NFV RG
Internet-Draft
Intended status: Informational
Expires: March 7, 2019

CJ. Bernardos, Ed.
UC3M
LM. Contreras
TID
I. Vaishnavi
Huawei
R. Szabo
Ericsson
J. Mangues
CTTC
X. Li
NEC
F. Paolucci
A. Sgambelluri
B. Martini
L. Valcarenghi
SSSA
G. Landi
Nextworks
D. Andrushko
MIRANTIS
A. Mourad
InterDigital

September 3, 2018

Multi-domain Network Virtualization
draft-bernardos-nfvrg-multidomain-05

Abstract

This document analyzes the problem of multi-provider multi-domain orchestration, by first scoping the problem, then looking into potential architectural approaches, and finally describing the solutions being developed by the European 5GEx and 5G-TRANSFORMER projects.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 7, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 3
2. Terminology 4
3. Background: the ETSI NFV architecture 5
4. Multi-domain problem statement 8
5. Multi-domain architectural approaches 9
5.1. ETSI NFV approaches 9
5.2. Hierarchical 17
5.3. Cascading 20
6. Virtualization and Control for Multi-Provider Multi-Domain . 20
6.1. Interworking interfaces 22
6.2. 5GEx Multi Architecture 23
6.3. 5G-TRANSFORMER Architecture 26
6.3.1. So-Mtp Interface (IF3) 28
6.3.2. So-So Interface (IF2) 29
6.3.3. Vs-So Interface (IF1) 30
7. Multi-domain orchestration and Open Source 31
8. IANA Considerations 32
9. Security Considerations 32
10. Acknowledgments 32
11. Informative References 33
Authors' Addresses 33

1. Introduction

The telecommunications sector is experiencing a major revolution that will shape the way networks and services are designed and deployed for the next decade. We are witnessing an explosion in the number of applications and services demanded by users, which are now really capable of accessing them on the move. In order to cope with such a demand, some network operators are looking at the cloud computing paradigm, which enables a potential reduction of the overall costs by outsourcing communication services from specific hardware in the operator's core to server farms scattered in datacenters. These services have different characteristics if compared with conventional IT services that have to be taken into account in this cloudification process. Also the transport network is affected in that it is evolving to a more sophisticated form of IP architecture with trends like separation of control and data plane traffic, and more fine-grained forwarding of packets (beyond looking at the destination IP address) in the network to fulfill new business and service goals.

Virtualization of functions also provides operators with tools to deploy new services much faster, as compared to the traditional use of monolithic and tightly integrated dedicated machinery. As a natural next step, mobile network operators need to re-think how to evolve their existing network infrastructures and how to deploy new ones to address the challenges posed by the increasing customers' demands, as well as by the huge competition among operators. All these changes are triggering the need for a modification in the way operators and infrastructure providers operate their networks, as they need to significantly reduce the costs incurred in deploying a new service and operating it. Some of the mechanisms that are being considered and already adopted by operators include: sharing of network infrastructure to reduce costs, virtualization of core servers running in data centers as a way of supporting their load-aware elastic dimensioning, and dynamic energy policies to reduce the monthly electricity bill. However, this has proved to be tough to put in practice, and not enough. Indeed, it is not easy to deploy new mechanisms in a running operational network due to the high dependency on proprietary (and sometime obscure) protocols and interfaces, which are complex to manage and often require configuring multiple devices in a decentralized way.

Furthermore, 5G networks are being designed to be capable of fulfilling the needs of a plethora of vertical industries (e.g., automotive, eHealth, media), which have a wide variety of requirements [ngmn_5g_whitepaper]. The slicing concept tries to make the network of the provider aware of the business needs of tenants (e.g., vertical industries) by customizing the share of the network assigned to them. The term network slice was coined to refer to a

complete logical network composed of network functions and the resources to run them [ngmn_slicing]. These resources include network, storage, and computing. The way in which services requested by customers of the provider are assigned to slices depends on customer needs and provider policies. The system must be flexible to accommodate a variety of options.

Another characteristic of current and future telecommunication networks is complexity. It comes from three main aspects. First, heterogeneous technologies are often separated in multiple domains under the supervision of different network managers, which exchange provisioning orders that are manually handled. This does not only happen between different operators, but also inside the network of the same operator. Second, the different regional scope of each operator requires peering with others to extend their reach. And third, the increasing variety of interaction among specialized providers (e.g., mobile operator, cloud service provider, transport network provider) that complement each other to satisfy the service requests from customers. In conclusion, realizing the slicing vision to adapt the network to needs of verticals will require handling multi-provider and multi-domain aspects.

Additionally, Network Function Virtualization (NFV) and Software Defined Networking (SDN) are changing the way the telecommunications sector will deploy, extend and operate its networks. Together, they bring the required programmability and flexibility. Moreover, these concepts and network slicing are tightly related. In fact, slices may be implemented as NFV network services. However, building a complete end-to-end logical network will likely require stitching services offered by multiple domains from multiple providers. This is why multi-domain network virtualization is crucial in 5G networks.

2. Terminology

The following terms used in this document are defined by the ETSI NVF ISG, and the ONF and the IETF:

NFV Infrastructure (NFVI): totality of all hardware and software components which build up the environment in which VNFs are deployed

NFV Management and Orchestration (NFV-MANO): functions collectively provided by NFVO, VNFM, and VIM.

NFV Orchestrator (NFVO): functional block that manages the Network Service (NS) lifecycle and coordinates the management of NS lifecycle, VNF lifecycle (supported by the VNFM) and NFVI

resources (supported by the VIM) to ensure an optimized allocation of the necessary resources and connectivity.

Network Service Orchestration (NSO): function responsible for the Network Service lifecycle management, including operations such as: On-board Network Service, Instantiate Network Service, Scale Network Service, Update Network Service, etc.

OpenFlow protocol (OFP): allowing vendor independent programming of control functions in network nodes.

Resource Orchestration (RO): subset of NFV Orchestrator functions that are responsible for global resource management governance.

Service Function Chain (SFC): for a given service, the abstracted view of the required service functions and the order in which they are to be applied. This is somehow equivalent to the Network Function Forwarding Graph (NF-FG) at ETSI.

Service Function Path (SFP): the selection of specific service function instances on specific network nodes to form a service graph through which an SFC is instantiated.

Virtualized Infrastructure Manager (VIM): functional block that is responsible for controlling and managing the NFVI compute, storage and network resources, usually within one operator's Infrastructure Domain.

Virtualized Network Function (VNF): implementation of a Network Function that can be deployed on a Network Function Virtualization Infrastructure (NFVI).

Virtualized Network Function Manager (VNFM): functional block that is responsible for the lifecycle management of VNF.

3. Background: the ETSI NFV architecture

The ETSI ISG NFV is a working group which, since 2012, aims to evolve quasi-standard IT virtualization technology to consolidate many network equipment types into industry standard high volume servers, switches, and storage. It enables implementing network functions in software that can run on a range of industry standard server hardware and can be moved to, or loaded in, various locations in the network as required, without the need to install new equipment. To date, ETSI NFV is by far the most accepted NFV reference framework and architectural footprint [etsi_nvf_whitepaper]. The ETSI NFV framework architecture framework is composed of three domains (Figure 1):

- Virtualized Network Function, running over the NFVI.

- NFV Infrastructure (NFVI), including the diversity of physical resources and how these can be virtualized. NFVI supports the execution of the VNFs.

- NFV Management and Orchestration, which covers the orchestration and life-cycle management of physical and/or software resources that support the infrastructure virtualization, and the life-cycle management of VNFs. NFV Management and Orchestration focuses on all virtualization specific management tasks necessary in the NFV framework.

