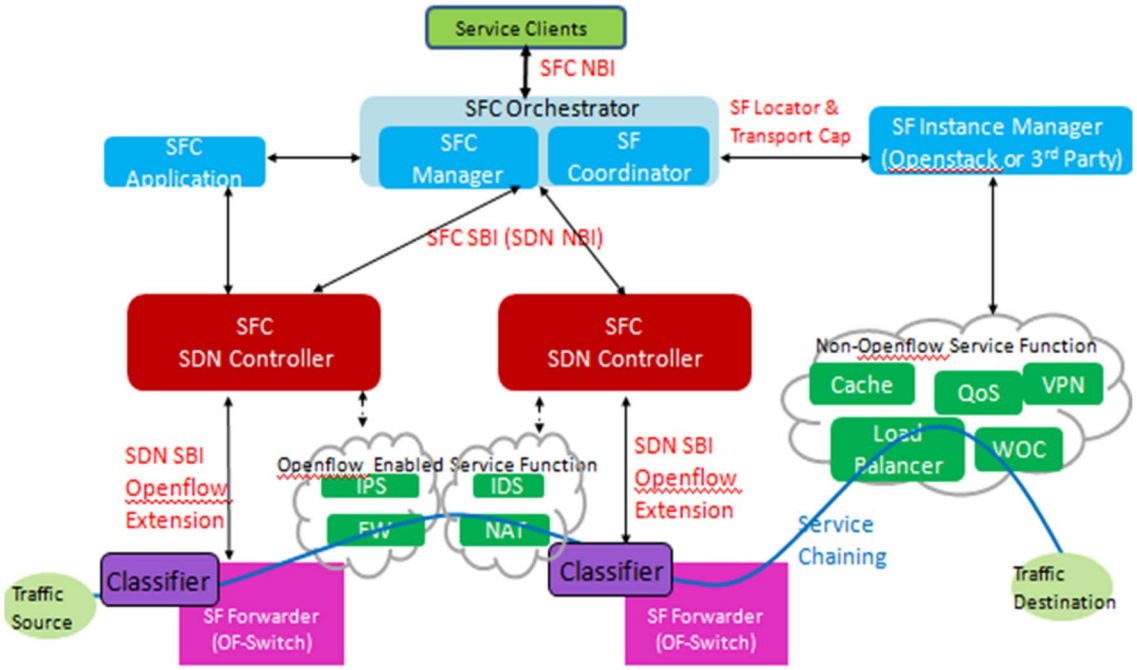
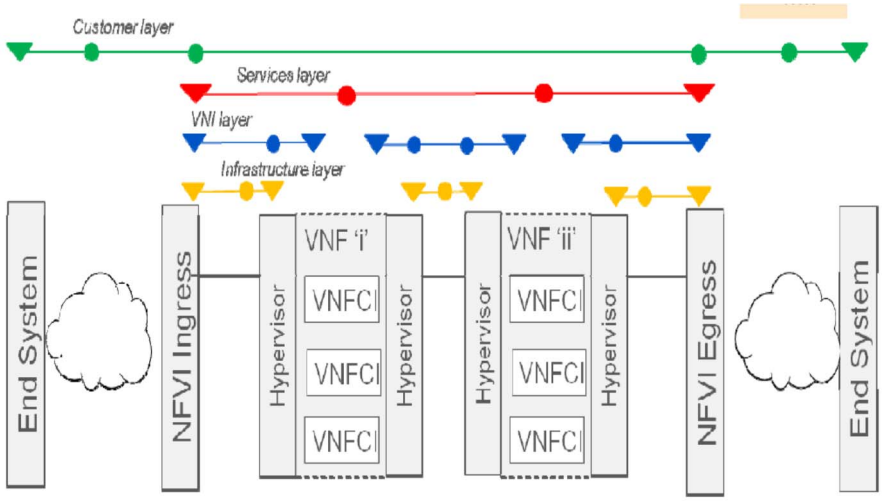
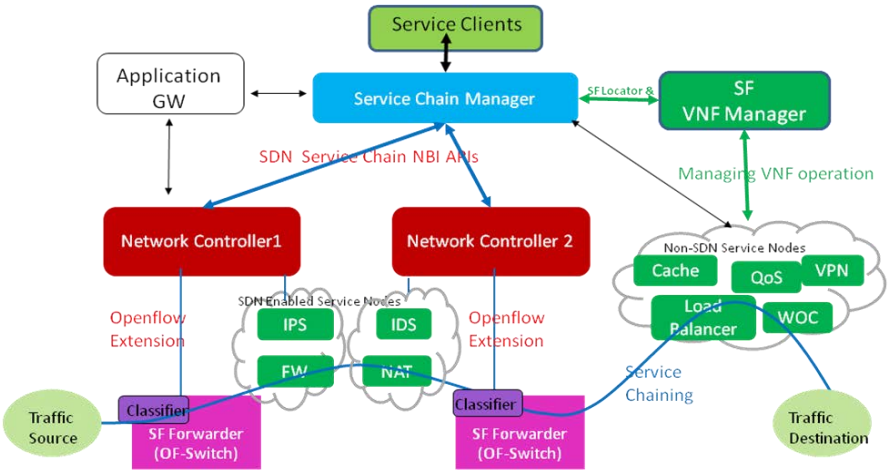



**Figure A.13: POC#28 SDN controlled VNF Forwarding Graph**

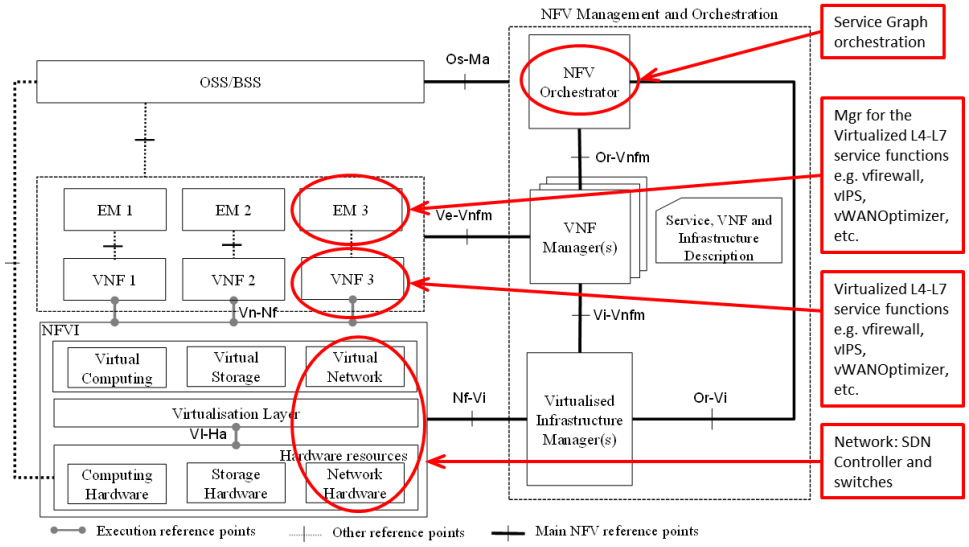
**Table A.12**

| Describe Application | SDN Controlled VNF Forwarding Graph |
|---|---|
| Role of SDN | In this PoC, the SDN is used to control individual flows traversing through their designated virtual network functions (VNF).<br>Assuming that the VNF are already interconnected, which might be controlled by traditional control protocols or SDN approaches.<br>This PoC is SDN controlled Service layer (the Red Line) as defined in ETSI GS NFV-INF 005 [i.1] clause 6.2.3 and in the reference diagram extracted from GS NFV-INF 005 [i.1] Network Function Virtualisation Infrastructure Architecture (figure 22):<br>  <br>Figure 22 – OAM MEP/MIP model |

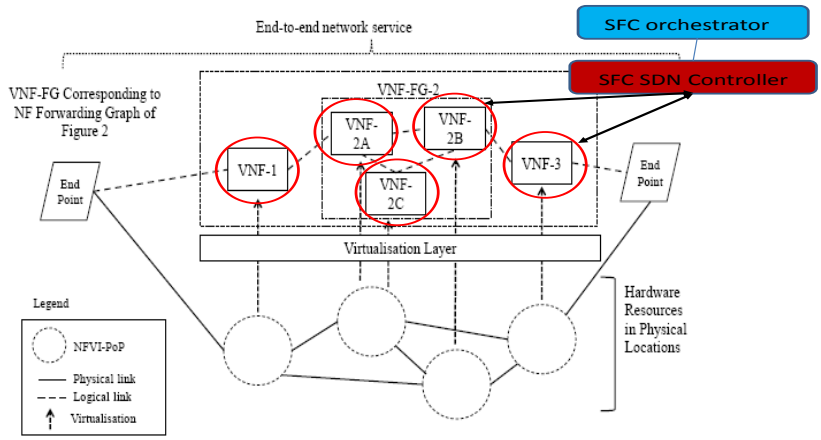
PoC Architecture




The following figures maps the PoC components to NFV Architectural framework (gs_NFV002v010101p.docx):

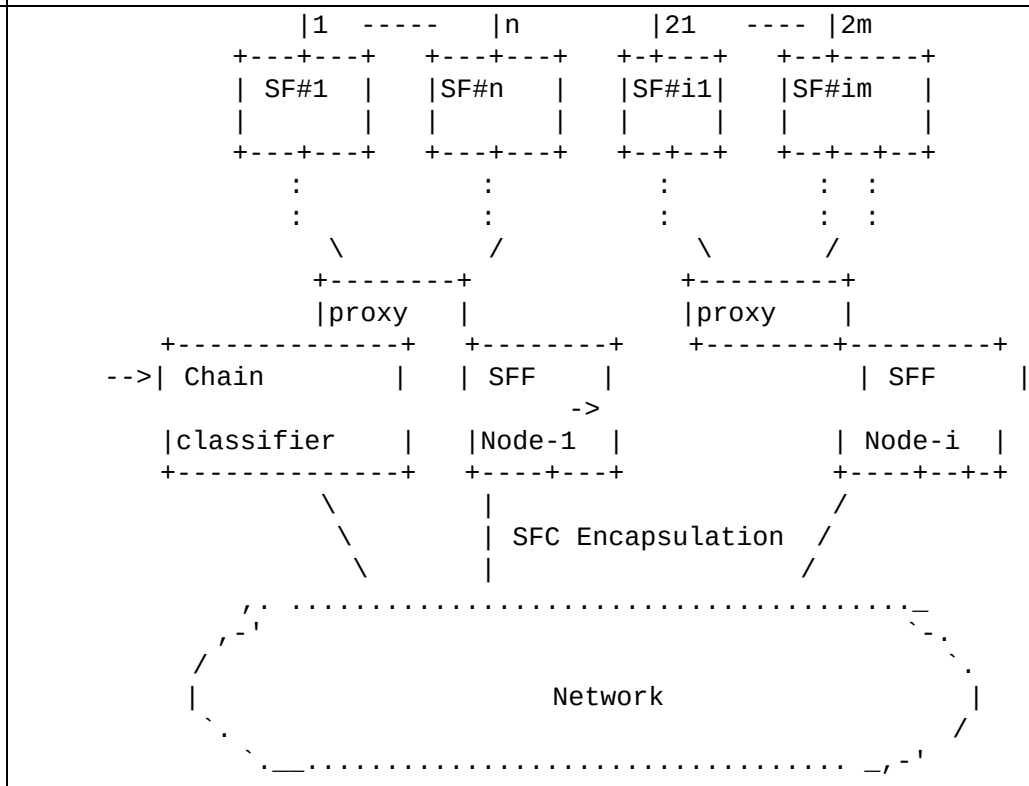



Mapping the PoC components to the End to End Service Forwarding Graph (ETSI GS NFV 002 [i.3] V1.1.1 (2013-10):




| Plane | Examples |
|---|---|
| Management | SFC Orchestrator, i.e. SFC manager and SF VNF Manager<br>SFC applications (e.g. SFC OAM functions) |
| Control | SFC SDN controllers |
| Data | Classifier<br>SF Forwarders, SF Proxy<br>SF Nodes |

| Describe Application | SDN Controlled VNF Forwarding Graph |
|---|---|
| | SFC management and control plane elements are responsible for translating user/application requests; converting requisite policies into network topology dependent paths; and disseminating steering commands to relevant forwarding nodes. SFC data plane components are responsible for carrying out the steering commands.<br>In the ONF SDN paradigm, the SFC Orchestrator requests necessary changes through the northbound interface (NBI) of the OpenFlow SDN controller, which in turn implements the requested changes via OpenFlow. If service function are not OpenFlow-speaking or are not under the domain of OpenFlow SDN controllers, the SFC Manager might need to interact directly with the service functions themselves or interact with another service function instance manager. |
| Role of NFV Orchestrator | The SFC Manager Component of the NFV Orchestrator (Service Orchestrator) translates clients' high-level intent policy based network-topology agnostic requirements into flow classification constructs, and sequences of L4-L7 service function constructs associated with the flows. The SFC Manager provides an interface for service clients to specify SFC requirements including the desired classification criteria, policies, sequence of service functions, etc.<br>The Orchestrator consists of two key components: the SF Chain Manager and the SF Instance Manager. The Chain Manager is responsible for translating user's SFC flow classification rules and SFC requirements into flow classification constructs and associated sequences of L4-L7 service functions. The Instance Manager is responsible for managing the life cycle of each service function instance, and for keeping track of each service function instance's locator and chain header encapsulation and transport capabilities. The Chain Manager will then pass these classification constructs, sequence of service functions, and the L4-L7 service function instance catalogue to the SDN Controller through the SDN NBI so that the SDN Controller might further translate them into southbound OpenFlow programming constructs. |
| Network Controller details | The SDN Controller is responsible for setting up service function steering/chaining paths. It translates flow classification constructs and associated sequences of SF locators into southbound OpenFlow programming commands. It locates the flow classifier and instructs it how to classify a flow and the flow packets with an appropriate SFC header. It is also responsible for locating the sequence of SF Forwarders associated with the sequence of SF Locators so as to enforce the flow going through the sequence of designated SF Nodes. That is, the SDN Controller will use those OpenFlow programming constructs to program the classifier to classify the traffic flows and program the SF Forwarders to steer the flow through the sequence of SF locators. |
| Network Controller to NFV Orchestrator interface and function | • This interface allows communication of VNF instance locator and flavor details as well as a service chain's service function ordering information. To support different types of SDN Controllers, the Service Chain Orchestrator (might be implemented in OpenStack) will interface with the SDN Controller through Plugin Driver mechanism. The Service Chain Orchestrator will pass the following information to the SDN Controller:<br>– Sequence of Requested Service Functions: This specifies the sequence of service function treatment for a tenant's traffic flow.<br>– Traffic Flow Classification rules: It consists of a set of flow descriptors.<br>– List of Available Service Function Instances: each SF Instance contains information such as the Instance's Locator, the Instance's Flavor. An example of a SF Instance is a firewall or an IDS. |

| Describe Application | SDN Controlled VNF Forwarding Graph |
|---|---|
| Data Forwarding Plane |  <br>The Service Function Forwarder (SFF) is responsible for forwarding data packets to their designated service function instances. As the case with other SFC entities, the SFF might come in physical, virtual or any other form factor. The SFF might make its forwarding/steering decision by matching the chain identification information carried in the SFC header with the next-hop information provided by the SDN controller. The SFC header would carry sufficient information for each SFF in the path to establish symmetric flows in forward and reverse directions, if necessary. |
| Data Control Plane (Programming plane) | OpenFlow 1.3+-capable SFFs might make its forwarding decisions on L2-L3 header fields such as MAC or IP addresses. With an OpenFlow version that supports L4 fields such as TCP and UDP ports, the SFF will be able to make forwarding decisions at L4. |
| Lessons learned and (new) requirements | One of the Key Components for Virtualised Network Functions:<br>• Data path have to be dynamically adapted to the Virtualised Network Functions as they are moved, added, or removed.<br>• Without SDN, it is very difficult, if not impossible, for data flows to dynamically steered to their designated service functions<br>• VNFs by different vendors have different features and need different provisioning. There are mechanisms today (e.g. OpenFlow or IETF's I2RS) for SDN controller to dynamically control L2/L3 switches and routers, it is virtually impossible to control security functions, such as Authentication VNF, FW/IPS/IDS VNFs because there is no standard interface to those VNFs<br>SDN controlled Service Graph forwarding might be deployed in today's network without requiring replacing existing equipment and control plan. But SDN Controller is not able to control non switch/router based VNFs yet. |
| Gaps Identified | There are many types of VNFs, the Video Optimization based VNF is very different from FW/IPS/IDS based VNF, or Router/switch based VNF.<br>In order for NFV Orchestration to automatically utilize various types of VNF, it is recommend that ETSI NFV to have an initiative to categorize VNFs, and then move on to define interfaces to various types of VNFs.<br>For example, for the category of Flow based Security Functions (VNFs), "Subject - Object - Function - Action" paradigm might be used to manage them as discussed in IETF I2NSF [i.11]<br>• Subject - Match values based on packet data Packet header or Packet payload<br>• Object - Match values based on context. E.g. State, time, geo-location, etc.<br>• Action- Egress processing, such as Invoke signaling; Packet forwarding and/or transformation; Possibility for SDN/NFV integration<br>• Function = Functional Profile – E.g. IPS:<Profile>, signature file, Anti-virus file, URL filtering file, Threat Database, etc. Integrated and one-pass checks on the content of packets.<br>The functional profile is vendor specific and differentiates vendor unique innovation. |