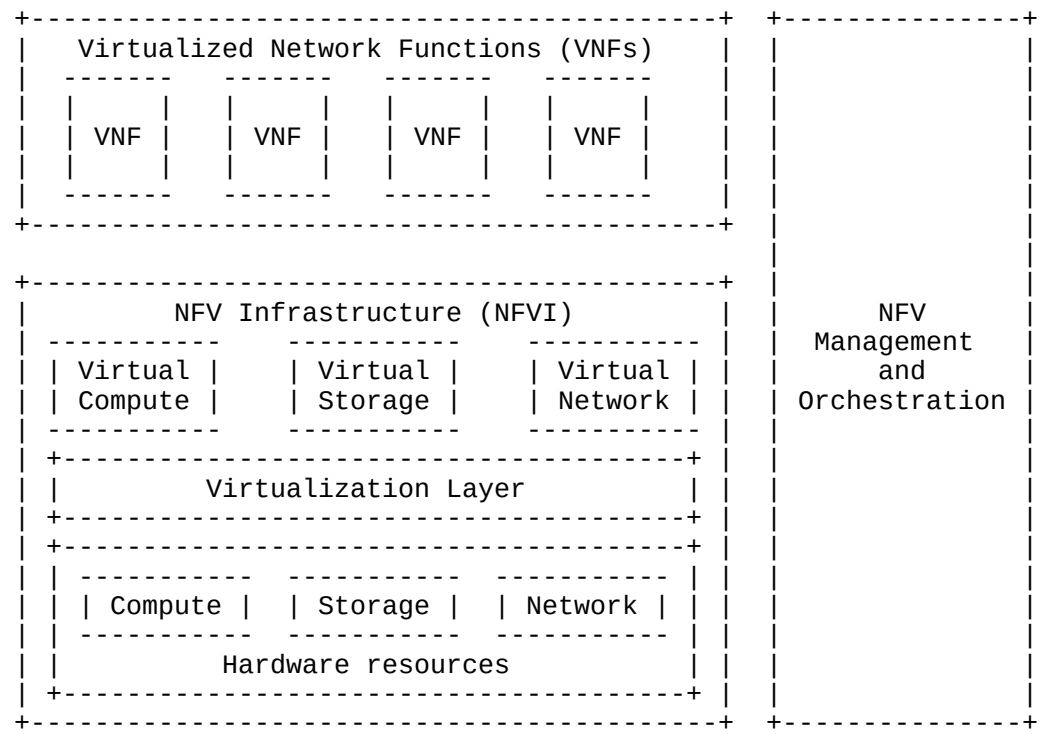


Figure 1: ETSI NFV framework

The NFV architectural framework identifies functional blocks and the main reference points between such blocks. Some of these are already present in current deployments, whilst others might be necessary additions in order to support the virtualization process and consequent operation. The functional blocks are (Figure 2):

- Virtualized Network Function (VNF).

- Element Management (EM).
- NFV Infrastructure, including: Hardware and virtualized resources, and Virtualization Layer.
- Virtualized Infrastructure Manager(s) (VIM).
- NFV Orchestrator.
- VNF Manager(s).
- Service, VNF and Infrastructure Description.
- Operations and Business Support Systems (OSS/BSS).

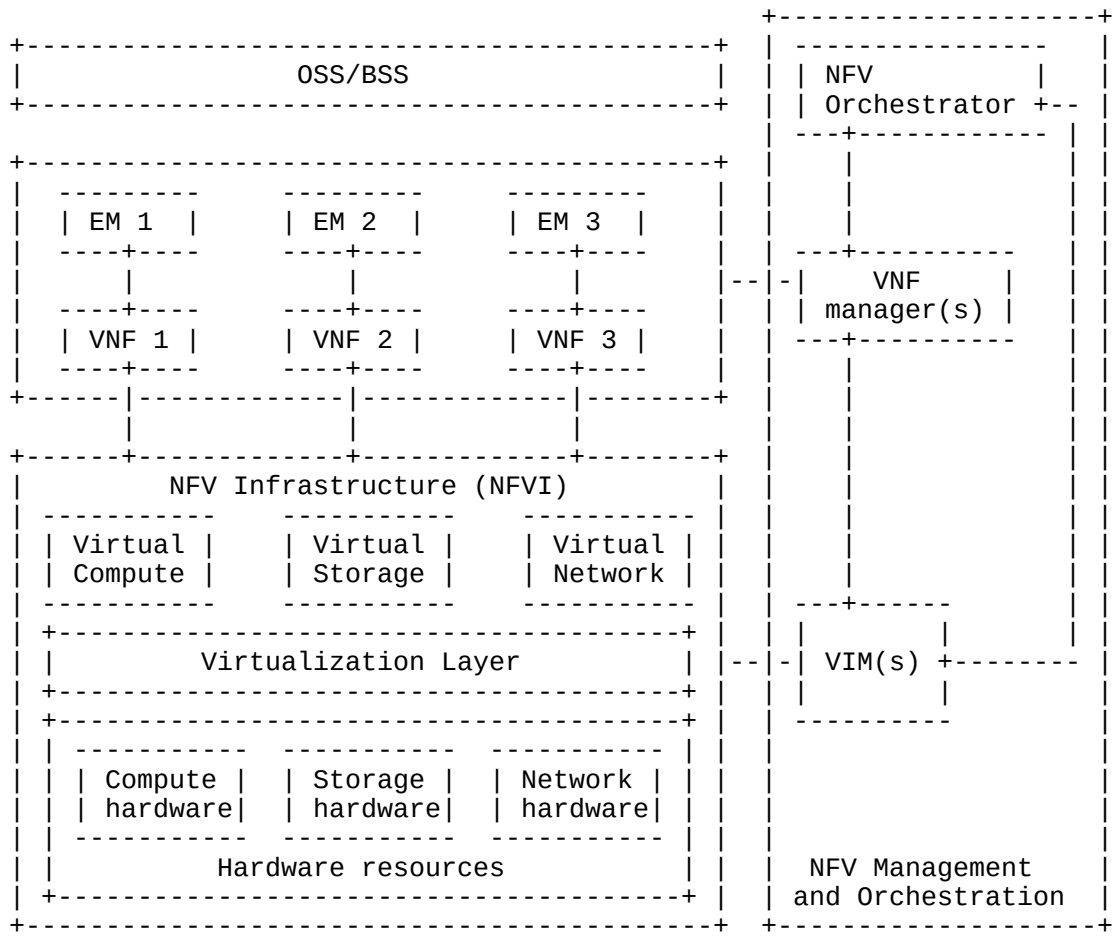


Figure 2: ETSI NFV reference architecture

4. Multi-domain problem statement

Market fragmentation results from having a multitude of telecommunications network and cloud operators each with a footprint focused to a specific region. This makes it difficult to deploy cost effective infrastructure services, such as virtual connectivity or compute resources, spanning multiple countries as no single operator has a big enough footprint. Even if operators largely aim to provide the same infrastructure services (VPN connectivity, compute resources based on virtual machines and block storage), inter-operator collaboration tools for providing a service spanning several administrative boundaries are very limited and cumbersome. This makes service development and provisioning very time consuming. For

example, having a VPN with end-points in several countries, in order to connect multiple sites of a business (such as a hotel chain), requires contacting several network operators. Such an approach is possible only with significant effort and integration work from the side of the business. This is not only slow, but also inefficient and expensive, since the business also needs to employ networking specialists to do the integration instead of focusing on its core business

Technology fragmentation also represents a major bottleneck internally for an operator. Different networks and different parts of a network may be built as different domains using separate technologies, such as optical or packet switched (with different packet switching paradigms included); having equipment from different vendors; having different control paradigms, etc. Managing and integrating these separate technology domains requires substantial amount of effort, expertise, and time. The associated costs are paid by both network operators and vendors alike, who need to design equipment and develop complex integration features. In addition to technology domains, there are other reasons for having multiple domains within an operator, such as, different geographies, different performance characteristics, scalability, policy or simply historic (e.g., result of a merge or an acquisition). Multiple domains in a network are a necessary and permanent feature however, these should not be a roadblock towards service development and provisioning, which should be fast and efficient.

A solution is needed to deal with both the multi-operator collaboration issue, and address the multi-domain problem within a single network operator. While these two problems are quite different, they also share a lot of common aspects and can benefit from having a number of common tools to solve them.

5. Multi-domain architectural approaches

This section summarizes different architectural options that can be considered to tackle the multi-domain orchestration problem.

5.1. ETSI NFV approaches

Recently, the ETSI NFV ISG has started to look into viable architectural options supporting the placement of functions in different administrative domains. In the document [etsi_nvf_ifa009], different approaches are considered, which we summarize next.

The first option (shown in Figure 3) is based on a split of the NFVO into Network Service Orchestrator (NSO) and Resource Orchestrator (RO). A use case that this separation could enable is the following:

a network operator offering its infrastructure to different departments within the same operator, as well as to a different network operator like in cases of network sharing agreements. In this scenario, an administrative domain can be defined as one or more data centers and VIMs, providing an abstracted view of the resources hosted in it.

A service is orchestrated out of VNFs that can run on infrastructure provided and managed by another Service Provider. The NSO manages the lifecycle of network services, while the RO provides an overall view of the resources present in the administrative domain to which it provides access and hides the interfaces of the VIMs present below it.

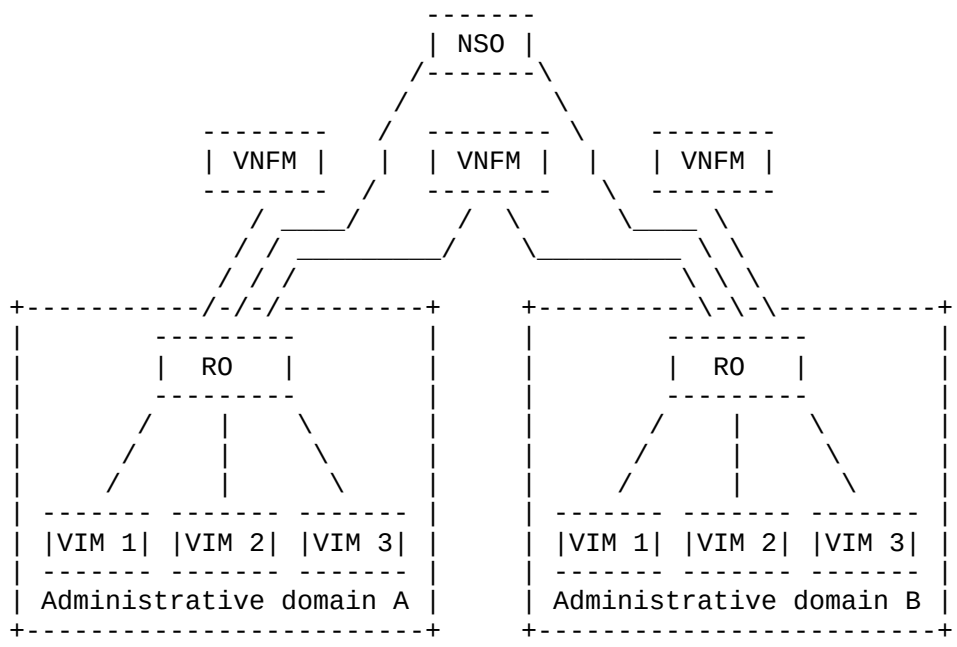


Figure 3: Infrastructure provided using multiple administrative domains (from ETSI GS NFV-IFA 009 V1.1.1)

The second option (shown in Figure 4) is based on having an umbrella NFVO. A use case enabled by this is the following: a Network Operator offers Network Services to different departments within the same operator, as well as to a different network operator like in cases of network sharing agreements. In this scenario, an administrative domain is compose of one or more Datacentres, VIMs, VNFMs (together with their related VNFs) and NFVO, allowing distinct specific sets of network services to be hosted and offered on each.

A top Network Service can include another Network Service. A Network Service containing other Network Services might also contain VNFs. The NFVO in each admin domain provides visibility of the Network Services specific to this admin domain. The umbrella NFVO is providing the lifecycle management of umbrella network services defined in this NFVO. In each admin domain, the NFVO is providing standard NFVO functionalities, with a scope limited to the network services, VNFs and resources that are part of its admin domain.

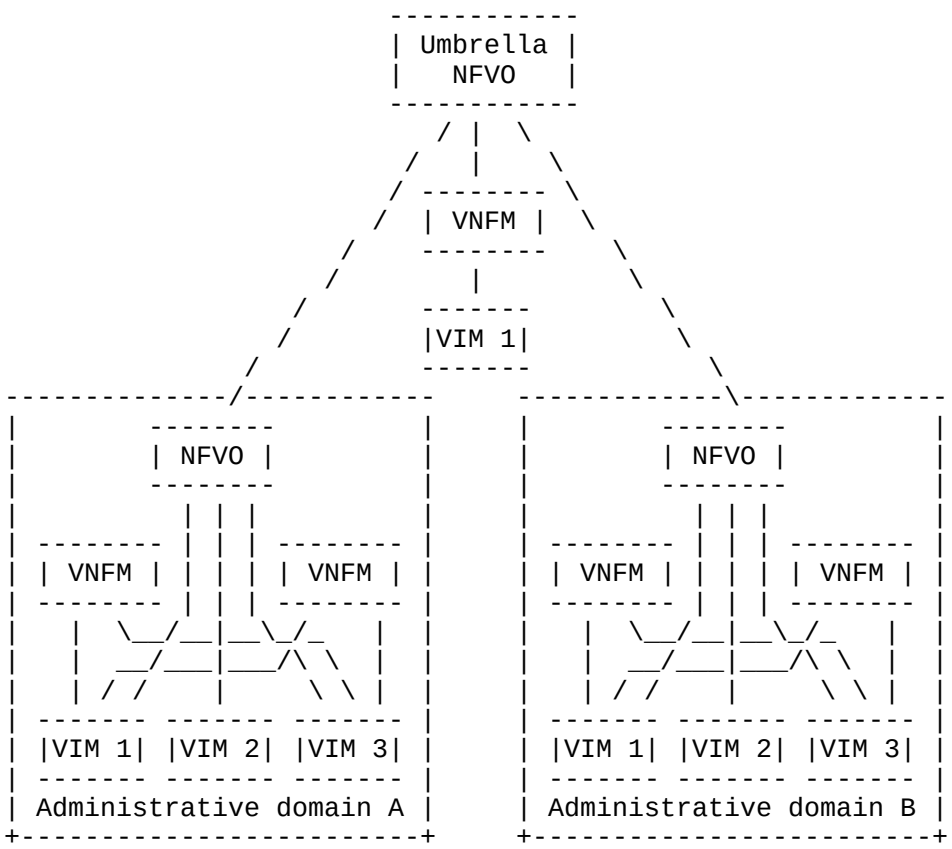


Figure 4: Network services provided using multiple administrative domains (from ETSI GS NFV-IFA 009 V1.1.1)

More recently, ETSI NFV has released a new whitepaper, titled "Network Operator Perspectives on NFV priorities for 5G" [etsi_nvf_whitepaper_5g], which provides network operator perspectives on NFV priorities for 5G and identifies common technical features in terms of NFV. This whitepaper identifies multi-site/ multi-tenant orchestration as one key priority. ETSI highlights the

support of Infrastructure as a Service (IaaS), NFV as a Service (NFVaaS) and Network Service (NS) composition in different administrative domains (for example roaming scenarios in wireless networks) as critical for the 5G work.

In January 2018 ETSI NFV released a report about NFV MANO architectural options to support multiple administrative domains [etsi_nvf_ifa028]. This report presents two use cases: the NFVI as a Service (NFVIaaS) case, where a service provider runs VNFs inside an NFVI operated by a different service provider, and the case of Network Services (NS) offered by multiple administrative domains, where an organization uses NS(s) offered by another organization.

In the NFVIaaS use case, the NFVIaaS consumer runs VNF instances inside an NFVI provided by a different service provider, called NFVIaaS provider, that offers computing, storage, and networking resources to the NFVIaaS consumer. Therefore, the NFVIaaS consumer has the control on the applications that run on the virtual resources, but has not the control of the underlying infrastructure, which is instead managed by the NFVIaaS provider. In this scenario, the NFVIaaS provider’s domain is composed of one or more NFVI-PoPs and VIMs, while the NFVIaaS consumer’s domain includes one or more NSs and VNFs managed by its own NFVO and VNFMs, as depicted in Figure 5.

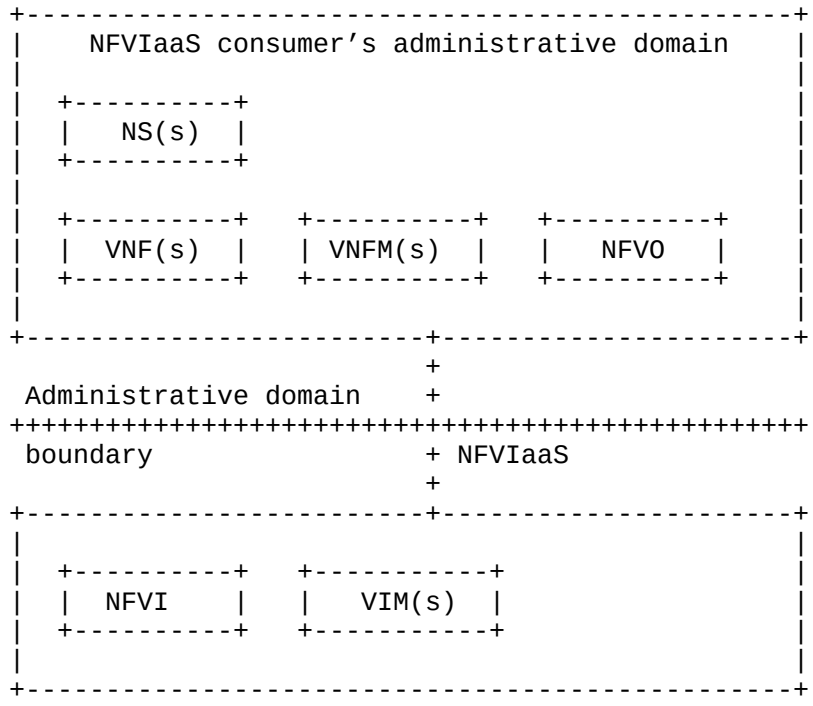


Figure 5: NFVI use case

The ETSI IFA 028 defines two main options to model the interfaces between NFVIaaS provider and consumer for NFVIaaS service requests, as follows:

1. Access to Multiple Logical Points of Contacts (MLPOC) in the NFVIaaS provider's administrative domain. In this case the NFVIaaS consumer has visibility of the NFVIaaS provider's VIMs and it interacts with each of them to issue NFVIaaS service requests, through Or-Vi (IFA 005) or Vi-Vnfm (IFA 006) reference points.