# A.13 POC#34: SDN-enabled Virtual EPC Gateway

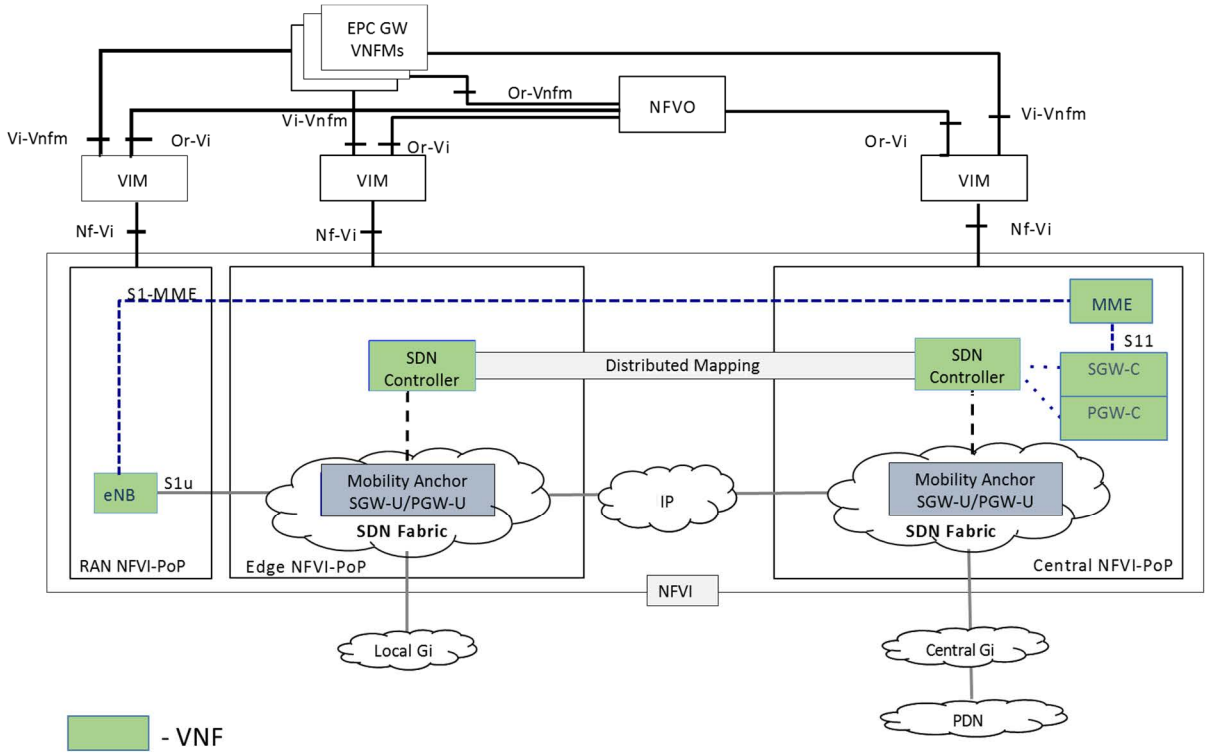



**Figure A.14: POC#34 SDN enabled Virtual EPC Gateway**

**Table A.13**

| | |
|---|---|
| Describe Application | This PoC verifies an enhanced EPC Virtualisation architecture: by introducing Software Defined Networking (SDN), EPC Virtual Functions might be distributed to location of choice.<br>Rather than virtualizing EPC as a whole, the PoC presents an SDN-based decomposition of EPC Gateways, SGW and PGW, into Control and Data functions.<br>The EPC Gateway decomposition is designed to meet 3GPP functionality and comply with SGW/PGW Data and Control external interfaces.<br>The distribution of Data Plane functions to the network edges, enables Local Breakout towards network edge content and services and the implementation of an Edge Service Chaining.<br>This solution enables service continuity for users handed over between network edge locations.<br>Virtual Network Functions in the PoC are secured with an Intrusion Detection System. |
| Role of SDN | SDN is enabling decomposition of EPC Gateways, SGW and PGW, into Control Plane and Data plane functions.<br>The EPC Data Plane is implemented based on SDN. SDN allows the programming of GTP tunnels, per attached users and active bearers.<br>The SDN switch strips off GTP tunnel encapsulation in case traffic needs to breakout locally.<br>SDN is also used to implement Service Chaining both in the core Gi-LAN and in the edge. |
| Role of the NFV Orchestrator | NFV Orchestrator instantiates the EPC and RAN VNFs, including EPC Control Plane, EPC Data Plane and eNBs. |
| SDN Network Controller details | SDN Controller is based on OpenDayLight, with required enhancements to support GTP. |

| | |
|---|---|
| | The SDN Controller is exposing an ACI (Application Control Interface) to the EPC Control Plane to be able to manage GTP tunnels, based on events indicated by the EPC Control. The downstream ACI is used to reflect user state changes, active bearer information and user mobility events, all mapped to EPC procedures (e.g. Attach, handover, etc.).<br>The upstream ACI is used for Data Plane usage reports.<br>The Controller's Resource Control Interface (RCI) uses OpenFlow 1.3 |
| SDN NC to VIM and NFV Orchestrator interfaces and functions | SDN Controller is running on OpenStack that is functioning as the Virtualised Infrastructure Manager (VIM).<br>The Orchestrator coordinates the allocation of virtual resources in the VIM and the Virtual Network Functions |
| Data Forwarding Plane | The EPC Data Forwarding Plane was implemented based on SW-based SDN switches, enhanced with GTP support, one acting as an Edge Mobile Anchor and the other as a Core Mobile Anchor.<br>Data Forwarding Plane between eNB and SDN Switch acting as an Edge Mobile Anchor uses GTP-U.<br>Data Forwarding Plane between SDN Switches, uses GTP-U.<br>Data Forwarding Plane between VNFs on the Gi-LAN and on the Edge Gi-LAN uses VXLAN. |
| Data Control Plane (Programming plane) | Data Control Plane is based on OpenFlow 1.3. OpenFlow rules are placed based on control messages received from the EPC Control Plane through the SDN Controller Application Control Interface (ACI) |
| Lessons learned and (new) requirements | An SDN-based vEPC Split has proven to support 3GPP functionality, while allowing better performance and scale and supporting placement of Data Plane functions, independently of Control Plane functions.<br>This architecture allows best-of-breed selection of EPC sub-functions.<br>The PoC architecture has proven to enable higher performance on the EPC Control Plane, by extracting Data Plane functionality and implementing it in an SDN infrastructure.<br>Data Plane performance is also proven to be high, based on OpenFlow switches.<br>vEPC functions might be placed anywhere, and specifically Data Plane functions might be pushed to network edges to allow local breakout towards edge content and services.<br>Network Edge service examples include network edge caching, edge CDN (Content Distribution Networking), multi-unicast of OTT Live Traffic and hair-pinning of video conference traffic.<br>Service continuity is maintained in the event of mobility between network edge locations.<br>Service Chaining is applicable in the network edge, avoiding over-subscription of edge services. |
| Gaps Identified | It is recommended that the VIM to SDN Controller interface be specified to support carrier-grade life cycle management |

## A.14 POC#38: Full ISO-7 layer stack fulfilment, activation and orchestration of VNFs in carrier networks

This PoC is built upon the ETSI/NFV reference architecture framework. The PoC includes VNFs from multiple vendors, each with its own Element Manger and VNF Manager. Additional components to the ETSI framework are highlighted in blue. These are described below.

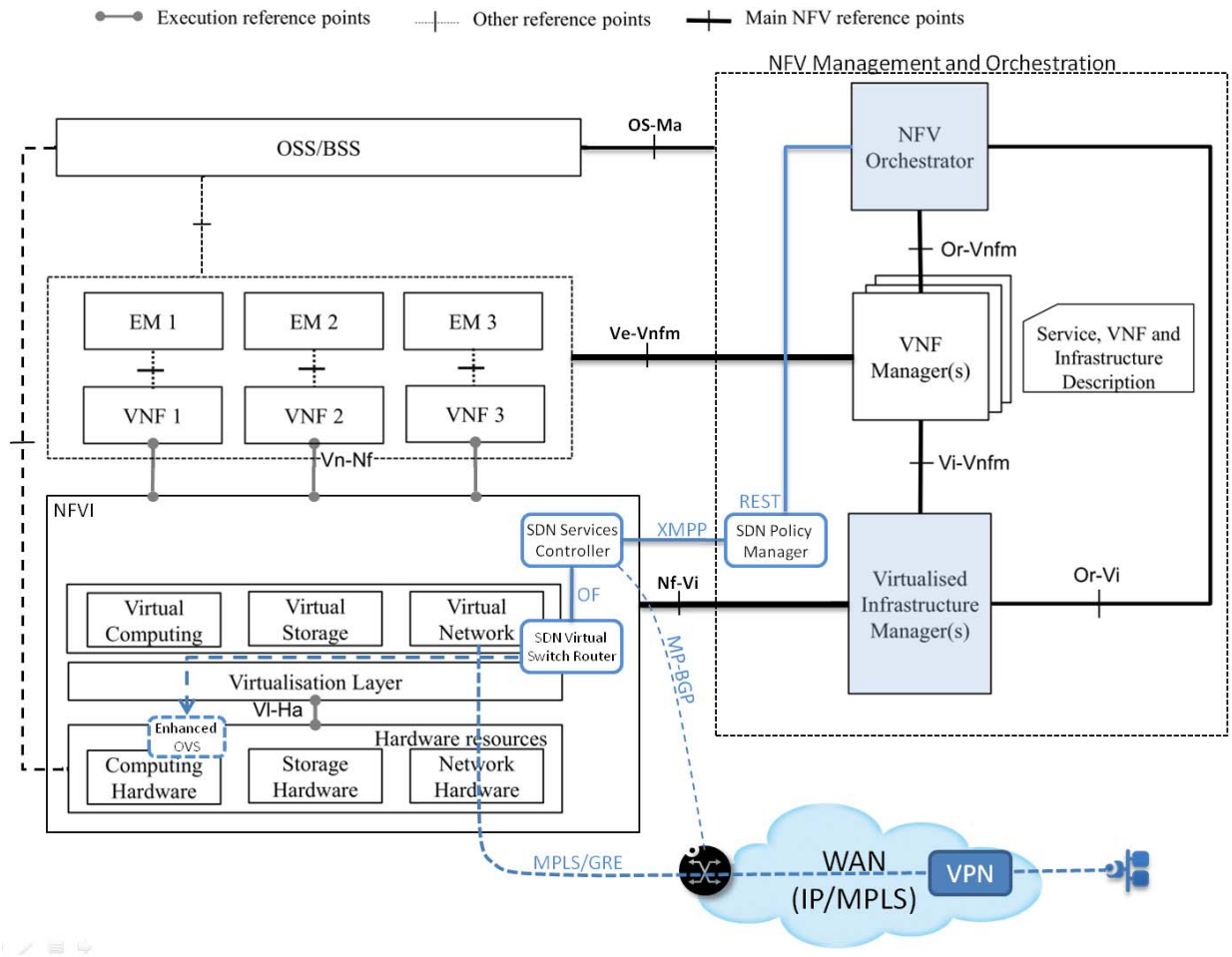



**Figure A.15: POC#38 SDN controlled VNF Forwarding Graph**

**Table A.14**

| | |
|---|---|
| Describe Application | **Use of SDN in an NFV architectural framework**<br>This POC is focussed on full ISO 7-layer stack fulfilment, activation and orchestration of VNFs in carrier networks. As part of fulfilment, the connectivity between VNFs and between VNFs and enterprise network customers' needs to be instantiated. SDN programming of the virtual network is used to automatically create the required connectivity as part of fulfilment. Creation of forwarding graphs to support service chaining is also utilized to show how additional VNFs might be added to existing customer services to provide new value added capabilities. |
| Role of SDN | The SDN Policy Manager contains the networking policy requirements for tenants in the NFVI. Each NFVI tenant represents a different virtual private network (VPN) which might be used to :<br>Associate VNFs to a specific end customer (e.g. vFW or vDPI for a specific enterprise)<br>Administrative domains for SP VNFs (e.g. management our OOB control networks)<br>Emulate the switch fabric connectivity between multiple VMs (VNFC) that might represent components of a single Network function (e.g. if individual line cards are represented as VMs)<br>In the following description the term *domain* will be used to describe the networking connectivity for a tenant or VPN. For each NFVI domain, policies might represent L2 or L3 connectivity between VNFs (including addressing requirements, security, and forwarding graphs) as well as how these domains will extend to MPLS VPNs in the event a VNF needs to be exposed to enterprise customers across the existing WAN network. |

| | |
|---|---|
| | The SDN Services Controller maintains the forwarding state of the domains, for example the connectivity between the VNFs and extension to the MPLS VPN network. Any new forwarding changes (e.g. add/move/delete of VNFs or changes in the VPN network) are processed by the controller and used to program the virtual network domain. The SDN services controller uses the SDN policy manager in the event further information on VNF connectivity or networking domains are required. The controller uses MP-BGP to communicate network changes with the MPLS WAN network and Openflow to program connectivity between the VNFs.<br>The SDN Virtual Switch Router is an enhanced OVS that resides locally with each Hypervisor. It is programmed by the SDN services controller to provide connectivity between VNFs instantiated as VMs within the hypervisor. VXLAN overlay tunnels are initiated between the hypervisors to allow L2 or L3 connectivity between relevant VNFs in the same domain. MPLS/GRE tunnels are initiated between the hypervisors and MPLS network if VNF connectivity to end enterprise customers is required. Security policies or forwarding graphs are also programmed in the Virtual Switch Router as defined by policy requirements. Connectivity with the MPLS WAN network is based on MP-BGP VPN-IPv4 routes and MPLS/GRE tunnels. Connectivity between the switch routers is based on IETF draft draft-ietf-bess-evpn-overlay.<br>Upon receiving requests from the NFV Portal, "Orchestrator" interacts with VIM and "VNF Managers" (through standardized APIs across multiple vendors) to validate (e.g. syntax, resource management, etc.) and finally executes the requests. When needed, the NFV Orchestrator (or the VNF Manager depending on the policy) elastically scales up/down or in/out a given VNF.<br>The VNFs shown in the figure above are likely to be interworking with some of the existing PNFs. More details on the complete list of VNFs, the SDN controller (e.g. vendor, version, models, etc.), and whether or not the test-bed environment includes both VNFs and PNFs will be described in the final PoC report.<br>When VNFs or VNFC are instantiated by the VIM, metadata is passed from the VIM to the HV which identifies the connectivity for interfaces of that VNF/VNFC. Metadata might specify virtual location by naming the relevant domain (network or subnet level), or specifically for a (virtual) port if service chaining is required. A VNF might be part of multiple domains (e.g. a vFW might have OOB management, public/insecure and private/secure interfaces that exist in different domains). A VNFC might also have an interface that represents the internal fabric to connect to other VNFC of the distributed VNF. In this scenario metadata is passed to represent the domains that each of the corresponding VNF/VNFC interfaces connect to. This metadata is captured by the SDN Virtual Switch Router and sent to the Services Controller. If the Services Controller already has local state of the domain, the controller will:<br>program the local switch router on the hypervisor with connectivity to other VNFs or MPLS VPNs in the same domain<br>program the other relevant VNFs already connected to the domain with connectivity to the new VNF<br>program the MPLS WAN network (using MP-BGP) with connectivity information to the new VNF<br>If required the SDN services controller will pull the relevant information from the SDN policy manager before taking the above actions.<br>If the VIM needs to re-instantiate or move the VNF (e.g. for failure recovery or scaling), the corresponding metadata for the VNF will be pushed to the new HV to ensure that the connectivity for VNF at the new location might be programmed via the SDN Service Controller.<br>The SDN components are highlighted in blue on the ETSI framework (above). The SDN Policy Manager is a centralized component and might be considered part of the MANO framework. The SDN Services Controller is part of the NFVI and controls virtual network. Multiple SDN services controllers might be added for scaling (i.e. as hypervisor endpoints are added). The SDN Virtual Switch Router is part of the NFVI, and individual instances are installed as part of the hypervisor on the compute platforms. |

| | |
|---|---|
| Role of the NFV Orchestrator | The NFV Orchestrator interacts with the SDN Policy Manager through a standard REpresentational State Transfer (REST) API. As part of this interaction the Orchestrator is able to create network and security policies as well as virtual interfaces and domains within the SDN Policy Manager. These policies and interfaces are defined within Policy Manager logical constructs known as domains and zones and are applied to virtual port interfaces (vport) at both layers 2 and 3. VNFs are linked to vports within the Policy Manager through metadata that is configured at the virtual machine level by NFV Orchestrator and instantiated via the VIM. The metadata is read by the SDN Virtual Switch Router as VNFs are created and will be used by the SDN Policy Controller to determine network connectivity, how traffic is handled and which policies are applied to the VNF virtual machine network interfaces. Templates might also be used to apply common network policies to multiple domains in the NFVI. |
| SDN Network Controller details | The SDN Services Controller uses Openflow to configure L2 and L3 connectivity for each VNF at the Virtual Switch Router on the hypervisor. Multiple network domains might be supported using separate L2 and L3 forwarding tables on the enhanced OVS and separate VXLAN Network Identifiers (VNI) on overlay VXLAN tunnels to other VNFs on other hypervisors. When a VNF is created:<br>Metadata is passed from the VIM to Virtualisation layer as part of VM/VNF creation<br>This metadata is passed from the Virtual Switch Router to the Services Controller, and used by Controller to query the defined policies on Policy Manager. This determines connectivity requirements for the created VNF.<br>Based on the required connectivity policies for the VNF, the SDN Services Controller will use Openflow to configure connectivity between the created VNF and existing VNFs in the same domain and zones as required. This connectivity might include security rules and forwarding graphs.<br>If WAN connectivity is required, The SDN Services Controller will use MP-BGP to signal the MPLS WAN with the VNF information. WAN interworking uses MP-BGP VPN-IPv4. The ability to signal VNF creation to the MPLS WAN network supports VNF portability, e.g. VNFs might be moved across hypervisors or between NFVI's and still be advertised to the existing WAN network. In the POC a MPLS WAN gateway was utilized to translate MP-BGP routes from the DC/NFVI to MPLS core, and MPLS core to DC/NFVI, while changing the NH in both directions to the local gateway local address. This isolated the addresses of the NFVI from the core. Tunnels from VNF/hypervisors are created to the gateway in the NFVI, and tunnels from the WAN are also created to the gateway from the WAN. |
| SDN NC to VIM and NFV Orchestrator interfaces and functions | The NFV Orchestrator provides the primary interface between the VIM and the SDN Policy Manager. In both instances these interfaces are through standard REST APIs. NFV Orchestrator will configure the VNF virtual machines with metadata obtained from the SDN Policy Manager. The VIM instantiates the VNF, and this metadata will be used by the Virtual Switch Router to determine the policies to be applied to the VNF interfaces.<br>The NFV Orchestrator streamlines the integration between the network management function in VIM and SDN Policy Manager and provides operator and user simplified overviews and management options for the entire L2 and L3 networks. The NFV Orchestrator also ensures the consistency between the network view in VIM and the actual network in SDN. |
| Data Forwarding Plane | Data plane forwarding between VNFs is based on draft-ietf-bess-evpn-overlay and IETF RFC 7348 [i.21] Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualised Layer 2 Networks over Layer 3 Networks.<br>Data plane between the VNFs and MPLS network is MPLS/GRE. For this scenario the MPLS WAN gateway was responsible for terminating tunnels from the NFVI before forwarding traffic to the MPLS core. |
| Data Control Plane (Programming plane) | Control plane programming between the SDN Services Controller and SDN Virtual Switch Router on the hypervisors uses Openflow.<br>Control plane between the SDN Services Controller and MPLS network is MP-BGP using VPN-IPv4 address family. |