2. Access to a Single Logical Point of Contact (SLPOC) in the NFVIaaS provider's administrative domain. In this case the NFVIaaS provider's VIMs are hidden from the NFVIaaS consumer and a single unified interface is exposed by the SLPOC to the NFVIaaS consumer. The SLPOC manages the information about the organization, the availability and the utilization of the infrastructure resources, forwarding the requests from the NFVIaaS consumer to the VIMs. The interaction between SLPOC and NFVIaaS consumer is based on IFA 005 or IFA 006 interfaces, while

the interface between the SLPOC and the underlying VIMs is based on the IFA 005.

The two options are shown in Figure 6 and Figure 7 respectively, where we assume the direct mode for the management of VNF resources. In addition, the ETSI IFA 028 includes the possibility of an indirect management mode of the VNF resources through the consumer NFVIaaS NFVO and the IFA 007 interface. In this latter case between the consumer NFVIaaS NFVO and the provider NFVIaaS NFVO only the IFA 005 interface is utilized.

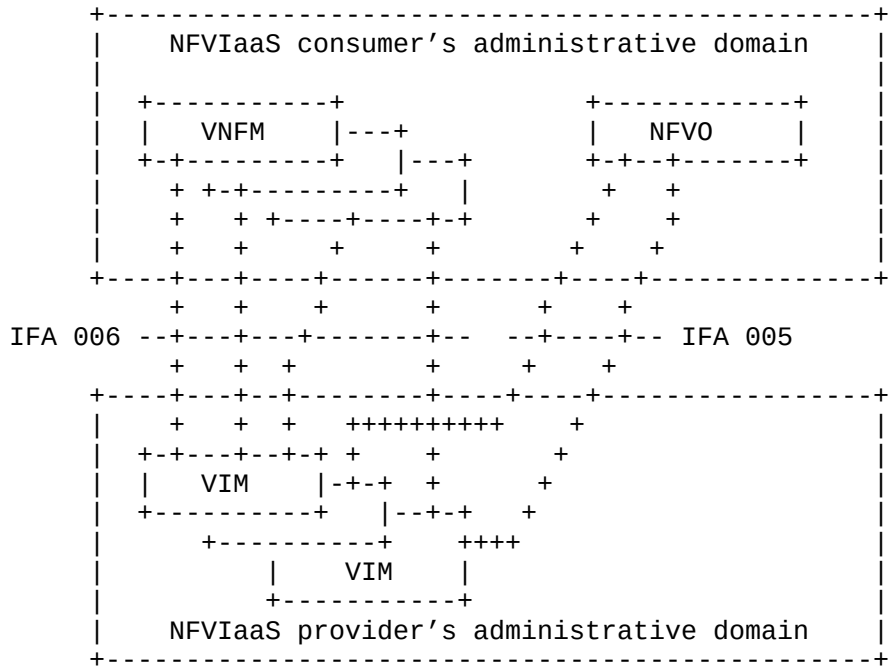


Figure 6: NFVIaaS architecture: MLPOC option

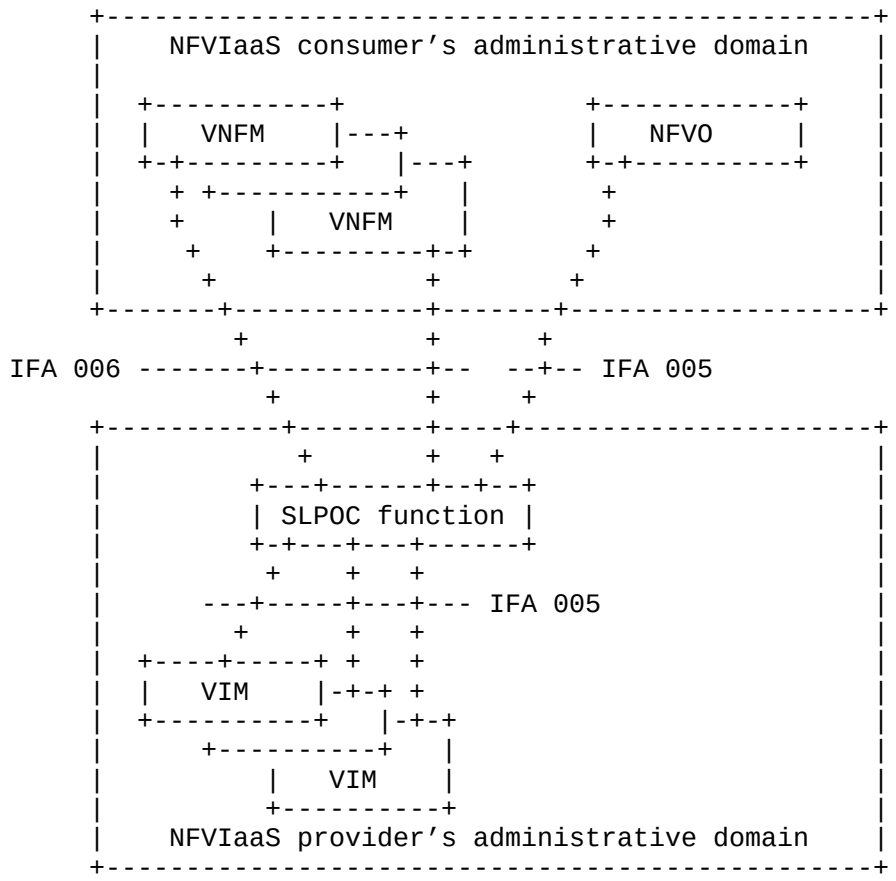


Figure 7: NFVIaaS architecture: SLPOC option

In the use case related to Network Services provided using multiple administrative domains, each domain includes an NFVO and one or more NFVI PoPs, VIMs and VNFMs. The NFVO in each domain offers a catalogue of Network Services that can be used to deploy nested NSs, which in turn can be composed into composite NSs, as shown in Figure 8. Nested NSs can be also shared among different composite NSs.

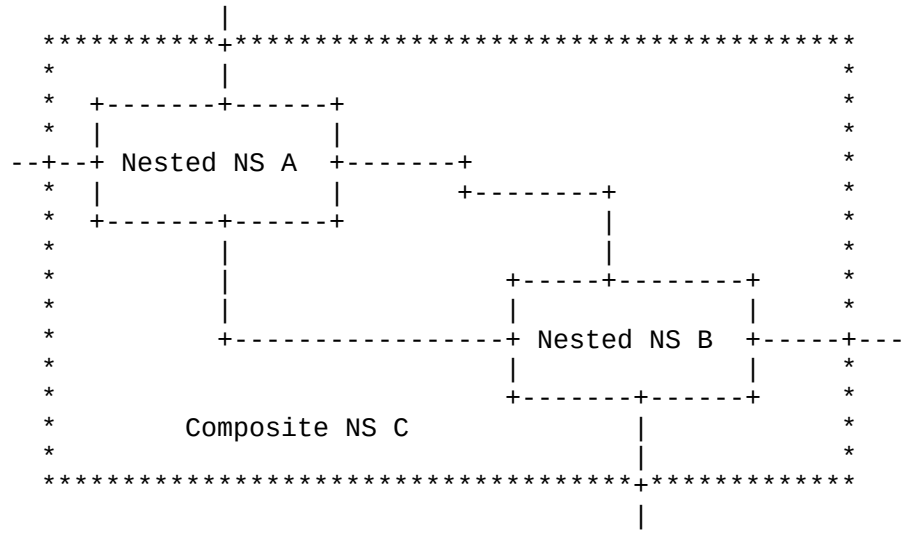


Figure 8: Composite and nested NSs

The management of the NS hierarchy is handled through a hierarchy of NFVOs, with one of them responsible for the instantiation and lifecycle management of the composite NS, coordinating the actions of the other NFVOs that manage the nested NSs. These two different kinds of NFVOs interact through a new reference point, named Or-Or, as shown in Figure 9, where NFVO-1 manages composite NSs and NFVO-2 manages nested NSs. To build the composite NSs, the responsible NFVO consult its own catalogue and may subscribe to the NSD notifications sent by other NFVOs.

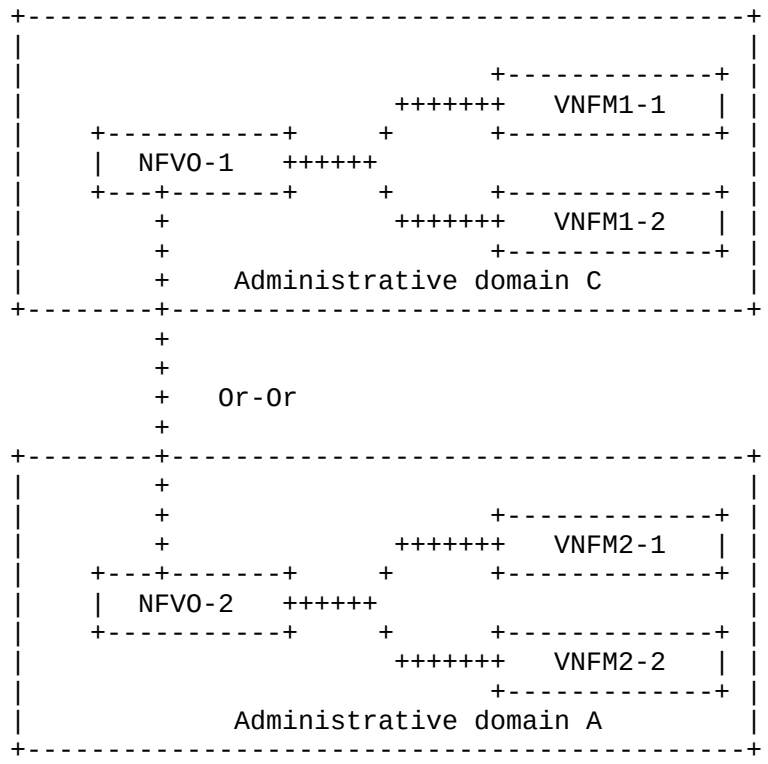


Figure 9: Architecture for management of composite and nested NS

5.2. Hierarchical

Considering the potential split of the NFVO into a Network Service Orchestrator (NSO) and a Resource Orchestrator (RO), multi-provider hierarchical interfaces may exist at their northbound APIs. Figure 10 illustrates the various interconnection options, namely:

E/NSO (External NSO): an evolved NFVO northbound API based on Network Service (NS).

E/RO (External RO): VNF-FG oriented resource embedding service. A received VNF-FG that is mapped to the northbound resource view is embedded into the distributed resources collected from southbound, i.e., VNF-FG_in = VNF-FG_out_1 + VNF-FG_out_2 + ... + VNF-FG_out_N, where VNF-FG_out_j corresponds to a spatial embedding to subordinate domain "j". For example, Provider 3's MP-NFVO/RO creates VNF-FG corresponding to its E/RO and E/VIM sub-domains.

E/VIM (External VIM): a generic VIM interface offered to an external consumer. In this case the NFVI-PoP may be shared for multiple consumers, each seeing a dedicated NFVI-PoP. This corresponds to IaaS interface.

I/NSO (Internal NSO): if a Multi-provider NSO (MP-NSO) is separated from the provider's operational NSO, e.g., due to different operational policies, the MP-NSO may need this interface to realize its northbound E/NSO requests. Provider 1 illustrates a scenario the MP-NSO and the NSO are logically separated. Observe that Provider 1's tenants connect to the NSO and MP-NSO corresponds to "wholesale" services.

I/RO (Internal RO): VNF-FG oriented resource embedding service. A received VNF-FG that is mapped to the northbound resource view is embedded into the distributed resources collected from southbound, i.e., VNF-FG_in = VNF-FG_out_1 + VNF-FG_out_2 + ... + VNF-FG_out_N, where VNF-FG_out_j corresponds to a spatial embedding to subordinate domain "j". For example, Provider 1's MP-NFVO/RO creates VNF-FG corresponding to its I/RO and I/VIM sub-domains.

I/VIM (Internal VIM): a generic VIM interface at an NFVI-PoP.

Nfvo-Vim: a generic VIM interface between a (monolithic) NFVO and a VIM.

Some questions arise from this. It would be good to explore use-cases and potential benefits for the above multi-provider interfaces as well as to learn how much they may differ from their existing counterparts. For example, are (E/RO, I/RO), (E/NSO, I/NSO), (E/VIM, I/VIM) pairs different?

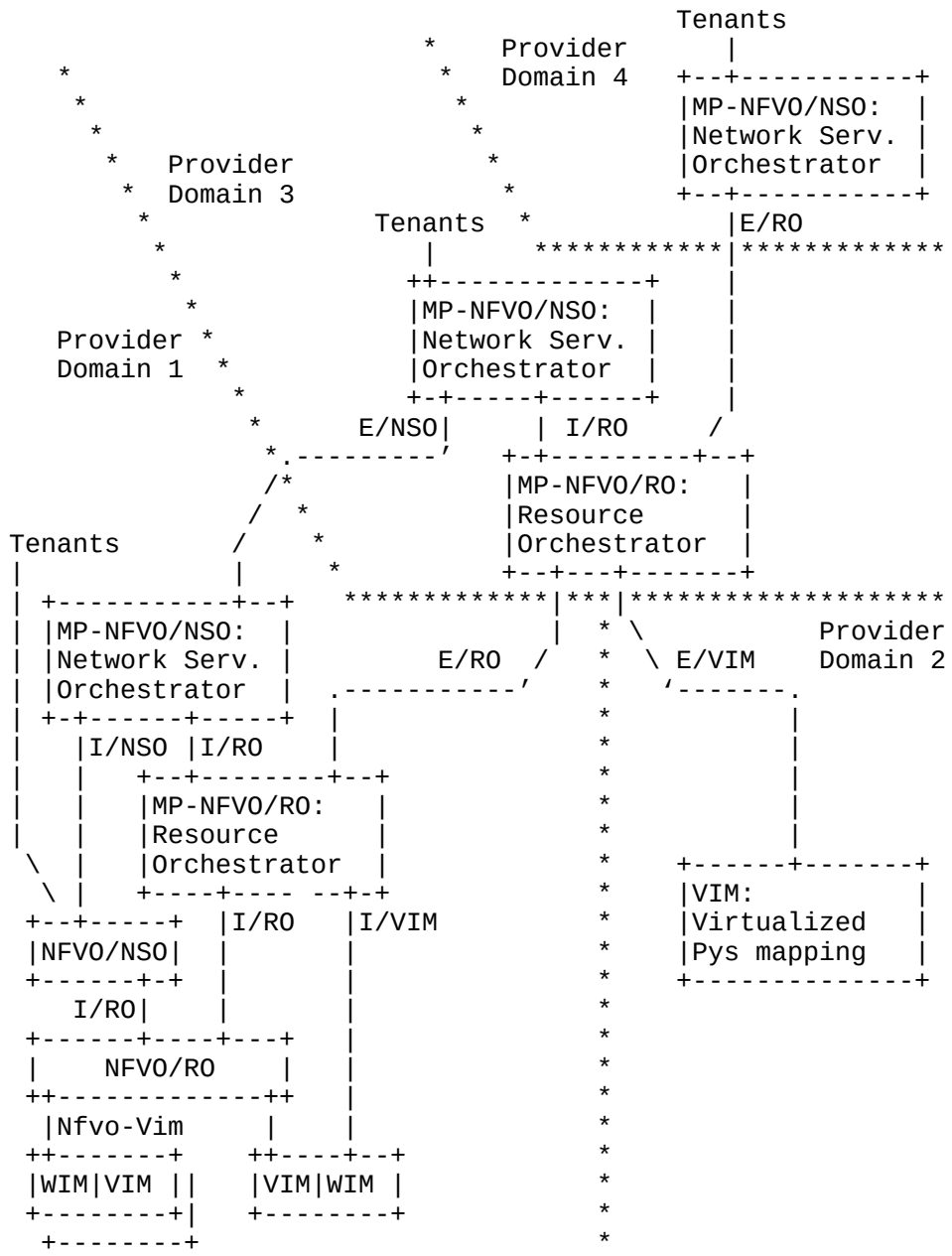


Figure 10: NSO-RO Split: possible multi-provider APIs - an illustration

5.3. Cascading

Cascading is an alternative way of relationship among providers, from the network service point of view. In this case, service decomposition is implemented in a paired basis. This can be extended in a recursive manner, then allowing for a concatenation of cascaded relations between providers.