| | |
|---|---|
| Lessons learned, gaps identified and (new) requirements | The following items are for further study.<br>(A) Selective service chaining<br>For interworking with MPLS VPNs (specifically when providing VAS services for enterprise customers e.g. FW/NAT for Internet services for VPN enterprise customers), a default MP-BGP route was injected into the MPLS VPN to attract internet traffic into the FW/NAT function in the NFVI before forwarding to the internet. This method attracts all traffic that is not destined to VPN prefixes/sites and does not discriminate either based on the source or type of traffic. This was done to be backwards compatible with the existing MPLS network.<br>A method to create a service chain/forwarding graph from the customer endpoint to the first VNF in a service chain provides the options to provide VAS services for selective sites. Additionally the ability to utilize policy based forwarding rules to selective forward only specific types of traffic provides additional control that might help scale the VNF to support relevant traffic. These requirements needs to be linked to the VNF creation and needs to cater for VNF portability.<br>Following is an example of requirements :<br>• labelling each VNF with a unique tag which might be advertised (along with VNF location) during creation time<br>• PBF policy at the customer site to the VNF tag<br>• a tunnelling mechanism to deliver customer site traffic to the VNF<br>When the customer site receives an advertisement with the VNF location, it updates its PBF policy with this information (including location) and uses the advertised tunnelling method to steer traffic to the VNF. This allows a service chain to be defined at the customer site that is linked to the VNF such that when the VNF creation is signalled to the WAN network, a service chain might be created regardless of which hypervisor or NFVI location is utilized. An example is to label VNFs with a unique ESI value advertised in an EVPN route with reachability information for the VNF.<br>(B) Support for L2 VAS services<br>For interworking with MPLS VPNs (specifically when providing VAS services for enterprise customers), layer 3 connectivity was used. This was done to be backwards compatible with the existing MPLS network. VNFs might also need to be deployed as L2 (bump in the wire). In this scenario a method of extending L2 connectivity into the MPLS VPN is required. IETF RFC 7432 [i.23] defines EVPN MP-BGP. EVPN has the added benefit over VPN-IPv4 of supporting MAC address advertisement. This could be used to extend L2 NFVI domains into MPLS VPNs. In combination with VXLAN data plane, EVPN could be used to support service chains from customer sites to VNFs. E.g. service chain policy at the customer site to a specific VNF IP/MAC/ESI. When an EVPN MP-BGP route is received signalling VNF creation, the advertised data plane tunnel method in the MP-BGP route might be used to create connectivity directly to the VNF.<br>(C) Portability<br>One of the benefits of Virtualisation is the ability to create networks functions where they are most needed, e.g. based on geography or resources. This portability requires a method of identifying VNFs and corresponding connectivity requirements, regardless of where the VNF is created.<br>In this POC, this was achieved by using metadata when VNFs were created by the VIM. This metadata was utilized by the SDN solution to identify the VNF and ensure the correct connectivity. |

| | |
|---|---|
| **Gaps Identified** | For service chaining, a default route to the first VNF is injected into the MPLS VPN network. In the event this first VNF fails traffic will be lost. For redundancy 2 x FW/NAT VNFs were required, however a default route pointing at Virtual IP address was not supported. This has been addressed in subsequent releases of the SDN solution but not tested as part of this POC.<br>Other suggestions for EVE005 :<br>Portability is probably covered in a different area, however the fact VNFs might be instantiated anywhere has implication on SDN, which is responsible for the connectivity of these portable VNFs. As such there is a need for a linkage between portability and the SDN requirements to carry this information. In theory when the VIM creates the VNF/C (based on what whatever resource criteria) a method is needed to identify the location of the VM, other relevant VM/endpoints that it needs to communicate to, and create this connectivity. How the connectivity occurs is dependent on the SDN resource (as per clause 4.3.3). E.g. in our scenario this SDN resource/endpoint is really in the virtual switch in the compute hardware (i.e. case c). So the VIM would pre-configure the network policies in advance (via the policy manager). When the VIM instantiates the VNF, the metadata linked to the VM creation is added to allow identification of where this fits from a network perspective. The metadata is captured via the virtual switch router, and forwarded to the controller. The controller then polls the policy manager with this metadata to identify the relevant networking policy, and then programs the connectivity for the VNF via the virtual switch router component on the compute node, and all other relevant endpoints that need connectivity to the VNF. This is an automated pull model, i.e. pre-configure all policy and then when a VNF is created pull the policy and configure the connectivity for the VNFs automatically. The metadata here is just to correlate/identify the VNF creation to where it fits in the network policy, so it needs to be defined at both the policy manager and VNF creation. The controller here then becomes a proxy to get the relevant policy information when the VNF is created, and then program the relevant connectivity.<br>It is recommended that Clause 5 also mention connectivity between VNFs and end customers<br>Clause 5.5.2 Virtualisation of SDN controller: when VNF connectivity requires the SDN controller, and when the SDN controller is put into a VNF it still requires external connectivity to other functions. This is somewhat recursive with reliability implications, e.g. like trying to manage a distributed network with only inband connections. |

# Annex B (informative): SDN Use Cases in NFV environment

## B.1 Introduction

This clause contains a number of use cases that illustrate scenario with SDN combined with NFV.

## B.2 Multi-Layer Bandwidth on Demand

### B.2.1 Introduction

SWA GS clause A.3 contains ETSI NFV use cases related to SDN. It currently contains one use case covering *L2 Transparent Network Service Chaining with Traffic Steering.* This clause adds additional uses cases. The set of use cases focuses on expanding the coverage to include multi-layer use cases including circuit oriented networks (e.g. optical).

### B.2.2 Problem Description

Networks are constructed from elements operating in different network layers. This structure is usually not visible to the end user and is independent of service definition for that user. A service request from the end user perspective is best stated in terms of requirements important to the user and independent of the network instantiation of the service within a multi-layer network. For example, a service request for connectivity between a client and storage/compute resource is best stated in terms of requirement bandwidth, delay, reliability, etc. and not in terms of the technologies used to provide the service.

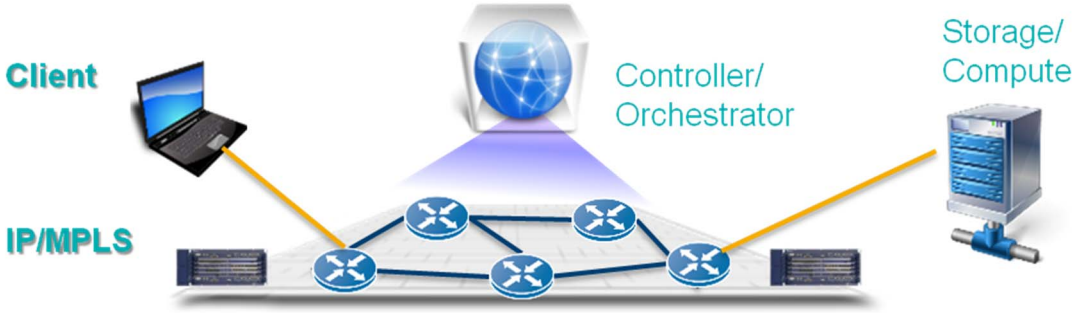



**Figure B.1: Service Request**

Automation of the process to map the service request requirements into available network resources is needed. Dynamic management of these resources might map the service into the appropriate network layer for the most efficient use of network resources. NFV/SDN might be used to automate this mapping.

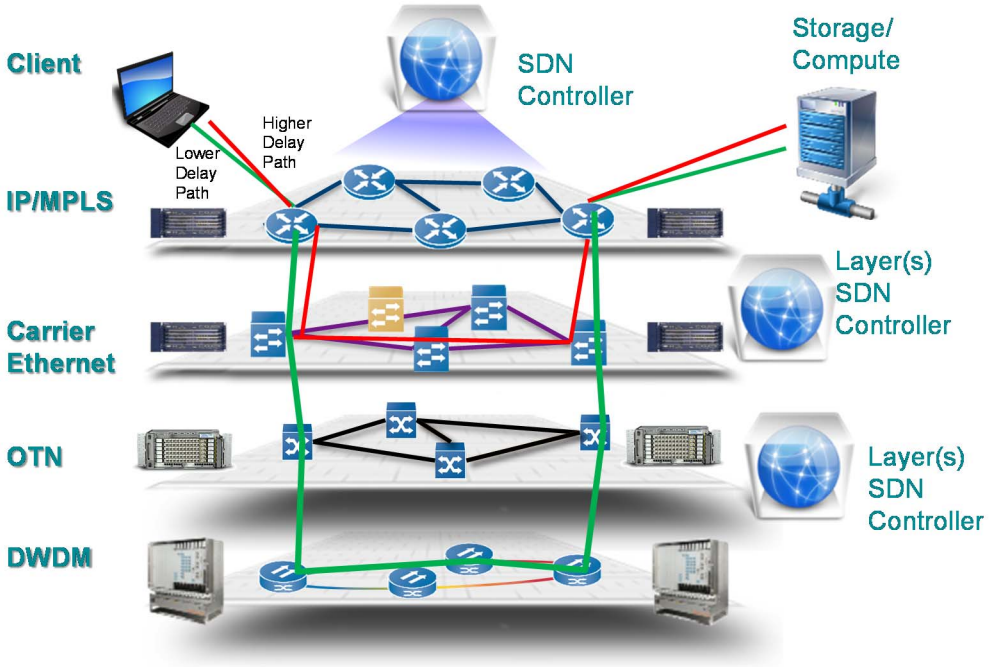



**Figure B.2: Service Instantiation**

## B.2.3 Solution Description

The following procedure will be followed:

A service request is made to the upper SDN Controller. It requests the bandwidth, latency, and other constraints important to the end user application. The role of this controller is to match the service requirements into the layer(s) of the network that will be needed to satisfy the request. This might result in one or more layers being involved in the service instantiation.

The upper SDN Controller maps the request into resources that best match the required service parameters without wasting network resources ("best worst fit"). For example a very low latency requirement might require the service to be carried over the shortest possible optical paths to lower total latency. Services not requiring such low latency would not be routed over these paths. In the figure these two delay options are shown. One path through the Carrier Ethernet Layer and the other path through the lower delay optical switching layer.

The required layer SDN Controllers are instructed by the upper SDN Controller to make the flow connections required to satisfy the service. For instance, for services being instantiated via Carrier Ethernet, if not enough bandwidth is available in the Carrier Ethernet network layer additional bandwidth might be requested by the upper SDN Controller from the DWDM layer's SDN Controller; if the request is satisfied the new resource is made available to the Carrier Ethernet layer.

# B.3 Bandwidth Defragmentation

## B.3.1 Problem Description

In current multi-layer networks channelized resources become fragmented as services are instantiated and removed during network operation (see Multi-layer Bandwidth on Demand). Today operators periodically audit the circuit oriented portions (e.g. SONET/SDH/OTN/Optical) of their networks to defragment these resources. Defragmentation means moving services (either in service affecting or non-service affecting manner) to allow larger bandwidth service to be instantiated when without defragmentation, such instantiation would not be possible.

### B.3.2 Solution Description

Under NFV/SDN control, the upper SDN Controller or an SDN controller for a layer network is used to instantiate and remove all services. Therefore there is continual awareness of the network topology. This allows continuous "defragmentation" of resources. An example of this is:

1) A service request arrives at an SDN Controller. The Controller decides which layer will be used to implement the service in the "best worst fit" solution (see above). For example, a latency requirement might force the use of a DWDM resource rather than IP/MPLS.

2) The request is passed from the upper SDN Controller to the DWDM SDN Controller to service the request. The SDN Controller maps the required bandwidth into the available bandwidth. It finds that there is not enough bandwidth available to grant the request.

3) When de-fragmentation is being used, while processing the service request, the controller will move lower bandwidth service(s) to free enough bandwidth to instantiate the service request. If enough bandwidth to service the request is not found in this manner the layer SDN Controller might refuse the service request or bring up another DWDM lambda to service it.

## B.4 Policy Based Configuration

### B.4.1 Problem Description

In a number of cases a large number of Network Services are expected to be deployed by various Tenants or Cloud administrators, following a common set of rules specifying how the traffic between the various deployed VNFs is supposed to flow, possibly through some rerouting of the traffic via other VNFs responsible for common tasks such as Firewalling, NAT or DPI.

Group Based Policy projects progressing in Openstack and OpenDaylight (ODL) are addressing these use cases. They rely on a declarative approach to define how data traffic is supposed to flow between network resources. The goal is to facilitate the deployment of Network Services by automating the configuration of the network and leverage SDN to apply policies dynamically

Network Intent Composition (ODL NIC) in OpenDaylight is also taking the view that a configuration of a network is a descriptive way to get what is desired from the communication infrastructure.

### B.4.2 Solution Description

Policy Based configuration separates information about connectivity requirements from information about the underlying details of the SDN network infrastructure.

A set of components are responsible for managing the policy definition, the associated configuration and related state.

Renderer components are responsible for applying the policy as specified by the user, to the SDN network infrastructure.

At a high level, Policies are typically expressed as a contract between two groups of Endpoints.

EndPoints might be part of the resources managed by the SDN Controllers of the concerned administrative domain. They might also be external entities which are expected to send/receive traffic to some of the resources managed in the domain concerned.

Some Policy Rules in the contract describing the policy are used to ensure that appropriate actions will be taken on the traffic between the concerned Endpoint groups when some Classification criteria are met.

Policy Model

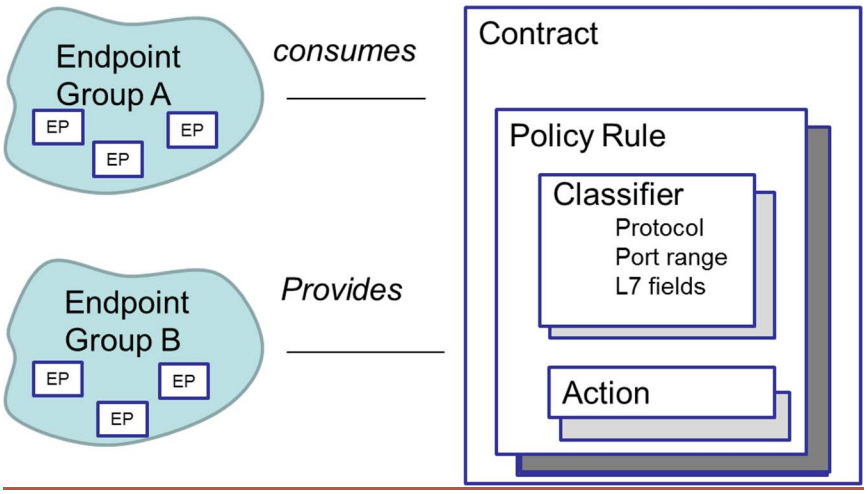



**Figure B.3: Policy applying to Endpoint Groups**

Architectural view

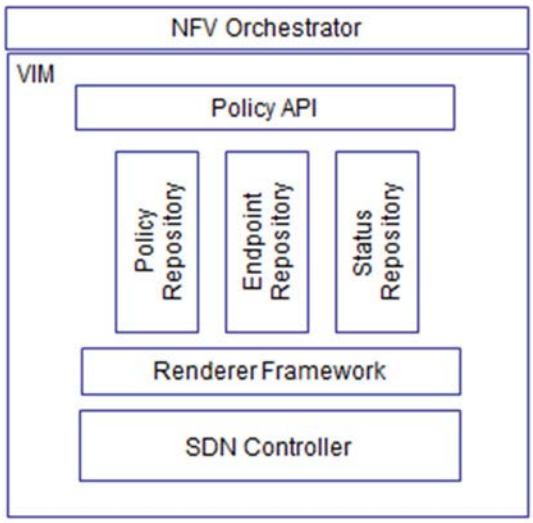



**Figure B.4: VIM Based Policy implementation**

GBP for example is implemented by the VIM (as for Openstack GBP, and possibly Congress). Other Policy flavour is implemented by the SDN Network Controller (as for ODL GBP and ODL NIC).

Applying a policy might trigger some interactions with the NFVO and/or the VIM as it possibly modifies deployed Network Services.

For example when traffic matching a policy is redirected to go through some transparent middle boxes, the renderer might reconfigure some virtual switches to encapsulate the traffic in a corresponding service function chain, and possibly instantiate VNFs which need to be part of it.

New repositories are needed to support GBP, to store the Abstract Policies, the Endpoints and possibly additional status information (for example: availability of VNFs of a particular type).