As a complement to this, from a service perspective, the cascading of two remote providers (i.e., providers not directly interconnected) could require the participation of a third provider (or more) facilitating the necessary communication among the other two. In that sense, the final service involves two providers while the connectivity imposes the participation of more parties at resource level.

6. Virtualization and Control for Multi-Provider Multi-Domain

Orchestration operation in multi-domain is somewhat different from that in a single domain as the assumption in single domain single provider orchestration is that the orchestrator is aware of the entire topology and resource availability within its domain as well as has complete control over those resources. This assumption of technical control cannot be made in a multi domain scenario, furthermore the assumption of the knowledge of the resources and topologies cannot be made across providers. In such a scenario solutions are required that enable the exchange of relevant information across these orchestrators. This exchange needs to be standardized as shown in Figure 11.

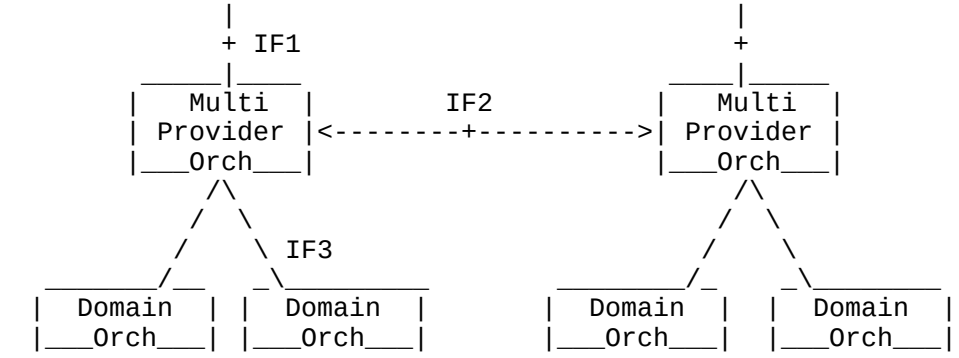


Figure 11: Multi Domain Multi Provider reference architecture

The figure shows the Multi Provider orchestrator exposing an interface 1 (IF1) to the tenant, interface 2 (IF2) to other Multi Provider Orchestrator (MPO) and an interface 3 (IF3) to individual

domain orchestratrators. Each one of these interfaces could be a possible standardization candidate. Interface 1 is exposed to the tenant who could request his specific services and/or slices to be deployed. Interface 2 is between the orchestrator and is a key interface to enable multi-provider operation. Interface 3 focuses on abstracting the technology or vendor dependent implementation details to support orchestration.

The proposed operation of the MPO follows three main technical steps. First, over interface 2 various functions such as abstracted topology discovery, pricing and service details are detected. Second, once a request for deploying a service is received over interface 1 the Multi Provider Orchestrator evaluates the best orchestrators to implement parts of this request. The request to deploy these parts are sent to the different domain orchestrators over IF2 and IF3 and the acknowledgement that these are deployed in different domain are received back over those interfaces. Third, on receipt of the acknowledgement the slice specific assurance management is started within the MPO. This assurance function collects the appropriate information over IF2 and IF3 and reports the performance back to the tenant over IF1. The assurance is also responsible for detecting any failures in the service and violations in the SLA and recommending to the orchestration engine the reconfiguration of the service or slice which again needs to be performed over IF2 and IF3.

Each of the three steps is assigned to a specific block in our high level architecture shown in Figure 12.

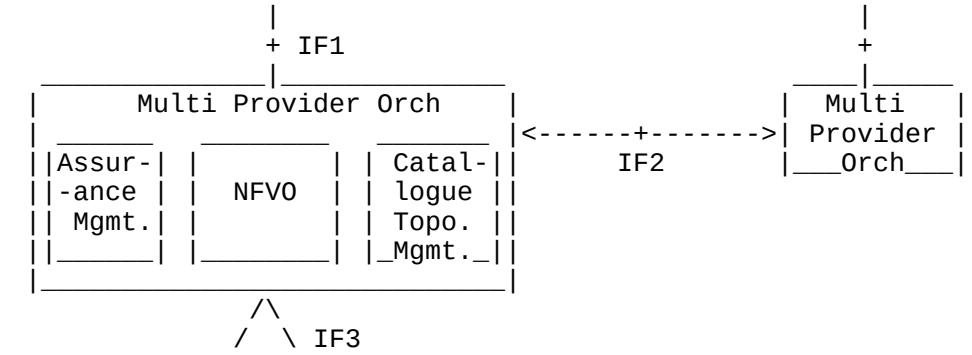


Figure 12: Detailed MPO reference architecture

The catalogue and topology management system is responsible for step 1. It discovers the service as well as the resources exposed by the other domains both on IF2 and IF3. The combination of these services with coverage over the detected topology is provided to the user over IF1. In turn the catalogue and topology management system is also

responsible for exposing the topology and service deployment capabilities to the other domain. The exposure over interface 2 to other MPO maybe abstracted and the mapping of this abstracted view to the real view when requested by the NFVO.

The NFVO (Network Function Virtualization Orchestrator) is responsible for the second step. It deploys the service or slice as is received from the tenant over IF2 and IF3. It then hands over the deployment decisions to the Assurance management subsystem which use this information to collect the periodic monitoring tickets in step 3. On the other end it is responsible for receiving the request over IF2 to deploy a part of the service, consult with the catalogue and topology management system on the translation of the abstraction to the received request and then for the actual deployment over the domains using IF3. The result of this deployment and the management and control handles to access the deployed slice or service is then returned to the requesting MPO.

The assurance management component periodically studies the collected results to report the overall service performance to the tenant or the requesting MPO as well as to ensure that the service is functioning within the specified parameters. In case of failures or violations the Assurance management system recommends reconfigurations to the NFVO.

6.1. Interworking interfaces

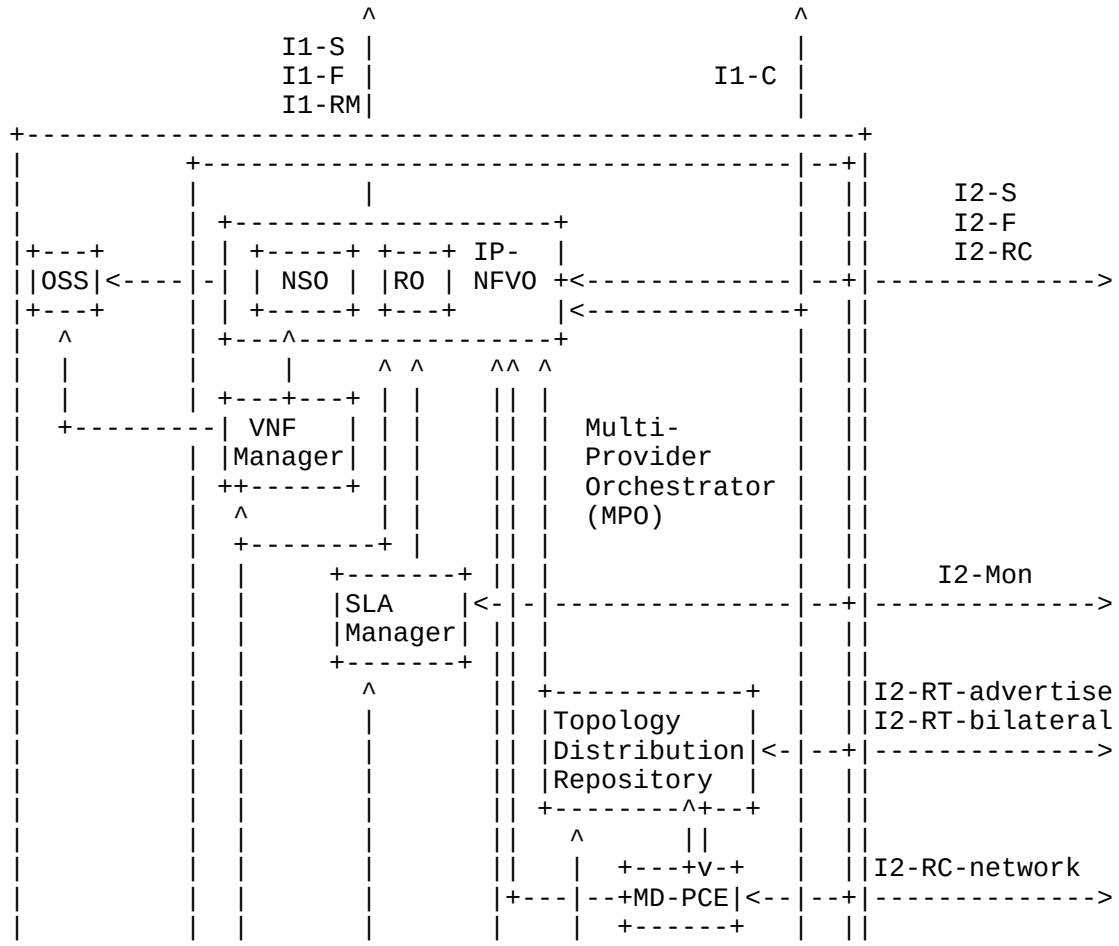
In this section we provide more details on the interworking interfaces of the MPO reference architecture. Each interface IF1, IF2 and IF3 is broken down into several sub-interfaces. Each of them has a clear scope and functionality.