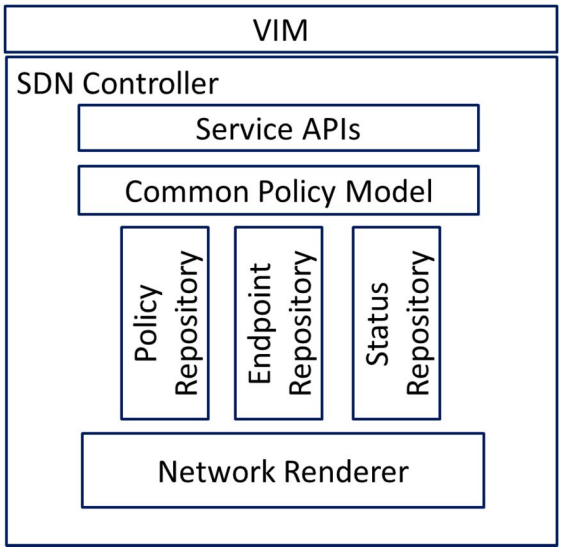



**Figure B.5: SDN Controller Policy Model**

It is noted that Policy Based Models are seen also as an internal means to address the multiple writer problem, which takes place when independently designed or implemented APIs need to coexists while driving a common set of network resources. The OpenDaylight GBP API for example might be used to implement various Openstack neutron APIs consistently.

### B.4.3 Scope of Policy Based Configuration

A typical "Action" in the OpenDayLight Policy Model could allow redirecting the traffic between two VNFs on a Service Function Chain (SFC). The chain might then be seen as a Network Service dynamically created as a result of a policy configuration.

GBP might be applied inside a VNF or inside a Network Service. Typical usage of GBP is to insert firewall and DPI functions in a Network Service, for security purposes.

Finally it might be used to steer external traffic, issued by component external to the administrative domain covered by the SDN controller for example, through a Network Service.

### B.4.4 Questions raised by Policy Based Configuration

Implementations differ on what type of Rules might be used to describe what type of traffic between groups is concerned by the policy. Does the rule apply only to resources managed by the administrative domain, does it deal with Tenant's end users, with Time of Day or various network events such as Radio Access Type change for Gi- LAN use case?

In addition, the question is at which level is the policy repository located? Is it part of the VIM or of the SDN Controller? Or even is it shared with the virtual or physical switches in some cases?

## B.5 Virtual CPE

### B.5.1 Problem Description

Nowadays, deployment of new customer services requires change to the access equipment on the customer premise, as well as activation of configuration on Operator Service platform. This practice increases the cost of new CPE and service delivery, as well as CPE and service maintenance, especially in complex service created by chaining existing services or services from partners. It also decreases the ability of customer self-care, for example adding new Value Added Services (VAS) in near real-time for users in its home network.

There are many studies about vCPE in the industry. It is about simplification of customer infrastructure to layer 2 access, shifting layer 3 functionalities from customer home network to the edge of Operator network.

Below, as example, is description of a two use cases of service provisioning.

The first is VAS - Parental control and second Anti-DDoS service.

Solution describes how Network Controller (SDN Controller), VNFO, VNFM, VIM and EMS are used in vCPE architecture, to steer traffic to appropriate service functions to make service chain.

**Use case Parental Control:**

Parents want to prevent their children from visiting adult network resources and other particular internet resources.

This control is implemented per children's device basis. Parent manage this control through the customer portal.

1) Parent1 accesses the portal using credentials he was given after ordering the service.
2) Parent1 registers the terminal he uses to access the portal. The system now knows that a particular MAC address (MAC1 from the table on Slide 2) is associated with user 'Parent1' under the main customer account.
3) After that, he registers the terminals and users for Parent2 (MAC2, laptop), Child1 (MAC3, smartphone) and Child2 (MAC4, tablet).
4) For Parent1 and Parent2 terminals parental control is left in 'Off' state.
5) For MAC3 and MAC4 terminals, 'Parental Control - Teen' and 'Parental Control - Pre-teen' are selected, respectively.
6) Child1 accesses his Facebook page and views his 'Friends' page.
7) He tries to open a link to adult content and is blocked from doing that.
8) Child2 tries to access Facebook to check his friend's birthday date and is blocked from doing that.
9) He asks his Parent1 to give him access.
10) Parent1 logs into the self-care portal and disables the parental control on Child2 tablet (MAC4) temporarily.
11) Child2 checks his friend's birthday successfully and Parent1 turns the parental control feature back on or waits for the Parental Control inactivity timer to expires.
12) Child2 tries to access Facebook again and is blocked from doing that.

**Use case DoS/DDoS protection:**

Service provider wants to prevent outside DoS attacks to subscribers.

This function is typically preconfigured by system administrators.

1) All inbound internet traffic goes through Intelligent Firewall/Traffic Analyzer VNF.
2) System administrator makes pre-configuration for DoS detection - signatures, SYN/FIN/ACK packets number, min/max packets per second threshold, stateful inspection, etc.
3) Subscribers protected from DoS attack and still able access resources which located in affected subnets.

## B.5.2 Solution Description

Figures B.6 and B.7 are illustrative vCPE solution architecture, with MANO functional blocks, and Network Controller for service chaining for two use cases:

**Use case Parental Control:**

Below is the flow of management actions to provision the parental control service, including activity of the Network Controller:

1) Parent1 accesses the portal using credentials he was given after ordering the service.
2) Parent1 turns on Parental control service.

3) Customer portal forwards information about MAC addresses and associated policies for parental control to OSS/BSS.

4) Forwarding Path configuration change request:

- OSS/BSS sends FP-change request to NFVO.
- NFVO forwards request to VIM.
- VIM makes changes in SDN Controller configuration.
- Network Controller changes traffic flow path for particular packets based on MAC address information.

5) VNF configuration change request:

- OSS/BSS sends VNF-change request to VNF Manager.
- VNF Manager forwards request EMS.
- EMS makes changes in DPI VNF configuration (specific internet resource filters).
- DPI VNF processed traffic filtering accordingly to customer portal requirements.

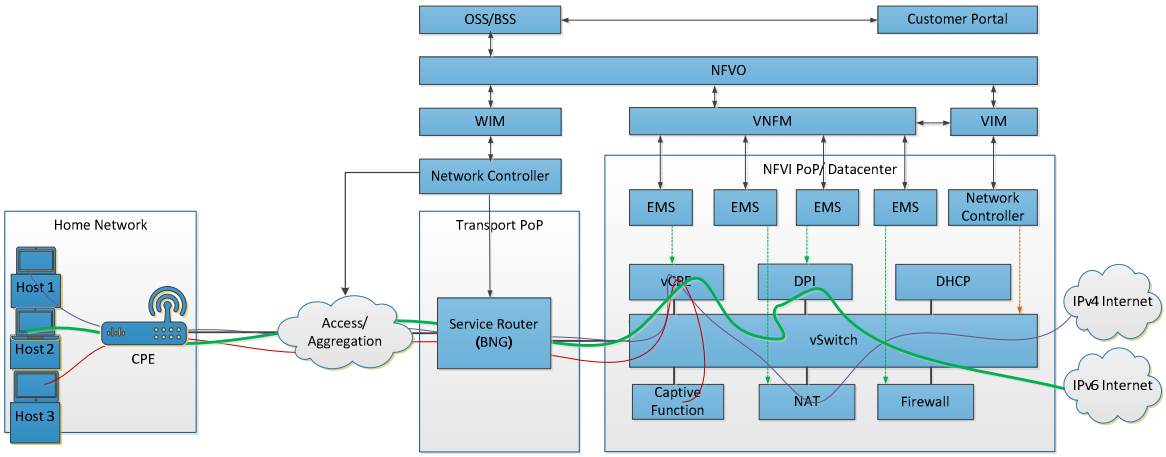



**Figure B.6: vCPE Solution architecture - Parental Control use case**

**Use case DoS/DDoS protection:**

1) DoS/DDoS attack detected from particular subnet/subnets by VNF Firewall(TA) VNF. Firewall(TA) VNF also recognized legitimate traffic from these particular subnets because connection was initiated by internal subscribers.

2) Forwarding Path configuration change request:

- Firewall(TA) VNF sends FP-change request to NFVO.
- NFVO forwards change request to VIM.
- VIM makes Network Controller configuration changes.
- Network Controller redirects traffic flows from particular subnet/subnets to DPI.

3) VNF configuration change request:

- Firewall(TA) VNF sends VNF-change request to NFVO.
- NFVO forwards change request to VNFM.
- VNFM forwards request to appropriate EMS.
- EMS makes DPI VNF configuration changes.

- DPI VNF renews filters and traffic might be cleaned precisely.
- DPI blocks only non-legitimate traffic and passes all legitimate traffic to subscribers.

4) After amount of time DPI engine recognized there is no DoS/DDoS traffic anymore from particular subnets.

5) Forwarding Path-configuration change request:

- DPI VNF sends FP-change request to NFVO.
- NFVO forwards change request to VIM.
- VIM makes Network Controller configuration changes.
- Network Controller redirects traffic flows from particular subnet/subnets to Firewall(TA) VNF.

6) VNF configuration change request:

- DPI VNF sends VNF-change request to NFVO.
- NFVO forwards VNF-change request to VNFM.
- VNFM forwards request to appropriate EMS.
- EMS makes DPI VNF configuration changes.
- DPI VNF removing unnecessary filters.

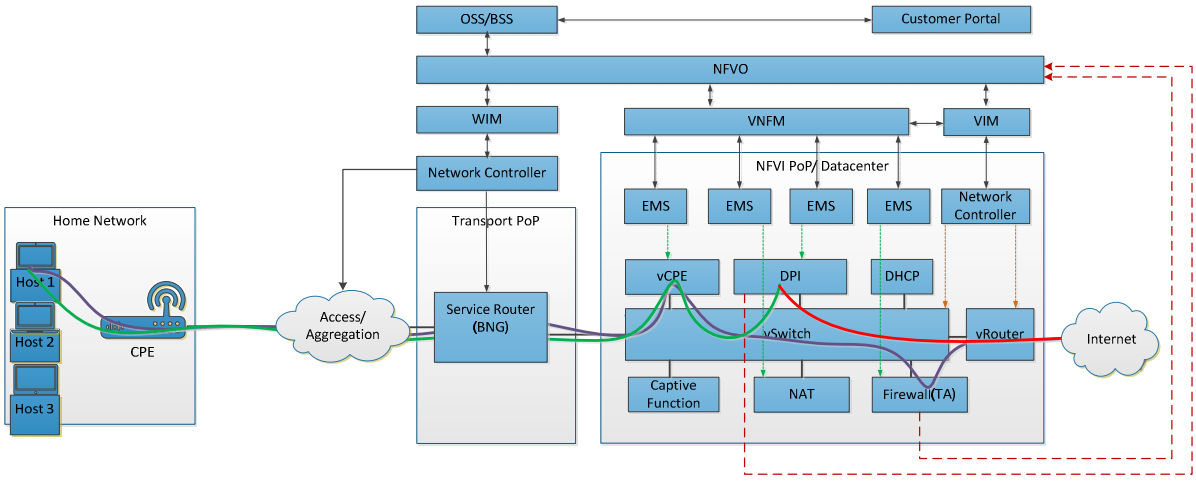



**Figure B.7: vCPE Solution architecture - DoS/DdoS protection use case**

# Annex C (informative): Comparison of Opensource SDN Controller

## C.1 Introduction

As part of ETSI NFV reference architecture is NFVI+VIM, and network elements. As network evolve towards SDN, SDN switches and SDN controllers are being integrated in this architecture. Both non-opensource and open source software-based controllers are being developed and are available in the industry. A number of opensource software implementations are available. The objective of this clause is to collect a list of publicly available opensource SDN controller and perform a comparison according to a common set of criteria.

## C.2 List of Opensource SDN controller

### C.2.0 Introduction

There are a number of opensource controllers available today.

The 1st one was NOX opensourced in 2008.

Other controllers came along such as POX, or Beacon is 2010, Trema which is a Ruby based controller or Ryu.

Floodlight was issued out of Beacon and released under the Apache® 2.0 licence (see note).

NOTE 1: "Apache is a registered trademark of the Apache Software Foundation. This information is given for the convenience of users of the present document and does not constitute an endorsement by ETSI of the product named. Equivalent products may be used if they can be shown to lead to the same results".

Lately some companies joined forces to release OpenDaylight under Linux® Foundation (see note). While another company was releasing OpenContrail in Opensource.

NOTE 2: "Linux is a registered trademark of Linus Torvalds. This information is given for the convenience of users of the present document and does not constitute an endorsement by ETSI of the product named. Equivalent products may be used if they can be shown to lead to the same results".

Last, as a challenger to OpenDaylight, ONOS was released in November 2014, out of a collaboration between multiple vendors and universities.

**Table C.1: list of Opensource SDN controller**

| | Name | URL | Release/Version |
|---|---|---|---|
| 1 | Floodlight | [i.12] | 1.0 |
| 2 | OpenDaylight | [i.13] | Lithium |
| 3 | OpenContrail | [i.14] | 1.2 |
| 4 | ONOS | [i.15] | Blackbird |
| 5 | Ryu | [i.16] | 3.2.4 |
| 6 | MidoNet® | [i.17] | 05-2015 |
| NOTE: "Midonet is the trade name of a collaborative open source project from Midokura SARL Ltd. This information is given for the convenience of users of the present document and does not constitute an endorsement by ETSI of the product named. Equivalent products may be used if they can be shown to lead to the same results". | | | |

Other opensource controllers have been developed such as: ONIX, Helios, Hyperflow, Maestro, Kandoo, NOX, Indigo, etc.

## C.2.1 Floodlight

The Floodlight [i.12] Open SDN Controller is an enterprise-class, Apache-licensed, Java®-based (see note) OpenFlow Controller, supporting physical or virtual SDN switches as shown in figure C.1. It is supported by a community of developers.It was forked from the Beacon controller, originally developed at Stanford.

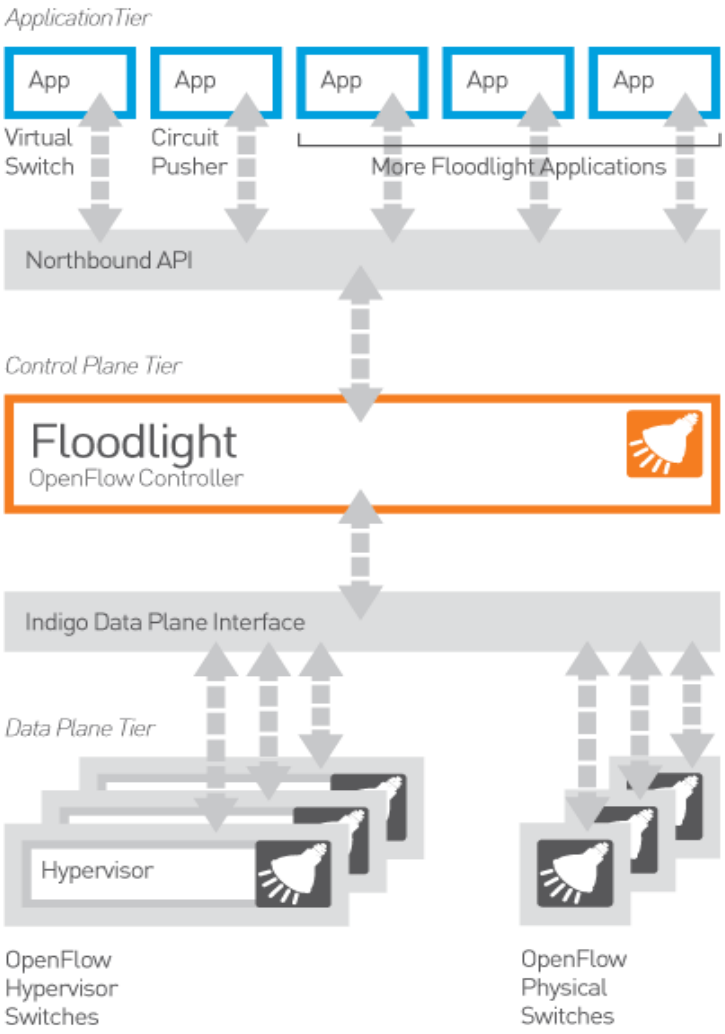



**Figure C.1: Floodlight SDN controller architecture overview**

NOTE: "Java is a registered trademark of Sun Microsystems, Inc. This information is given for the convenience of users of the present document and does not constitute an endorsement by ETSI of the product named. Equivalent products may be used if they can be shown to lead to the same results".

## C.2.2 OpenDaylight

OpenDaylight (ODL) is a highly available, modular, extensible, scalable and multi-protocol controller infrastructure built for SDN deployments on modern heterogeneous multi-vendor networks. It provides a model-driven service abstraction platform that allows users to write apps that easily work across a wide variety of hardware and southbound protocols [i.13].

Lithium is the third release of OpenDaylight, the open source platform for building programmable, software-defined networks. With Lithium, service providers and enterprises might transition to SDN with particular focus on broadening the programmability of intelligent networks. They might compose their own service architectures or leverage an OpenDaylight-based commercial offering to deliver dynamic network services in a cloud environment, craft dynamic intent-based policies and begin virtualizing functions with Service Function Chaining (SFC).

The OpenDaylight platform provides a common foundation and a robust array of services to enable a wide breadth of applications and use cases. ODL might deliver the benefits of SDN to use cases as diverse as managing cable modems, connecting the Internet of Things, or controlling Ethernet switches using the OpenFlow protocol.

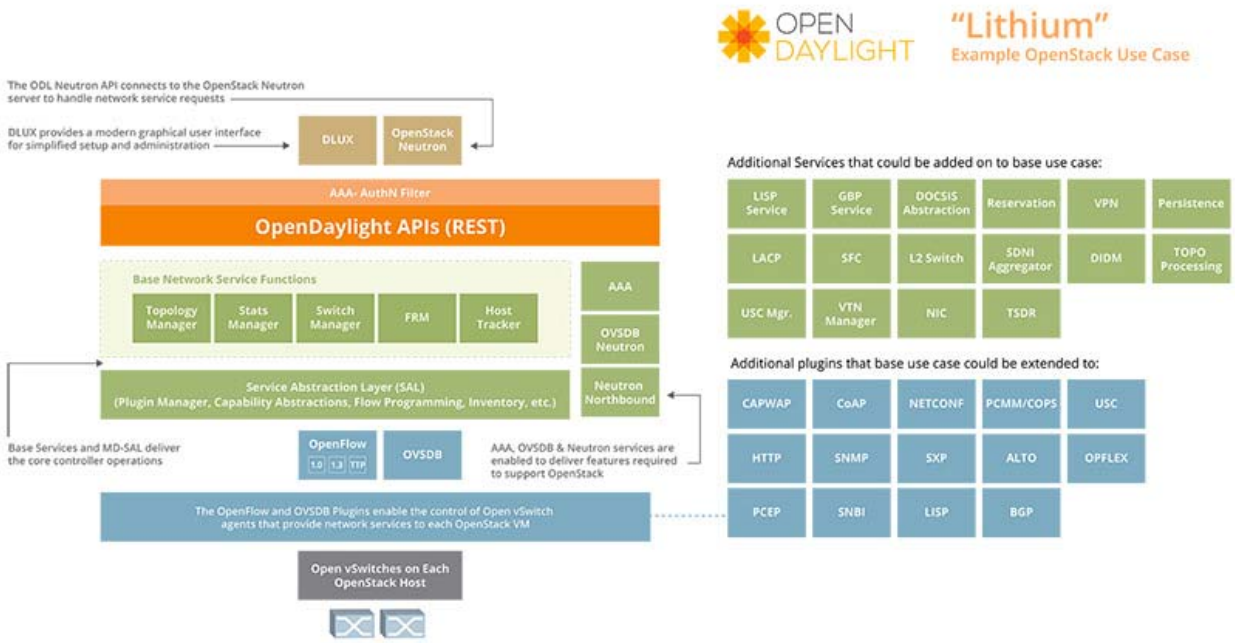



**Figure C.2: OpenDaylight Lithium architecture with Example OpenStack use case**

As shown in figure C.2, OpenDaylight is composed of a number of different modules that might be combined as needed to meet the requirements of a given scenario. Users might download OpenDaylight directly to compose their own service architecture or leverage one of the 20+ available ODL-based commercial offerings.

One common use case explored here is providing network services for cloud data center platforms such as OpenStack. As shown in figure C.2 a standard OpenStack cloud might be paired with OpenDaylight to offload network processing and provide enhanced services. This basic use case might be implemented after starting the core ODL controller by enabling the AAA, Neutron and OVSDB services. These services alone might support OpenStack by automating network Virtualisation and providing centralized control and management of distributed virtual switches and routers across the OpenStack cloud.

With OpenStack within the SDN context, controllers and applications interact using two channels: OpenFlow and OVSDB. OpenFlow addresses the forwarding-side of the OVS functionality. OVSDB, on the other hand, addresses the management-plane. Open vSwitch (OVS) is generally accepted as the unofficial standard for Virtual Switching in the Open hypervisor based solutions.

Other use cases have been defined such as Service Function Chaining and integration with OPNFV.

Some highlights of what is new in Lithium: (more on https://www.opendaylight.org/whats-new-lithium):

- Controller: For Lithium the Controller Project added key new capabilities to improve performance and scalability including MD-SAL clustering along with additional enhancements and API Extensions.
- AAA: Key Updates for Lithium: Added support for persistent data stores, Federation and SSO with OpenStack Keystone, and additional extensions and enhancements.
- ALTO: Application-Layer Traffic Optimization (ALTO) is an IETF protocol to provide network information to applications. The ALTO project in OpenDayLight is an effort to implement ALTO in OpenDayLight. In addition to implementing the ALTO base protocol (IETF RFC 7285 [i.20]), the project leverages OpenDayLight to introduce a provisioning interface for ALTO.

- BGP/LS/PCEP: This project is an effort to bring two south-bound plugins into the controller: one for supporting BGP Linkstate Distribution as a source of L3 topology information, the other one to add support for Path Computation Element Protocol (PCEP) as a way to instantiate paths into the underlying network. Key Updates for Lithium: Added support for BGP Flowspec, graceful restart, segment routing, secure transport for PCEP.

- GBP: The group-based policy (GBP) project defines an application-centric policy model for OpenDaylight that separates information about application connectivity requirements from information about the underlying details of the network infrastructure. Key Updates for Lithium: added support for integration with OpenStack Neutron; support for Service Function Chaining; one renderer called OpenFlow Overlay Renderer (OfOverlay) with NAT support, table offsets; overall improvements in code quality, stability and performance.

- IoT: The IoTDM project is about developing a data-centric middleware that will act as a oneM2M compliant IoT Data Broker (IOTDM) and enable authorized applications to retrieve IoT data uploaded by any device.

- LACP: The LACP Project within OpenDaylight implements Link Aggregation Control Protocol (LACP) as an MD-SAL service module and will be used to auto-discover and aggregate multiple links between an OpenDaylight controlled network and LACP-enabled endpoints or switches.

- LISP flow mapping service: LISP (IETF RFC 6830 [i.19]) is a protocol that enables separation of Endpoint Identity (EID) from Routing Location (RLOC) by defining an overlay in the EID space which is mapped to the underlying network in the RLOC space. LISP also provides a Mapping Service that provides the [i.20] EID-to-RLOC mapping information including forwarding policy (load balancing, traffic engineering, etc.) to LISP routers for tunneling and forwarding purposes. The LISP Mapping Service might serve the mapping data to dataplane nodes as well as to ODL applications. To leverage this service, a northbound API allows OpenDaylight applications and services to define the mappings and policies in the LISP Mapping Service. This project also includes a southbound LISP plugin that enables LISP dataplane devices to interact with the OpenDaylight via the LISP protocol. Key Updates for Lithium: Improved ELP Processing, NB API transitioned to MD-SAL, Continuous integration with Service Function Chaining, Neutron enhancements.

- NIC: Network Intent Composition (NIC) is an interface that allows clients to express a desired state in an implementation-neutral form that will be enforced via modification of available resources under the control of the OpenDaylight system.

- Neutron and OVSB plugin: Several services and plugins in opendaylight work together to provide simplified integration with the OpenStack Neutron framework. These services enable OpenStack to offload network processing to ODL, while also enabling ODL to provide enhanced network services to OpenStack. Key Updates for Lithium: Transitioned API's to MD-SAL, increased feature parity with Neutron including support for LBaaS, added Distributed Virtual Router, SNAT, External Gateway & Floating IP support, improvements to performance and stability.

- Openflow plugin: The plugin is based on the Model Driven Service Abstraction Layer (MD-SAL) architecture. Key Updates for Lithium: For Lithium the OpenFlow Plugin underwent a complete rearchitecture, this new OpenFlow 1.0/1.3 MD-SAL based plugin is distinct from the old OpenFlow 1.0 plugin which was based on the API driven SAL (AD-SAL) architecture; support for Table Type Patterns added.

- Opflex: The ODL Opflex Agent is a policy agent that works with OVS to enforce a group-based policy networking model with locally attached virtual machines or containers.

- Reservation: Reservation is meant to provide dynamic low level resource reservation so that users might get network as a service, connectivity or a pool of resources (ports, bandwidth) for a specific period of time. The idea behind the reservation application is to be able to have end to end multi-layer provisioning. Assume the two end to be two ports in a device. The link provisioning could be done based on information such as latency, bandwidth, etc. Link reservation also might consider the period of time (duration), and taking into account the cost of reservation.

- SFC: OpenDaylight Service Function Chaining (SFC) provides the ability to define an ordered list of a network services (e.g. firewalls, load balancers). These service are then "stitched" together in the network to create a service chain. This project provides the infrastructure (chaining logic, APIs) needed for ODL to provision a service chain in the network and an end-user application for defining such chains. Key Updates for Lithium: Added Yang models for expressing service function chains; SFC receiver for intent expressions from REST and RPC; UI for service chain construction; LISP support; function grouping for load balancing; Openflow renderer for NSH, MPLS and VLAN; Southbound REST interface; SFC-OVS for SFF bridges, tunnels, and ports CRUD via OVSDB; Classifier (IP Tables based) for classifying packets into selected Service Chains; Integration with ODL GBP project.

- SNBI: SNBI uses a zero-touch approach to bootstrapping that leverages manufacturer-installed IEEE 802.1AR certificates to secure even the initial communications. SNBI devices and controllers automatically discover each other, get an IP-address assigned, and establish secure IP connectivity. In addition, this discovery process reveals the physical topology of the network, exposes each type of a device, and assigns the domain for each device. Key Updates for Lithium: Added Linux side abstraction layer for forwarding elements along with enhancements to feature abstraction and bootstrapping infrastructure.

- Topology Processing Service (TPS): Provides a framework for simplified aggregation and querying of topology data which might be used to enable a unified topology view including multi-protocol, Underlay and Overlay resources.

- VPN: Virtual Private network (VPN) key updates with Lithium: API for L3 VPN Services; Integration with open source routing suites including Quagga & RYU; OpenStack Neutron integration.

- VTN: Virtual Tenant Network (VTN) migration to MD-SAL; Significant improvements to policy management in VTN Manager.

- USC: The Unified Secured Channel (USC) framework provides a central server to coordinate encrypted communications between endpoints.

OpenDaylight Lithium Deployment: deploying OpenDaylight has some pre-requisites.

The OpenDaylight Karaf container, OSGi bundles, and Java class files are portable run on any Java 7- or Java 8-compliant JVM to run. Certain projects and certain features of some projects might have additional requirements. Those are noted in the project-specific release notes.

Projects and features which have known additional requirements are:

- TCP-MD5 requires 64-bit Linux

- TSDR has extended requirements for external databases

- Persistence has extended requirements for external databases

- SFC requires addition features for certain configurations

- SXP depends on TCP-MD5 on thus requires 64-bit Linux

- SNBI has requirements for Linux and Docker * OpFlex requires Linux

- DLUX requires a modern web browser to view the UI

- AAA when using federation has additional requirements for external tools

- VTN has components which require Linux

Typically SNBI pre-requisites are: 64 bit Ubunutu® (see note) 14.04 LTS Especially for Docker, its required that be deployed docker version greater than 1.0 on a 14.04 Ubuntu. 4 GB RAM 4 GB of Hard disk space, sufficient enough to store certificates. Java Virtual Machine 1.7 or above Apache Maven 3.04 or above.

NOTE: "Ubuntu is the trade name of a product supplied by CANONICAL Ltd. This information is given for the convenience of users of the present document and does not constitute an endorsement by ETSI of the product named. Equivalent products may be used if they can be shown to lead to the same results".

## C.2.3 OpenContrail

OpenContrail [i.14] is an Apache 2.0-licensed project that is built using standards-based protocols and provides all the necessary components for network Virtualisation-SDN controller, virtual router, analytics engine and published northbound APIs. As shown in the following subsections it has an extensive REST API to configure and gather operational and analytics data from the system. Built for scale, OpenContrail might act as a fundamental network platform for cloud infrastructure. The key aspects of the system are:

**Network Virtualisation:** Virtual networks are the basic building block of the OpenContrail approach. Access-control, services and connectivity are defined via high level policies. By implementing inter-network routing in the host, OpenContrail reduces latency for traffic crossing virtual-networks. Eliminating intermediate gateways also improves resiliency and minimizes complexity.

**Network Programmability and Automation:** OpenContrail uses a well-defined data model to describe the desired state of the network. It then translates that information into configuration needed by each control node and virtual router. By defining the configuration of the network versus of a specific device, OpenContrail simplifies and automates network configuration.

**Big Data for Infrastructure:** The analytics engine is designed for very large scale ingestion and querying of structured and unstructured data. Real-time and historical data is available via a simple REST API, providing visibility over a wide variety of information.

OpenContrail might forward traffic within and between virtual networks without traversing a gateway. It supports features such as IP address management; policy-based access control; NAT and traffic monitoring. It interoperates directly with any network platform that supports the existing BGP/MPLS L3VPN standard for network Virtualisation.

OpenContrail might use most standard router platforms as gateways to external networks and might easily fit into legacy network environments. OpenContrail is modular and integrates into open cloud computing platforms such as OpenStack, Cloudstack, and is currently supported across multiple Linux distributions and hypervisors.

The Contrail Controller is based on the Border Gateway Protocol (BGP) that is already embedded in widely deployed IP infrastructures, (switches and routers from leading network vendors).

It also employs XMPP, a protocol for transmitting message-oriented middleware messages, to control the virtual switches inside of hypervisors. It uses Multiprotocol Label Switching (MPLS), which encapsulates packets on a network and controls their forwarding through those labels; MLPS exists between Layers 2 and 3 in the network stack.

There are a number of other protocols that the Contrail Controller uses to separate the data and control planes in the switching stack and making them more malleable, but at the moment the OpenFlow protocols are not supported.

**OpenContrail Architecture Overview**

As shown below in figure C.3, the OpenContrail System consists of two parts: a logically centralized but physically distributed controller and a set of vRouters that serve as software forwarding elements implemented in the hypervisors of general purpose Virtualised servers.

The controller provides northbound REST APIs used by applications. These APIs are used for integration with the cloud computing system, for example for integration with OpenStack via a neutron plugin. The REST APIs might also be used by other applications and/or by the operator's OSS/BSS. Finally, the REST APIs are used to implement the web-based GUI included in the OpenContrail System.

The OpenContrail System provides three interfaces: a set of north-bound REST API's that are used to talk to the Applications, south-bound interfaces that are used to talk to virtual network elements (vRouters) or physical network elements (gateway routers and switches), and an east-west interface used to peer with other controllers. OpenStack and CloudStack are supported via standard BGP (the east-west interface), XMPP is the southbound interface for vRouters, BGP and Netconf and the southbound interfaces for gateway routers and switches.

Internally, the controller consists of three main components:

- Configuration nodes which are responsible for translating the high-level data model into a lower level form suitable for interacting with network elements.
- Control nodes which are responsible for propagating this low level state to and from network elements and peer systems in an eventually consistent way.

- Analytics nodes which are responsible for capturing real-time data from network elements, abstracting it and presenting it in a form suitable for applications to consume.

All of these nodes will be described in detail later in this clause.

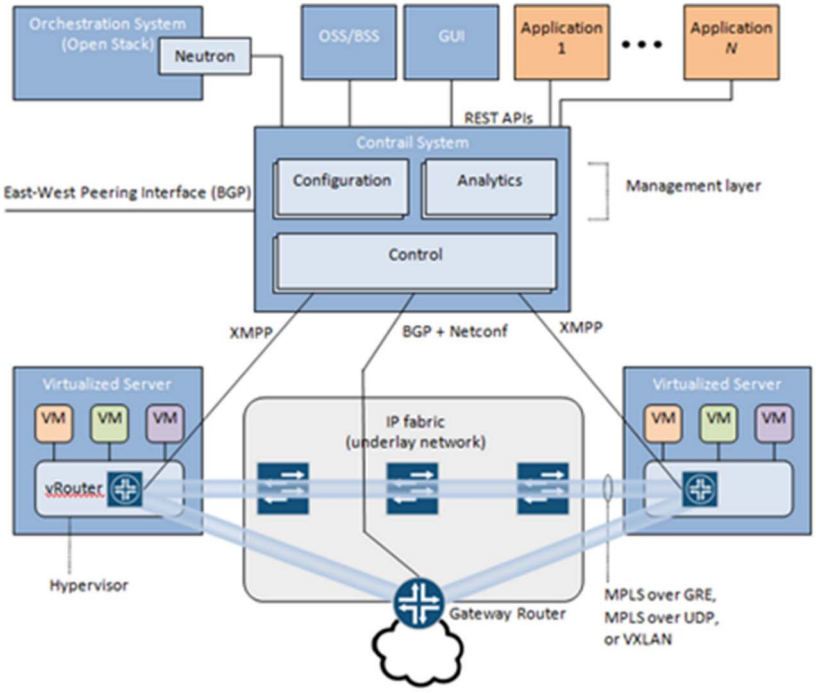



**Figure C.3: OpenContrail SDN controller system overview**

The vRouters are network elements implemented entirely in software. They are responsible for forwarding packets from one virtual machine to other virtual machines via a set of server-to-server tunnels. The tunnels form an overlay network sitting on top of a physical IP-over-Ethernet network. Each vRouter consists of two parts: a user space agent that implements the control plane and a kernel module that implements the forwarding engine.

The OpenContrail System implements three basic building blocks:

1) physical or virtual network services such as firewalls, Deep Packet Inspection (DPI), or load balancers. Multi-tenancy, also known as network Virtualisation or network slicing is the ability to create Virtual Networks that provide Closed User Groups (CUGs) to sets of VMs.

2) Gateway functions: this is the ability to connect virtual networks to physical networks via a gateway router (e.g. the Internet), and the ability to attach a non-Virtualised server or networking service to a virtual network via a gateway.

3) Service chaining as an NFV use case: this is the ability to steer flows of traffic through a sequence of physical or virtual network services such as firewalls, Deep Packet Inspection (DPI) or load balancers.

**OpenContrail Nodes**

Looking at the internal structure of the system, as shown in figure C.4, the system is implemented as a cooperating set of nodes running on general-purpose x86 servers. Each node might be implemented as a separate physical server or it might be implemented as a Virtual Machine (VM).

All nodes of a given type run in an active-active configuration so no single node is a bottleneck. This scale out design provides both redundancy and horizontal scalability.

- Configuration nodes keep a persistent copy of the intended configuration state and translate the high-level data model into the lower level model suitable for interacting with network elements. Both these are kept in a NoSQL database.

- Control nodes implement a logically centralized control plane that is responsible for maintaining ephemeral network state. Control nodes interact with each other and with network elements to ensure that network state is eventually consistent.

- Analytics nodes collect, store, correlate, and analyse information from network elements, virtual or physical. This information includes statistics, logs, events and errors.

In addition to the node types which are part of the OpenContrail Controller, some additional nodes types have been identified for physical servers and physical network elements performing particular roles in the overall OpenContrail System:

- Compute nodes are general-virtualizedalised servers which host VMs. These VMs might be tenant running general applications, or these VMs might be service VMs running network services such as a virtual load balancer or virtual firewall. Each compute node contains a vRouter that implements the forwarding plane and the distributed part of the control plane.

- Gateway nodes are physical gateway routers or switches that connect the tenant virtual networks to physical networks such as the Internet, a customer VPN, another data center, or to non-Virtualised servers.