For multi provider Network Service orchestration, the Multi-domain Orchestrator (MdO) offers Network Services by exposing an OSS/BSS - NFVO interface to other MPOs belonging to other providers. For multi-provider resource orchestration, the MPO presents a VIM-like view and exposes an extended NFVO - VIM interface to other MPOs. The MPO exposes a northbound sub-interface (IF1-S) through which an MPO customer sends the initial request for services. It handles command and control functions to instantiate network services. Such functions include requesting the instantiation and interconnection of Network Functions (NFs). A sub-interface IF2-S is defined to perform similar operations between MPOs of different administrative domains. A set of sub-interfaces -- IF3-R and IF2-R -- are used to keep an updated global view of the underlying infrastructure topology exposed by domain orchestrators. The service catalogue exposes available services to customers on a sub-interface IF1-C and to other MPO service operators on sub-interface IF2-C. Resource orchestration

related interfaces are broken up to IF2-RC, IF2-RT, IF2-RMon to reflect resource control, resource topology and resource monitoring respectively. Furthermore, the sub-interfaces introduced before are generalised and also used for interfaces IF3 and IF1.

6.2. 5GEx Multi Architecture

The 5G-PPP H2020 5GEx projects addresses the proposal and the deployment of a complete Multi-Provider Orchestrator providing, besides network and service orchestration, service exposition to other providers. The main assumptions of the 5GEx functional architecture are a) a multi-operator wholesale relationship, b) a full multi-vendor inter-operability and c) technology-agnostic approach for physical resources. The proposed functional architecture of the 5GEx MPO is depicted in Figure 13.



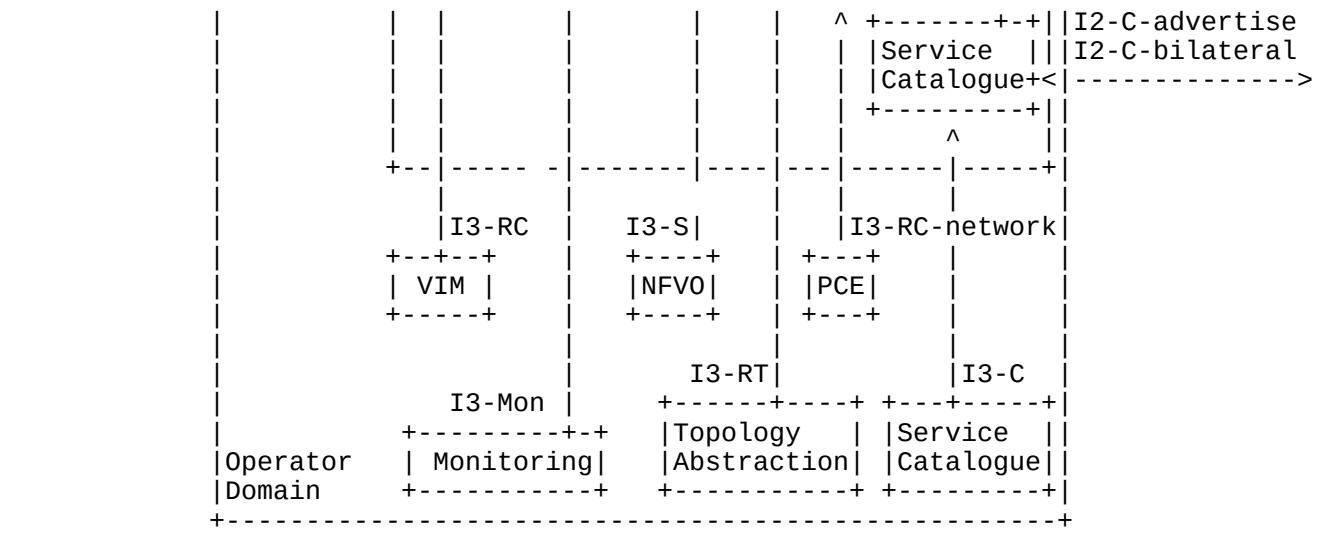


Figure 13: 5GEx MPO functional architecture

Providers expose MPOs service specification API allowing OSS/BSS or external business customers to perform and select their requirements for a service. Interface I1-x is exploited as a northbound API for business client requests. Peer MPO-MPO communications implementing multi-operator orchestration operate with specific interfaces referred to as I2-x interfaces. A number of I2-based interfaces are provided for communication between specific MPO modules: I2-S for service orchestration, I2-RC for network resource control, I2-F for management lifecycle, I2-Mon for inter-operator monitoring messages, I2-RT for resource advertisement, I2-C for service catalogue exchange, I2-RC-network for the QoS connectivity resource control. Some I2 interfaces are bilateral, involving direct relationship between two operators, and utilized to exchange business/SLA agreements before entering the federation of inter-operator orchestrators. Each MPO communicates through a set of southbound interface, I3-x, with local orchestrators/controllers/VIM, in order to set/modify/release resources identified by the MPO or during inter-MPO orchestration phase. A number of I3 interfaces are defined: I3-S for service orchestration towards local NFVO, I3-RC for resource orchestration towards local VIM, I3-C towards local service catalogue, I3-RT towards local abstraction topology module, I3-RC-network towards local PCE or network controller, I3-Mon towards local Resource Monitoring agent. All the considered interfaces are provided to cover either flat orchestration or layered/hierarchical orchestration. The possibility of hierarchical inter-provider MPO interaction is enabled at a functional level, e.g., in the case of

operators managing a high number of large administrative domains. The main MPO modules are the following:

- The Inter-provider NFVO, including the RO and the NSO, implementing the multi-provider service decomposition
- the VNF/Element manager, managing VNF lifecycle, scaling and responsible for FCAPS (Fault, Configuration, Accounting, Performance and Security management)
- the SLA Manager, in charge of reporting monitoring and performance alerts on the service graph
- the Service Catalogue, exposing available services to external client and operators
- the Topology and Resource Distribution module and Repository, exchanging operators topologies (both IT and network resources) and providing abstracted view of the own operator topology
- the Multi-domain Path Computation Element (PCE implementing inter-operator path computation to allow QoS-based connectivity serving VNF-VNF link).

The Inter-provider NVFO selects providers to be involved in the service chained request, according to policy-based decisions and resorting to Inter-Provider topologies and service catalogues advertised through interfaces I2-RT-advertise and I2-C-advertise, respectively. Network/service requests are sent to other providers using the I2-RC and I2-S interfaces, respectively. Policy enforcement for authorized providers running resource orchestration and lifecycle management are exploited through interfaces I2-RC and I2-F, respectively. The VNF/Element Manager is in charge of managing the lifecycle of the VNFs part of the services. More specifically, it is in charge to perform: the configuration of the VNFs, also in terms of security aspects, the fault recovery and the scaling according to their performance. The SLA Manager collects and aggregates quality measurement reports from probes deployed by the Inter-Provider NFVO as part of the service setup. Measurements results at the Manager represent aggregated results and are computed and stored utilizing the I2-Mon interface between Inter-Provider MPOs sharing the same service. Faults and alarms are moreover correlated to raise SLA violation to remote inter-provider MPOs and, optionally, to detect the source and the location of the violation, triggering service re-computation/rerouting procedures. The Service Catalogue stores information on network services and available VNFs and uses I2-C interfaces (either bilateral or advertised) to advertise and updating such offered services to other operators. To enable inter-

provider service decomposition, multi-operator topology and peering relationships need to be advertised. Providers advertise basic inter-provider topologies using the I2-RT-advertse interface including, optionally, abstracted network resources, overall IT resource capabilities, MPO entry-point and MD-PCE IP address. Basic advertisement takes place between adjacent operators. These information are collected, filtered by policy rules and propagated hop-by-hop. In 5GEx, the I2-RT-advertise interfaces utilizes BGP-LS protocol. Moreover, providers establish point-to-point bilateral (i.e., direct and exclusive) communications to exchange additional topology and business information, using the I2-RT-bilateral interface. Service decomposition may imply the instantiation of traffic-engineered multi-provider connectivity, subject to constraints such as guaranteed bandwidth, latency or minimum TE metric. The multi-domain PCE (MD-PCE) receives the connectivity request from the inter-provider NFVO and performs inter-operator path computation to instantiate QoS-based connectivity between two VNFs (e.g., Label Switched Paths). Two procedures are run sequentially:

operators/domain sequence computation, based on the topology database, provided by Topology Distribution module, and on specific policies (e.g., business, bilateral),

per-operator connectivity computation and instantiation.