- Service nodes physical network elements providing network services such as Deep Packet Inspection (DPI), Intrusion Detection (IDP), Intrusion Prevention (IPS), WAN optimizers, and load balancers. Service chains might contain a mixture of virtual services (implemented as VMs on compute nodes) and physical services (hosted on service nodes).

For clarity, the figure does not show physical routers and switches that form the underlay IP over Ethernet network. There is also an interface from every node in the system to the analytics nodes. This interface is not shown in figure C.4 to avoid clutter.

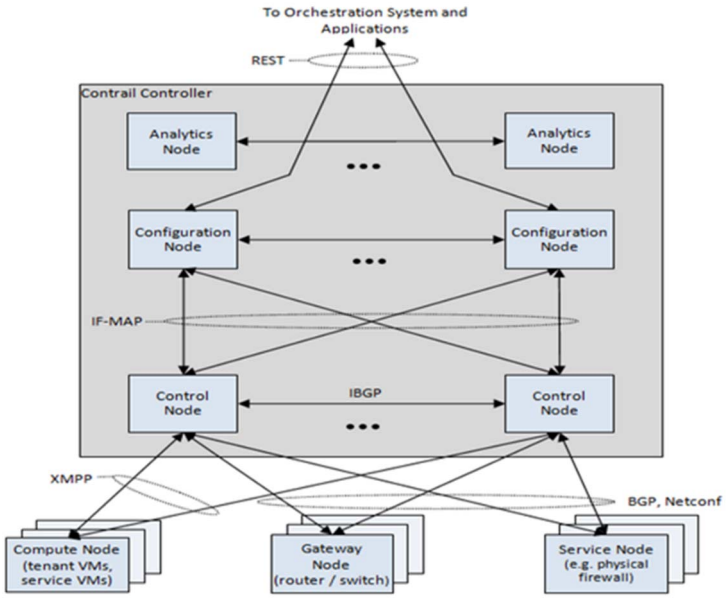



**Figure C.4: OpenContrail SDN Controller system implementation**

## C.2.4 Ryu

Ryu [i.16] is a component-based software defined networking framework as shown in figure C.5. Ryu provides software components with well-defined API that make it easy for developers to create new network management and control applications. Ryu supports various protocols for managing network devices, such as OpenFlow, Netconf, OF-config, etc. About OpenFlow, Ryu supports fully 1.0, 1.2, 1.3, 1.4 and Nicira Extensions. All of the code is in Python and is freely available under the Apache 2.0 license. Ryu means "flow" in Japanese. Ryu is pronounced "re"-yooh".

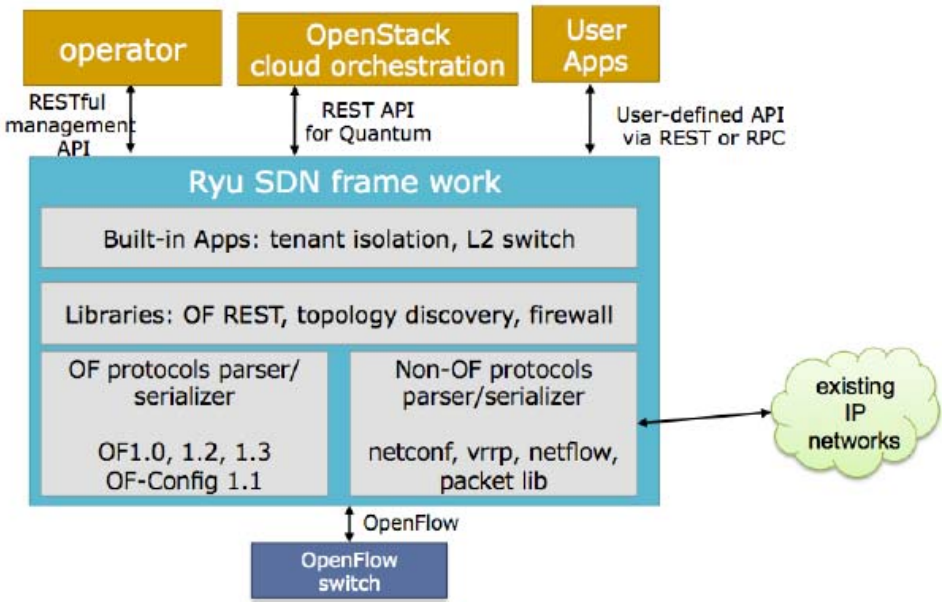



**Figure C.5: Ryu SDN Controller architecture overview**

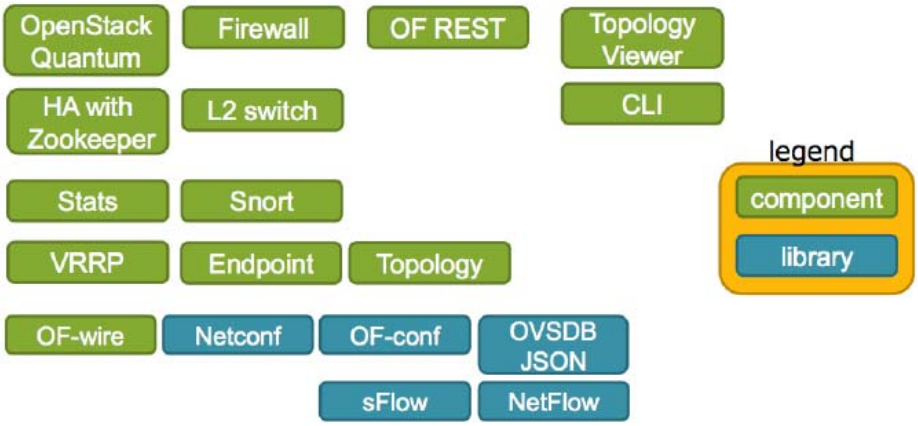



**Figure C.6: Ryu SDN controller software components**

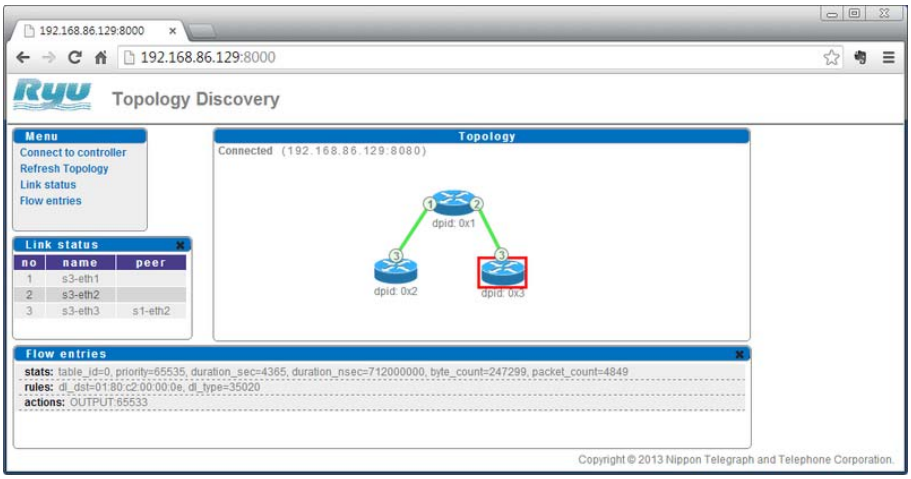



**Figure C.7: Ryu SDN Controller user interface and topology discovery**

## C.2.5 ONOS

### C.2.5.1 Introduction

ONOS [i.15] release considered here is Blackbird. ONOS releases quarterly in February, May, August and November.

SDN Operating System released by on.lab, www.onlab.us. On.lab is a non-profit 501 organization.

ONOS is a multi-module project whose modules are managed as OSGi bundles. ONOS is designed with a few goals in mind:

- Code Modularity: It is be possible to introduce new functionalities as self-contained units.
- Configurability: It is be possible to load and unload various features, whether it be at startup or at runtime.
- Separation of Concern: There are clear boundaries between subsystems to facilitate modularity.
- Protocol agnosticism: It, and its applications, are not be bound to specific protocol libraries or implementations.

Looking at each below.

**Code Modularity**

The project is comprised of a set of sub-projects, each with their own source tree that might be built independently. To do this, the ONOS source tree is organized in a hierarchical fashion that takes advantage of Maven's notion of a hierarchical POM file organization. Each sub-project has its own *pom.xml* file, and intermediate directories have parent aggregate *pom.xml* files. The latter contains shared dependencies and configurations for those sub-projects, enabling them to be built independent of unrelated sub-projects. The ONOS root contains the top-level POM (Project Object Model) file used to build the full project and all of its modules.

For implementation details of the source tree organization, refer to Appendix C of the Developer's Guide [i.24].

**Configurability**

ONOS is written to leverage Apache Karaf as its OSGi framework. In addition to dependency resolution at startup and dynamic module loading at runtime, Karaf provides the following:

- Enable use of standard JAX-RS API to develop REST APIs and make them secure
- The notion of features as a set of bundles allowing assembly of custom setups
- Strict semantic versioning of code bundles, including third-party dependencies
- Local and remote ssh console with easily extensible CLI
- The notion of run-time log levels

**Separation of Concern**

ONOS is partitioned into:

- Protocol-aware network-facing modules that interact with the network;
- Protocol-agnostic system core that tracks and serves information about network state; and
- Applications that consume and act upon the information provided by the core.

Each of the above are tiers in a layered architecture, where network-facing modules interact with the core via a *southbound (provider) API*, and the core with the applications via the *northbound (consumer) API*. The southbound API defines protocol-neutral means to relay network state information to the core, and for the core to interact with the network via the network-facing modules. The northbound API provides applications with abstractions that describe network components and properties, so that they may define their desired actions in terms of policy instead of mechanism.

**Protocol Agnosticism**

If ONOS needs to support a new protocol, it is possible to build a new network-facing module against the southbound API as a plugin that might be loaded into the system.

Figure C.8 illustrates ONOS SDN controller architecture overview.

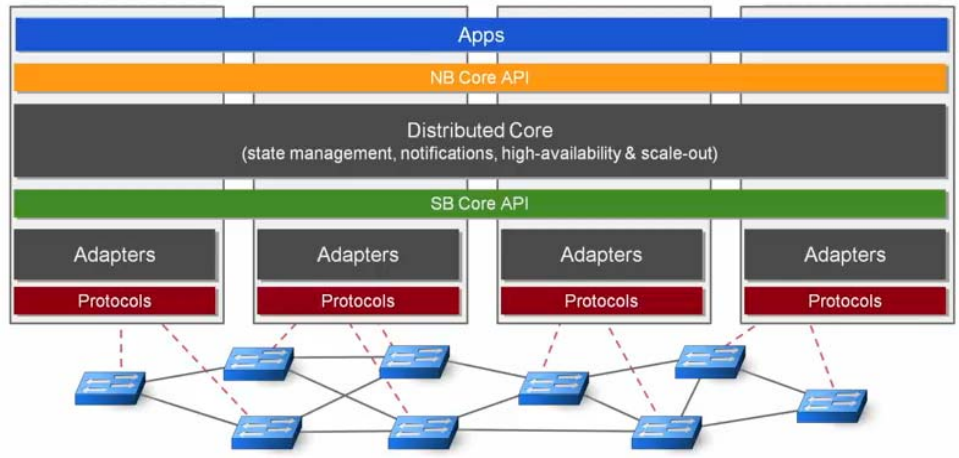



**Figure C.8: ONOS SDN Controller architecture overview**

## C.2.5.2 ONOS Architecture

ONOS is architected as a logically centralized but physically distributed SDN control plane. ONOS comprises a cluster of instances that work together to manage the applications and the network. As demands on the SDN control plane grow, either due to an increase in the size of the network or due to an increase in the number of network control applications, ONOS might scale by adding additional instances to the cluster.
ONOS automatically offloads a portion of the work to these new instances. Architecting a distributed but logically centralized SDN control plane such as ONOS is a true technical challenge. ONOS' ability to successfully provide and demonstrate high performance, scale and high availability together is what differentiates it from other open source SDN controllers available today. ONOS measurements that validate the benefits of its distributed architecture including providing high performance and scalability are highlighted in figure C.9.

The ONOS Blackbird release defines the following set of metrics to effectively measure performance and other carrier grade attributes of the SDN control plane.

Performance Metrics:

- Topology - Switch change latency
- Topology - Link change latency
- Flow installation throughput
- Intent (Northbound) install latency
- Intent (Northbound) withdraw latency
- Intent (Northbound) reroute latency
- Intent (Northbound) throughput

Target numbers for metrics

- Flow Throughput: 1 Million flow operations/second
- Latency: Less than 100 ms latency (ideally under 10 ms)

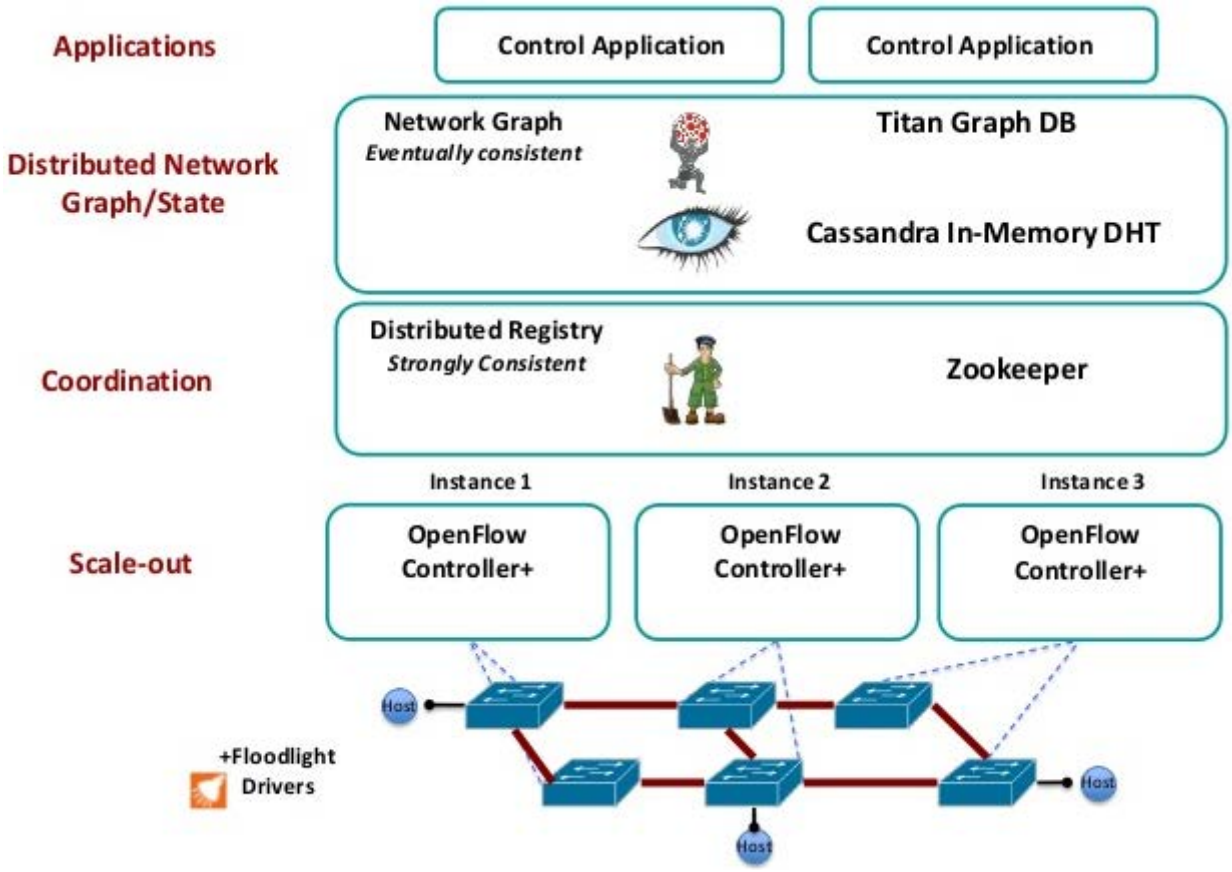



**Figure C.9: ONOS SDN Controller software architecture**

## C.2.5.3 ONOS User Interface

The ONOS user interface as described in figures C.12 and C.13 enables:

- IP packet and optical view
- Topology view

The layout is automatically done based on GPS information from the network elements.

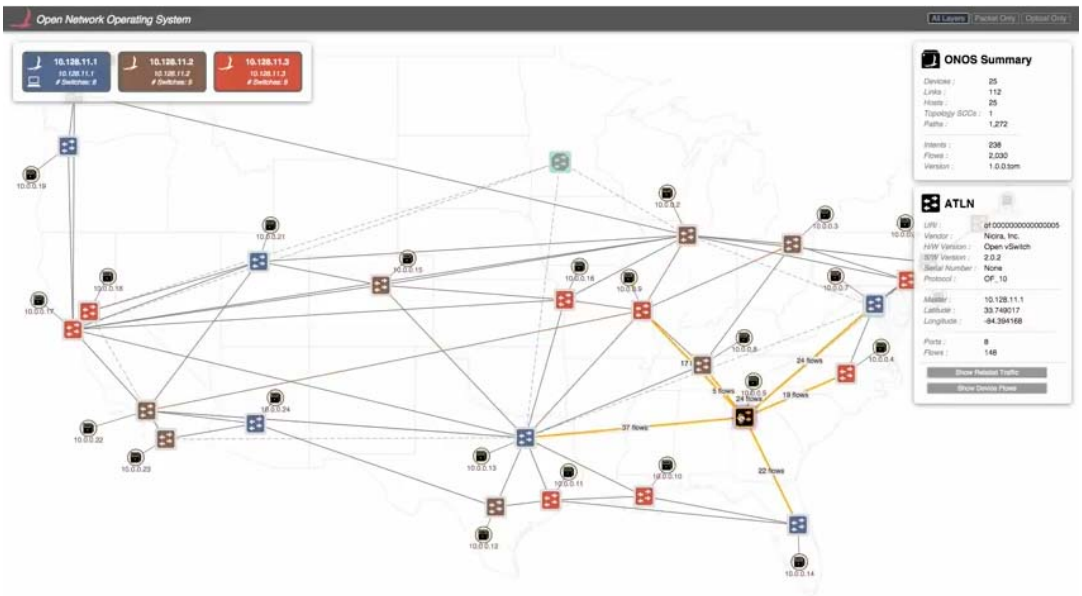



**Figure C.10: ONOS IP Packet and Optical View**

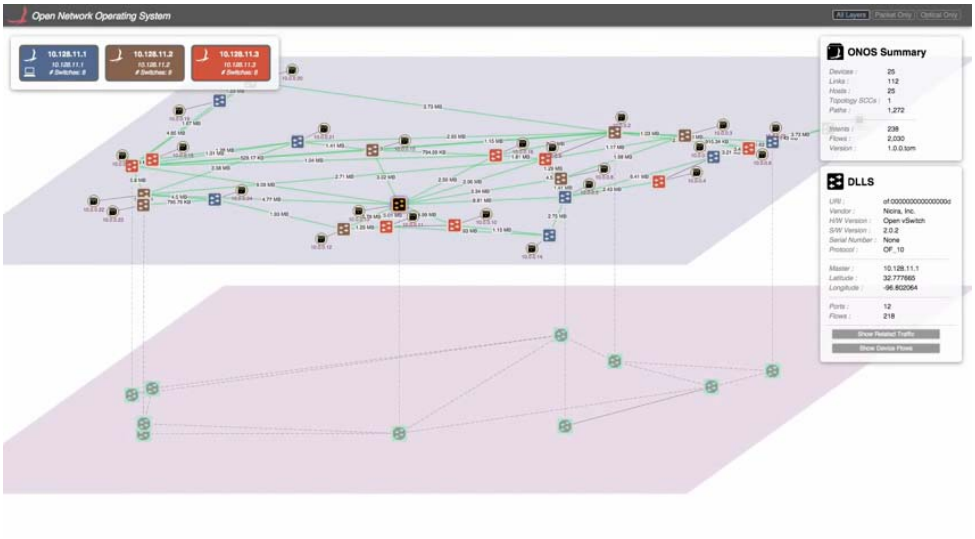



**Figure C.11: ONOS Topology user interface**

## C.2.6 MidoNet

MidoNet is a distributed, de-centralized, software-defined virtual network platform for Infrastructure as a Service (IaaS) as shown in figure C.12.

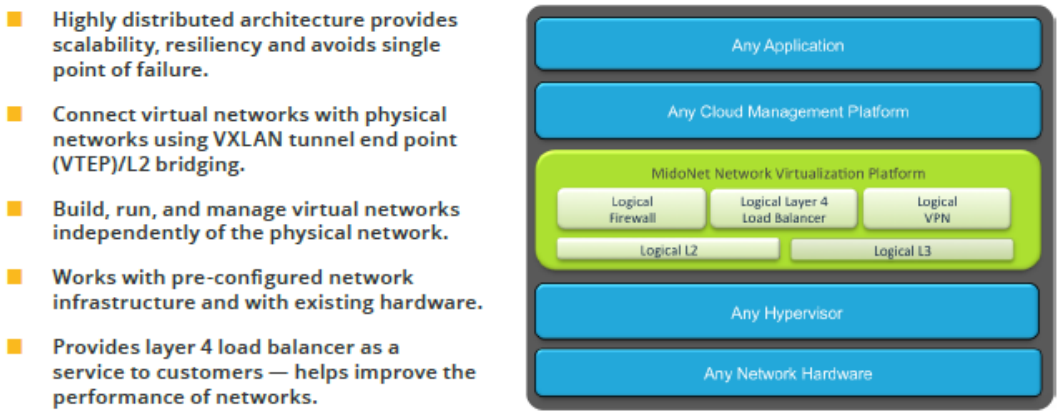



**Figure C.12: Midonet SDN controller architecture overview**

Midonet [i.17] was released as opensource end of 2013 by a Japanese startup company.

MidoNet virtualizes the network functionality for IaaS products, such as OpenStack, providing functionally advanced, robust, scalable, and secure networks. MidoNet is an overlay network that runs software on standard x86 servers, and sits on top of any scalable network underlay (for example, physical servers and switches), pushing the intelligent network functions to the edge of the network, in software, as shown in figure C.13.

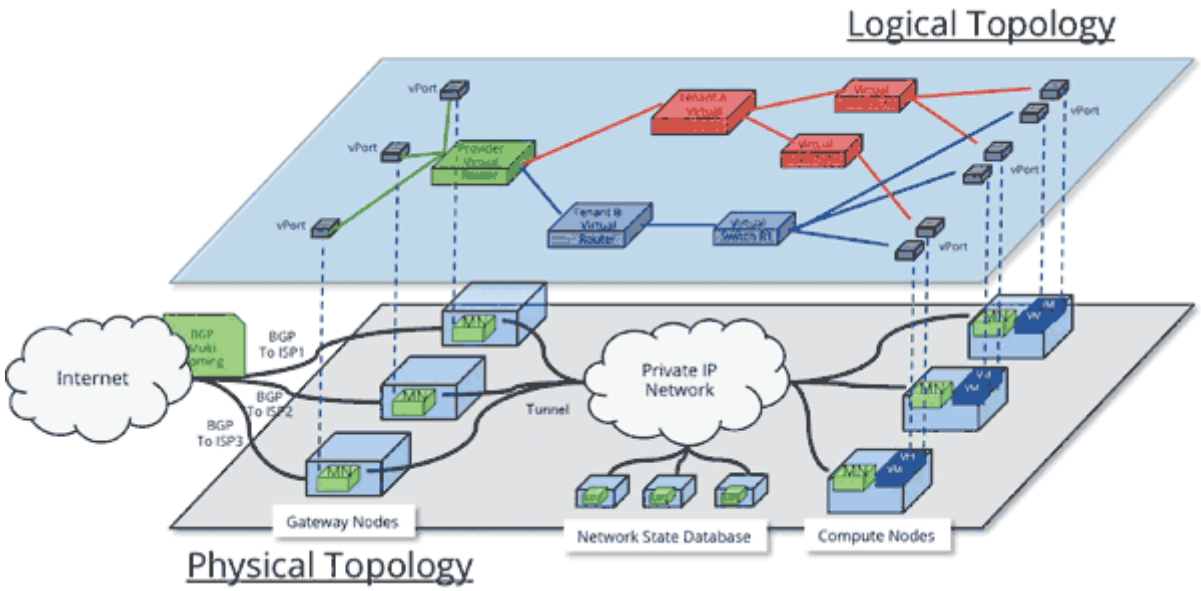



**Figure C.13: Midonet SDN controller deployment model**

Supports Ubuntu & RHEL® (see note).

NOTE: "RHEL is a registered trademark of Red Hat, Inc. This information is given for the convenience of users of the present document and does not constitute an endorsement by ETSI of the product named. Equivalent products may be used if they can be shown to lead to the same results".

These are the key features of MidoNet:

- Layer 2 through 4 networking
- VLAN-less VLANs - Virtual L2 distributed isolation and switching with virtually none of the limitations of conventional VLANs
- Fully distributed architecture with no single points of failure
- Virtual L3 distributed routing
- Distributed load balancing and firewall services
- Stateful and stateless NAT
- Access Control Lists (ACLs)
- Restful API
- Monitoring of networking services
- VXLAN support: VXLAN tunnel zones, VXLAN L2 Gateway
- Zero-delay NAT connection tracking

Member companies: there are about 20 member companies, all vendors.

**Table C.2: Midonet Versions available**

| Series | Status | Releases | Date |
|---|---|---|---|
| version 2015.4 | Under development | Due | Apr 30, 2015 |
| version 2015.3 | Under development | Due | Mar 30, 2015 |
| version 2015.2 | Under development | Due | Feb 27, 2014 |

# C.3 Comparison criteria

## C.3.1 List of Comparison Criteria

**Table C.3: list of comparison criteria**

| Criteria | Answer #1 | Other answer | Comment |
|---|---|---|---|
| OS | Linux | | Which Linux |
| Packages | Debian | other | |
| Language written in | C | Python others | |
| Protocols<br>Openflow<br>Netconf<br>BGP<br>OVSDB<br>LISP<br>VXLAN<br>GRE<br>etc. | Openflow 1.4 or prior | Other | List of supported southbound protocol |
| Openstack integration | Yes, version, API | No | + comment on level of integration |
| Cloudstack integration | Yes, version, API | | |
| North bound API | REST | Other | Northbound API Protocol and other details |
| Licence | Apache 2.0 | Other | Opensource licence |
| Virtualised | Yes | No | |
| Applications | Yes, which ones | No | |
| ETSI NFV POC | Yes | No | |
| Docker support | Yes | No | |

NOTE: Latest Openflow release is 1.4.

- OS: most common OS is Linux
- Language written: most common language is C, C++ or Python
- Protocol: most common protocol is Openflow, but others are supported too
  - Openflow: protocol between data plane and control plane defined by ONF (Open Networking Foundation)
  - OVSDB: The Open vSwitch Database Management Protocol (OVSDB) is an OpenFlow configuration protocol that is designed to manage Open vSwitch implementations.

Open vSwitch is a virtual switch that enables network automation, while supporting standard management interfaces and protocols, like NetFlow. Open vSwitch also supports distribution across multiple physical servers.

In an Open vSwitch implementation, a database server and a switch daemon are used. The OVSDB protocol is used in a control cluster, along with other managers and controllers, to supply configuration information to the switch database server. Controllers use OpenFlow to identify details of the packet flows through the switch. Each switch might receive directions from multiple managers and controllers, and each manager and controller might direct multiple switches.

Using the OVSDB protocol, IT professionals might determine the number of individual virtual bridges within an Open vSwitch implementation, allowing a network engineer to create, configure and delete ports and tunnels from a bridge. Engineers might also create, configure and delete queues:

- VXLAN
- GRE
- BGP
- LISP
- etc.

Licence: most common and flexible licence is the permissive Apache 2.0 license – this essentially means that anyone might deploy and modify the System code without any obligation to publish or release the code modifications.

## C.3.2 Impact of Docker

Docker is a way to rollout applications within Linux containers and manage their configurations in a portable fashion. While the LXC technology does not provide the resource level isolation that virtual machines (VM) had with the hypervisor, it provides some level of isolation.

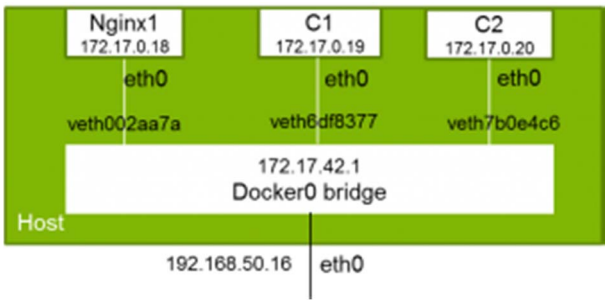



**Figure C.14: Docker architecture overview**

Today, Docker picks Linux bridge as the default network for each spawned container and allocates a static IP address from a default range. The container traffic is then NAT'ed through the host network. This is similar to spawning VMs with KVM or VirtualBox on your laptop. As container usage gets more complicated to span multiple hosts, multiple application tiers and multiple datacenters, good underlying plumbing and well understood abstractions are needed.

It is certainly possible to use same networking principles as was used with VMs, or adopt a clean slate thinking for container communication. In the VM world, each individual entity was addressable by a MAC/IP address combination. In the container world, however, it is possible that the containers are not individually addressable, and share the host network or use other means for communicating (e.g. Unix domain sockets).

While the technology and mode of plumbing is up for debate, some of the abstractions will still work in the container space. The foremost of that is network Virtualisation, the act of decoupling the application's virtual network from the underlying physical network. This abstraction brings in modularity in deployment, portability of groups of containers, access control across application boundaries, and plumbing for virtual network services. This is needed in the container world too (source ONOS wiki).

# C.4 Comparison table

**Table C.4**

| Criteria | OpenDaylight Lithium June 2015 | OpenContrail 1.2 | ONOS Blackbird | Ryu | Midonet (MEM) May 2015 |
|---|---|---|---|---|---|
| OS | Any operating system and hardware as long as it supports Java | Linux, centOS and Ubuntu | Any operating system and hardware as long as it supports Java 8 | | Linux, Ubuntu 14.04, RHEL |
| Language written in | Java - requires Java 7 or 8 | Python | Java | Python | |
| Openflow | Yes 1.3 and before | No | Yes 1.3+ and prior | Yes 1.4 and prior | |
| Netconf | Yes | Yes | Yes with 1.2.0 Cardinal | Yes | |
| OVSDB | Yes | | 1.3.0 Release | Yes | |
| LISP | Yes | | | | |
| BGP | Yes | Yes | Yes (SDN-IP, BGP Routing) | Yes | |
| XMPP | | Yes | | | |
| MPLS | Yes (VPN Service) | Yes, MPLS over GRE | Yes (Segment Routing) | | |
| VXLAN | Yes (OVSDB) | yes | 1.3.0 Release | Yes(OVSDB) | |
| GRE | Yes (OVSDB) | Yes | 1.3.0 Release | | |
| Hypervisor support | KVM | Xen, KVM | | kVM, Vmware, Xen | |
| Openstack integration | Yes with OpenFlow & OVSB | Yes Icehouse, NEUTRON API | Yes via OVX | Yes | yes |
| CloudStack | | yes | | | |
| Mgt tools integration | | Chef, puppet, cobbler, ganglia support | CLI, REST, shell | | |
| North bound API | REST | | REST + Java | | REST |
| Licence | EPL-1.0 | Apache 2.0 | Apache 2.0 | Apache 2.0 | Apache 2.0 |
| Virtualised | | | | | |
| Applications | Many, also called services<br>DLUX<br>VTN Coordinator<br>OpenStack Neutron<br>SDNI Wrapper<br>DdoS Protection | | • SDN-IP<br>• BGP Routing<br>• Segment Routing<br>• L2/L3 Forwarding (Ipv4 + Ipv6)<br>• IP/Optical<br>• ProxyARP<br>• Mobility<br>• CORD (1.2.0) | | |
| ETSI NFV POC | Yes | | OP-NFV Huawei PoC | | |
| Docker support | Yes i.e. SBNI | yes | | | |
| Ipv6 support | Yes | no | Yes 1.2.0 Release | | |
| TLS security | Yes | | | | |
| SSl security | Yes | | | | |
| Intent | yes | | Yes | | |

# C.5 ETSI NFV POC with Opensource SDN Controller

## C.5.1 List of POC and Opensource Controller used

Here is the list of ETSI NFV POC with Opensource SDN Controller.

**Table C.5**

| ETSI POC | SDN NE | SDN Controller | Comment |
|---|---|---|---|
| POC#1 | | | |
| POC#2 | OpenFlow | RYU | |
| POC#8 | OpenFlow | OpenDaylight | |
| POC#13 | OpenFlow | Vendor SDN controller | |
| POC#15 | OpenFlow | OpenDaylight | |
| POC#16 | OF 1.3 | RYU | |
| POC#21 | OpenFlow | Floodlight (POF) | |
| POC#23 | OpenFlow | Proprietary controller | |
| POC#26 | OpenFlow | Ryu | |
| POC#27 | OpenFlow, OF-epc | | |
| POC#28 | OpenFlow | Vendor SDN controller | |
| POC#34 | OpenFlow | OpenDaylight | With extensions for GTP |
| POC#38 | OpenFlow | Vendor SDN controller | |

## C.6 Lessons learnt

Following the comparison of OpenSource SDN controller and based on the experience of the POC using SDN controllers, including some of these Opensource software, a few lessons were learnt:

- Opensource SDN controller follow the ITU model with 3 layers: a layer of resources, a layer of controllers and a layer of applications.
- Most Opensource SDN controller provide some applications embedded inside the SDN controller , while they support SDN applications via an open interface , generally REST or java.
- All Opensource SDN controllers studied support OpenFlow, and this happens to be the most common protocol used between SDN controller and SDN resources within ETSI NFV POC using SDN.
- Most Opensource SDN controller support multiple protocols.
- Most Opensource SDN controller support hierarchy of SDN controller.
- Most Opensource SDN controller support federation of SDN controller.
- Some Opensource SDN controller provide a dedicated interface for federation, called East-West. This interface is generally not supporting communication with other type of Opensource SDN controller.
- While most Opensource SDN controller agree that they support Virtualisation, deploying a given Opensource SDN controller in an NFV architectural framework is not documented: no descriptors, no VNF lifecycle manager, no description of lifecycle management in NFV environment etc.
- All Opensource SDN controller support integration with OpenStack Neutron, however lots of ETSI NFV POC used direct access to the Opensource SDN controller interface because the neutron interface was not exposing the features needed in the POC, e.g. POC#26 vMME used SDN controller for control plane and HW for data plane, and used direct access between the two for the SDN controller to access directly information in the packet headers for dynamic packet forwarding.
- Most Opensource SDN controller do not provide performance data: number of SDN resources supported, latency on the interfaces, number of simultaneous access on the Application Control Interface, etc.

# C.7 Other Opensource SDN components

## C.7.1 Switch Software and Stand-Alone OpenFlow Stacks

**Open vSwitch**: (C/Python) Open vSwitch is an OpenFlow stack that is used both as a vswitch in Virtualised environments and has been ported to multiple hardware platforms. It is now part of the Linux kernel (as of clause 3.3).

**OpenFlow Reference** (C): The OpenFlow reference implementation is a minimal OpenFlow stack that tracks the spec.
**Indigo**: (C) Indigo is a for-hardware-switching OpenFlow implementation based on the Stanford reference implementation.

**Pantou**: (C) Pantou is an OpenFlow port to the OpenWRT wireless environment.

**OpenFaucet**: (Python) OpenFaucet is a pure Python implementation of the OpenFlow 1.0.0 protocol, based on Twisted. OpenFaucet might be used to implement both switches and controllers in Python.

**OpenFlowJ**: (Java) OpenFlow stack written in Java.

**Oflib-node**: (Javascript) Oflib-node is an OpenFlow protocol library for Node. It converts between OpenFlow wire protocol messages and Javascript objects.

**Nettle**: (Haskell) OpenFlow library written in Haskell.

## C.7.2 Controller Platforms

**POX**: (Python) Pox as a general SDN controller that supports OpenFlow. It has a high-level SDN API including a queriable topology graph and support for Virtualisation.

**OpenIRIS**: (Java) OpenIRIS is an OpenFlow-based SDN controller platform toward carrier-grade networks. It has an elaborate I/O engine for its performance enhancement, Loxigen-based OpenFlow message processor, Floodlight-compatible north-bound REST-API, and extensible user interface. For highly efficient packet processing, OpenIRIS I/O engine utilizes a combination of threading and event-driven architecture with a special type of synchronous queues. OpenIRIS supports multiple OpenFlow standards including OpenFlow 1.0.0 and 1.3.2.

**MUL**: (C) MūL, is an openflow (SDN) controller. It has a C based muli-threaded infrastructure at its core. It supports a multi-level north bound interface for hooking up applications. It is designed for performance and reliability which is the need of the hour for deployment in mission-critical networks.

**NOX**: (C++/Python) NOX was the first OpenFlow controller.

**Jaxon**: (Java) Jaxon is a NOX-dependent Java-based OpenFlow Controller.

**Trema**: (C/Ruby) Trema is a full-stack framework for developing OpenFlow controllers in Ruby and C.

**Beacon**: (Java) Beacon is a Java-based controller that supports both event-based and threaded operation.

**Maestro**: (Java) Maestro "s an OpenFl"w "o"erating sys"em" for orchestrating network control applications.

**ND–I - OESS**: OESS is an application to configure and control OpenFlow Enabled switches through a very simple and user friendly User Interface.

**NodeFlow** (JavaScript) NodeFlow is an OpenFlow controller written in pure JavaScript for Node.JS.

## C.7.3 Special Purpose Controllers

**RouteFlow**, is an open source project to provide Virtualised IP routing services over OpenFlow enabled hardware. RouteFlow is composed by an OpenFlow Controller application, an independent RouteFlow Server, and a virtual network environment that reproduces the connectivity of a physical infrastructure and runs IP routing engines (e.g. Quagga).

**Flowvisor** (Java) FlowVisor is a special purpose OpenFlow controller that acts as a transparent proxy between OpenFlow switches and multiple OpenFlow controllers.

**SNAC** (C++) SNAC is an OpenFlow controller built on NOX, which uses a web-based policy manager to manage the network.

**Resonance** Resonance is a Network Access Control application built using NOX and OpenFlow.

**Oflops** (C) OFlops (OpenFLow Operations Per Second) is a standalone controller that benchmarks various aspects of an OpenFlow switch.

### C.7.4 Misc

**Ironflow** ironflow is a IF-MAP client to map the offered data from an Openflow controller into the MAP-Infrastructure. In addition ironflow offers the possibility to react on request for Investigation requests with the blocking of network traffic for the hosts quoted in the request.

**STS** SDN Troubleshooting Simulator.

**FlowScale** FlowScale is a project to divide and distribute traffic over multiple physical switch ports. FlowScale replicates the functionality in load balancing appliances but using a Top of Rack (ToR) switch to distribute traffic.

**NICE-OF** NICE is a tool to test OpenFlow controller application for the NOX controller platform.

**OFTest** OFTest is a Python based OpenFlow switch test framework and collection of test cases. It is based on unittest which is included in the standard Python distribution.

**Mirage** Mirage is an exokernel for constructing secure, high-performance network applications across a variety of cloud computing and mobile platforms. Apparently, it supports OpenFlow.

**Wakame VDC** (Ruby) IaaS platform that uses OpenFlow for the networking portion.

**ENVI** ENVI is a GUI framework that was designed as an extensible platform which might provide the foundation of many interesting OpenFlow-related networking visualizations.

**NS3** (C++/Python) NS3 is a network simulator. It has openflow support built in to emulate an openflow environment and also it might be used for real-time simulations.

# Annex D (informative): Authors & contributors

The following people have contributed to the present document:

**Rapporteur**:

Marie-Paule Odini, HP – marie-paule.odini@hpe.com

**Other contributors**:

Ajay Sahai, HP (ex Contextream) – Ajay.Sahai@hpe.com

Andrew Veitch, NetCracker - andrew.veitch@NetCracker.com

Antonio Gamela, Portugal Telecom - AGamelas@telecom.pt

Ashiq Khan, NTT Docomo – khan@nttdocomo.com

Barak Perlman HP (ex Contextream) – barak.perlman@hpe.com

Benny Rochwerger, Radware - BennyR@Radware.com

Bruno Chatras, Orange – Bruno.chatras@orange.com

Bryan Kimm SKT - bryan.kim@sk.com

Christoph Meyer, Ericsson - christoph.meyer@ericsson.com

Chu Junsheng, ZTE - chu.junsheng@zte.com.cn

Chunglae Cho, ETRI - clcho@etri.re.kr

Dave Duggal, Enterprise Web - dave@enterpriseweb.com

Dave Hood, Ericsson – dave.hood@ericsson.com

Deborah Brungard, AT&T - db3546@att.com

Diego Lopez Garcia, Telefonica - diego.r.lopez@TELEFONICA.COM

DongKee Lee, SKT - DK Lee dongkee.lee@sk.com

Elena Demaria, Telecom Italia - elena.demaria@telecomitalia.it

Evelyn Roch, Huawei - Evelyne.Roch@huawei.com

Fabricio Invernizzi, Telecom Italia - fabrizio.invernizzi@telecomitalia.it

Gerald Kunzmann, Docomo – kunzmann@docomonlab-euro.com

Gao Gongying, China Unicom - gaogy16@chinaunicom.cn

Haleplidis Evangelos, University of Patras - ehalep@ECE.UPATRAS.GR

Han Jie, HP - jie.han3@hpe.com

Hiroshi Dempo, NEC - dem@ah.jp.nec.com

Hiroyuki Kitada, NTT Lab - kitada.hiroyuki@lab.ntt.co.jp

Jaye Sauer, Coriant - jaye.sauer@coriant.com

Jeff Higgs, HP - jeff.higgs@hpe.com

Joan Triay, Docomo – triay@docomolab-euro.com

Jonghan Park, SKT - jonghan.park@sk.com

Jorge Carapinha, Portugal Telecom - JorgeC@telecom.pt

Kazuaki Obana, NTT Docomo – kazuaki.obana.uz@nttdocomo.com

Luis M. Contreras, Telefonica - luismiguel.contrerasmurillo@telefonica.com

Misahiro Yoshida, NTT - yoshida.masahiro@lab.ntt.co.jp

Maxim Klyus, NetCracker - klyus@NetCracker.com

Mio Jie, China Unicom - miaojie9@chinaunicom.cn

Monica Lazer, AT&T - ml4535@att.com

Nabil Damouny, Netronome - nabil.damouny@netronome.com

Nicolas Bouthors, Qosmos – nicolas.bouthors@qosmos.com

Olaf Renner, Nokia - olaf.renner@nokia.com

Parviz Yegani, Juniper - pyegani@juniper.net

Reza Shahab, ALU - reza.shahab@alcatel-lucent.com

Ryosuke Kurbayashi, NTT Docomo – ryousuke.kurebayashi.pz@nttdocomo.com

Serge Manning, Sprint – serge.manning@sprint.com

Sharon Rozov, HP (ex Contextream) – Sharon.rozov@hpe.com

Takashi Shimizu, NTT - shimizu.takashi@LAB.NTT.CO.JP

Thinh Nguyenphu, Nokia - thinh.nguyenphu@nokia.com

Wang Shimao, HP - shi-mao.wang@hpe.com

Xue Miao, China Unicom - xuemiao9@chinaunicom.cn

Yang Xu, Huawei - yangxu5@huawei.com

Yohei Katayama, NTT Docomo – youhei.katayama.zr@nttdocomo.com

Zhu Lei, Huawei - lei.zhu@huawei.com

# Annex E (informative): Change History

| Date | Version | Information about changes |
|---|---|---|
| December 2014 | 0.0.1 | First publication of the GS after approval by ETSI NFV EVE#1<br>http://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2014//NFVEVE(14)000002r3_TOC_for_SDN-NFV_WI.docx<br><br>http://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2014/NFVEVE(14)000003r1_SDN_Section_1_to_4.docx<br>http://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2014/NFVEVE(14)000004r1_SDN_Section_5_and_6.docx<br>http://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2014/NFVEVE(14)000005r1_SDN_Annex_A_to_D.docx<br>http://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2014/NFVEVE(14)000006r2_SDN_annex.docx |
| January 2015 | 0.0.2 | Use of the latest ETSI GS Template<br>Some editorial changes<br>Approved contributions by ETSI NFV EVE#2<br>NFVEVE(15)000004r3<br>NFVEVE(15)000005r1<br>NFVEVE(15)000011 |
| March 2015 | 0.0.3 | Approved contribution by ETSI NFV EVE#3<br>http://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015/NFVEVE(15)000016r1_Scope_of_EVE-005.docx<br>Approved contribution by ETSI NFV#9 EVE<br>http://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015/NFVEVE(15)000025r4_NFVEVE-SDN_Clause_4_3.doc<br>http://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015/NFVEVE(15)000026_PoC14_Contribution_on_SDN.docx<br>http://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015/NFVEVE(15)000031_PoC1_CloudNFV_contribution_on_SDN_for_NFVEVE.doc<br>http://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015/NFVEVE(15)000047r1_NFVEVE_15_000010_Content_for_Annex_B_10_-_POC_23.docx<br>http://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015/NFVEVE(15)000052r1_Forwarding_Graphs_and_NFPs_in_SDN.docx<br>http://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015/NFVEVE(15)000054r1_SDN_based_NaaS_provided_by_other_Administration_Domain.docx<br>http://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015/NFVEVE(15)000055r1_Usage_of_SDN_in_POC_27.doc<br>http://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015/NFVEVE(15)000063r1_Text_for_NFVEVE_-WI-005_Section_4_1.docx<br>http://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015/NFVEVE(15)000064r1_Additional_text_for_Section_4_2_of_ETSI_GS_NFVEVE005_v_002.docx<br>http://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015/NFVEVE(15)000087__SDN__Opensource_SDN_controller_comparison_.doc |
| March 2015 | 0.0.4 | Approved contribution by ETSI NFV EVE#5<br>NFVEVE(15)000062r3_Discussion_of_VIM_interconnection_scenarios.docx<br>NFVEVE(15)000072r2_Usage_of_SDN_in_PoC_8.docx<br>Adding contributors name<br>Cleaning POC Clause by removing company names & logos |
| April 2015 | 0.0.5 | Approved contribution by ETSI NFV EVE#6<br>https://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015/NFVEVE(15)000036r2_PoC_16_NFVIaaS_Contribution_on_SDN.docx<br>https://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015/NFVEVE(15)000056r2_EVE005-Opensource_comparison-Opencontrail.doc |

| Date | Version | Information about changes |
|---|---|---|
| | | https://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015//NFVEVE(15)000066r4_Proposed_Addition_to_Discussion_of_VIM_interconnection_scena.docx<br>https://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015/NFVEVE(15)000084r3_Proposal_for_NFVEVE_-SDN_Annex_C.doc<br>https://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015/NFVEVE(15)000091r3_OpenDaylight_Licence.docx<br>https://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015/NFVEVE(15)000097r2_proposal_for_Annex_D_7_-_OpenIRIS.docx<br><br>alphabetic ordering of contributors names<br>update figure numbering |
| April 2015 | 0.0.6 | Approved contribution by EVE NFV EVE#7<br><br>https://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015//NFVEVE(15)000147r1_ETSI_GS_NFV-EVE005_v005_rev_PT_AG_-_part_3__Clauses_5_and_6.docx<br>https://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015//NFVEVE(15)000149r1_ETSI_GS_NFVEVE005_v005_section52_review_by_PT.docx<br>https://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015//NFVEVE(15)000156r2_ETSI_GS_NFV-EVE005_v005_rev_PT_NSN_-_part_2__Clause_4.docx<br>https://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015//NFVEVE(15)000144_ETSI_GS_NFVEVE005_v005_rev_PT_AG_-_part_1__Clauses_1-2-3.docx<br>https://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015//NFVEVE(15)000154_Remove_POC_6_from_EVE005.doc<br>https://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015//NFVEVE(15)000151_A_review_contribution_for_NFV-EVE005_draft_GS.zip<br>https://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015//NFVEVE(15)000155r1_Editorial_comments_on_EVE005_Annex_B.doc |
| April 2015 | 0.0.7 | Approved contribution in EVE#3<br>NFVEVE(15)000032 |
| May 2015 | 0.0.8 | Minor editorial updates on pages 1 and 2 + disclaimer text added to cover page before to send to ONF as Liaison Statement. |
| May 2015 | 0.0.9 | *Approved contribution in EVE#9*<br>https://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015//NFVEVE(15)000181_Annex_Update_-_ONOS_table.doc<br>https://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015//NFVEVE(15)000049r3_SDN_Controlled_VNF_Forwarding_PoC-28_Lessons_Learned.zip<br>https://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015/NFVEVE(15)000098r6_vCPE_Parental_Control.doc<br>https://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015//NFVEVE(15)000162r1_EVE005_Additions_to_SDN_for_inter-VIM_connectivity.docx<br>https://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015//NFVEVE(15)000090r3_GBP_Use_case_.zip |
| May 2015 | 0.0.10 | *Approved contributions in EVE#10-Sanya*<br>https://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015/NFVEVE(15)000200r2_clause_5_4__SDN_in_Virtualised_environment.zip<br>https://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015/NFVEVE(15)000086r6_SFC_with_SDN_and_NFV.zip<br>https://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015/NFVEVE(15)000203_ONOS_update.doc |
| June 2015 | 0.0.11 | *Approved contributions in EVE#11*<br>https://docbox.etsi.org/ISG/NFV/TSC/05-CONTRIBUTIONS/2015//NFVTSC(15)000041r1_Dislaimer__updated__for_NFV_drafts.docx |

| Date | Version | Information about changes |
|---|---|---|
| | | https://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015/NFVEVE(15)000184r4_Clause_5_3___hiearchy_of_SDN_controller.doc<br>https://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015//NFVEVE(15)000186r2_SDN_Hierarchy_Clause_5_3.docx<br>https://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015/NFVEVE(15)000223r1_EVE005_-_clause_5_3__Hiearchy_SDN_controller_-_combined.doc<br>https://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015/NFVEVE(15)000225r1_GS_v0010_content_suggestion_and_editorial_changes.docx<br>add last diagram of 25r4 that was dropped by the rapporteur in previous updates |
| June 2015 | 0.0.12 | *Approved contribution in EVE#12*<br>https://docbox.etsi.org/ISG/NFV/EVE/05-CONTRIBUTIONS/2015//NFVEVE(15)000231r1_EVE005_-_D_5_Lessons_learnt.doc<br>Replace Title of Clause 6 by "Functional Recommendations" and replaced 'requirements' by 'recommendations' in this clause |
| July 2015 | 0.0.13 | *Approved contribution in EVE#13*<br>Disclaimer update: NFVTSC(15)000041r3<br>NFVEVE(15)000068r6<br>NFVEVE(15)000241 |
| July 2015 | 0.0.14 | *Approved contribution in EVE#13*<br>NFVEVE(15)000268 |
| Aug, 2015 | 0.0.15 | *Approved contribution in EVE San Jose*<br>*Updating references, figures, Tables*<br>NFVEVE(15)000269r2<br>NFVEVE(15)000280r5<br>NFVEVE(15)000292<br>NFVEVE(15)000296r2<br>NFVEVE(15)000307r1<br>NFVEVE(15)000320<br>NFVEVE(15)000324r2<br>NFVEVE(15)000328r1<br>NFVEVE(15)000329r2 |
| Aug, 2015 | 0.1.0 | *Approved contribution in EVE#16*<br>NFVEVE(15)000337<br>NFVEVE(15)000339<br>NFVEVE(15)000345r1<br>NFVEVE(15)000348<br>NFVEVE(15)000349 |
| Sept 2015 | 0.1.1 | *CleanUp EditHelp*<br>*Approved contributions in EVE#17*<br>NFVEVE(15)000358<br>NFVEVE(15)000359r1<br>NFVEVE(15)000360r1<br>*Approved contributions in EVE#17-a*<br>NFVEVE(15)000362r3<br>NFVEVE(15)000364<br>NFVEVE(15)000365r1<br>NFVEVE(15)000366r1<br>NFVEVE(15)000367<br>NFVEVE(15)000368r1<br>NFVEVE(15)000369r1<br>NFVEVE(15)000370<br>NFVEVE(15)000383r1 |

| Date | Version | Information about changes |
|---|---|---|
| Sept 2015 | 0.1.2 | *Approved contributions in EVE#17-a*<br>NFVEVE(15)000381r1<br>*Approved contributions in EVE#18*<br>NFVEVE(15)000269r3<br>NFVEVE(15)000361r2<br>NFVEVE(15)000377r3<br>NFVEVE(15)000386r1<br>NFVEVE(15)000392r1<br>NFVEVE(15)000393r1<br>*Approved contributions in EVE#18-a*<br>NFVEVE(15)000387r1<br>NFVEVE(15)000390r1<br>NFVEVE(15)000391r2<br>NFVEVE(15)000395r1<br>NFVEVE(15)000397<br>NFVEVE(15)000404<br>NFVEVE(15)000405r1<br>NFVEVE(15)000406r2<br>NFVEVE(15)000407<br>NFVEVE(15)000410 |
| Sept 2015 | 0.1.3 | *Approved contributions in EVE#19*<br>NFVEVE(15)000417<br>NFVEVE(15)000418<br>NFVEVE(15)000419<br>NFVEVE(15)000420<br>NFVEVE(15)000423r1 |
| Sept 2015 | 0.2.0 | *Editorial change Figure Update*<br>*WG approved Draft* |

# History

| Document history | | |
|---|---|---|
| V1.1.1 | December 2015 | Publication |
| | | |
| | | |
| | | |
| | | |