In 5GEx, MD-PCE is stateful (i.e., current connectivity information is stored inside the PCE) and inter-operator detailed computation is performed resorting to the stateful Backward Recursive PCE-based computation (BRPC) [draft-stateful-BRPC], deploying a chain of PCEP sessions among adjacent operators, each one responsible of computing and deploying its segment. Backward recursive procedure allows optimal e2e constrained path computation results.

6.3. 5G-TRANSFORMER Architecture

5G-TRANSFORMER project proposes a flexible and adaptable SDN/NFV-based design of the next generation Mobile Transport Networks, capable of simultaneously supporting the needs of various vertical industries with diverse range of requirements by offering customized slices. In this design, multi-domain orchestration and federation are considered as the key concepts to enable end-to-end orchestration of services and resources across multiple administrative domains.

The 5G-TRANSFORMER solution consists of three novel building blocks, namely:

1. Vertical Slicer (VS) as the common entry point for all verticals into the system. The VS dynamically creates and maps the

vertical services onto network slices according to their requirements, and manages their lifecycle. It also translates the vertical and slicing requests into ETSI defined NFV network services (NFV-NS) sent towards the SO. Here a network slice is deployed as a NFV-NS instance.

2. Service Orchestrator (SO). It offers service or resource orchestration and federation, depending on the request coming from the VS. This includes all tasks related with coordinating and offering to the vertical an integrated view of services and resources from multiple administrative domains. Orchestration entails managing end-to-end services or resources that were split into multiple administrative domains based on requirements and availability. Federation entails managing administrative relations at the interface between SOs belonging to different domains and handling abstraction of services and resources.

3. Mobile Transport and Computing Platform (MTP) as the underlying unified transport stratum, responsible for providing the resources required by the NFV-NS orchestrated by the SO. This includes their instantiation over the underlying physical transport network, computing and storage infrastructure. It also may (de)abstract de MTP resources offered to the SO.

The 5G-TRANSFROMER architecture is quite in line with the general Multi Domain Multi Provider reference architecture depicted in Figure 11. Its mapping to the reference architecture is illustrated in the figure below.

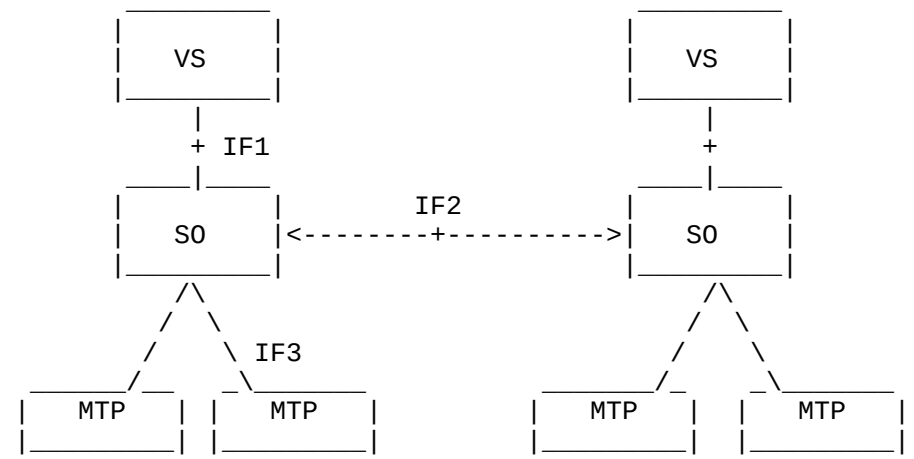


Figure 14: 5G-TRANSFORMER architecture mapped to the reference architecture

The MTP would be mapped to the individual domain orchestrators, which only provides the resource orchestration for the local administrative domain. The role of the SO is the Multi Provider orchestrator (MPO) responsible for multi-domain service or resource orchestration and federation. The operation of the SO follows three main technical steps handled by the three function components of the MPO shown in Figure 14, namely (i) the catalogue and topology management system; (ii) the NFVO (Network Function Virtualization Orchestrator); and the assurance management component.

Correspondingly, the interface between the SO and the VS (So-Vs) is the interface 1 (IF1), through which the VS requests the instantiation and deployment of various network services to support individual vertical service slices. The interface between the SOs (So-So) of different domains is the interface 2 (IF2), enabling multi domain orchestration and federation operations. The interface between the SO and the MTP (So-Mtp) is the interface 3 (IF3). It, on the one hand, provides the SO the updated global view of the underlying infrastructure topology abstraction exposed by the MTP domain orchestrators, while on the other hand it also handles command and control functions to allow the SO request each MTP domain for virtual resource allocation.

In 5G-TRANSFOMER, a set of sub-interfaces have been defined for the So-Mtp, So-So and Vs-So interfaces.

6.3.1. So-Mtp Interface (IF3)

This interface is based on ETSI GS-NFV IFA 005 and ETSI GS-NFV IFA 006 for the request of virtual resource allocation, management and monitoring. Accordingly, the 5G-TRANSFORMER identified the following sub-interfaces at the level of So-Mtp interactions (i.e., IF3-x interfaces regulating MPO-DO interactions).

So-Mtp(-RAM). It provides the Resource Advertisement Management (RAM) functions to allow updates or reporting about virtualized resources and network topologies in the MTP that will accommodate the requested NFVO component network services.

So-Mtp(-RM). It provides the Resource Management (RM) operations over the virtualized resources used for reserving, allocating, updating (in terms of scaling up or down) and terminating (i.e., release) the virtualized resources handled by each MTP and triggered by NFVO component (in Figure 14) to accommodate network services.

So-Mtp(-RMM). It provides the required primitives and parameters for supporting the SO resource monitoring management (RMM)

capability for the purpose of fault management and SLA assurance handled by assurance management component in Figure 14.

In the reference architecture (Fig. 6), the IF3-RC, IF3-RT, IF3-RMon sub-interface are defined for resource control, resource topology and resource monitoring respectively. The IF3-RT, IF3-RC and IF3-RMon sub-interfaces map to So-Mtp(-RAM), So-Mtp(-RM) and So-Mtp(-RMM) sub-interfaces from 5G-TRANSFORMER.

6.3.2. So-So Interface (IF2)

This interface is based ETSI GS-NFV IFA 013 and ETSI GS-NFV IFA 005 for the service and resource federation between the domains. The 5G-TRANSFORMER identified the following sub-interfaces at the level of So-So interactions (i.e., IF2-x interfaces regulating MPO interactions) to provide service and resource federation and enable NSaaS and NFVIaaS provision, respectively, across different administrative domains.

So-So(-LCM), for the operation of NFV network services. The reference point is used to instantiate, terminate, query, update or re-configure network services or receive notifications for federated NFV network services. The SO NFVO-NSO uses this reference point.

So-So(-MON), for the monitoring of network services through queries or subscriptions/notifications about performance metrics, VNF indicators and network service failures. The SO NFVO-NSO uses this reference point.

So-So(-CAT), for the management of Network Service Descriptors (NSDs) flavors together with VNF/VA and MEC Application Packages, including their Application Descriptors (AppDs). This reference point offers primitives for on-boarding, removal, updates, queries and enabling/disabling of descriptors and packages. The SO NFVO-NSO uses this reference point.

Furthermore, resource orchestration related operations are broken up to the following sub-interfaces to reflect resource control, resource topology and resource monitoring respectively.

So-So(-RM), for allocating, configuring, updating and releasing resources. The Resource Management reference point offers operations such as configuration of the resources, configuration of the network paths for connectivity of VNFs. These operations mainly depend of the level of abstraction applied to the actual resources. The SO NFVO-RO uses this reference point.

So-So(-RMM), for monitoring of different resources, computing power, network bandwidth or latency, storage capacity, VMs, MEC hosts provided by the peering administrative domain. The details level depends on the agreed abstraction level. The SO NFVO-RO uses this reference point.

So-So(-RAM), for advertising available resource abstractions to/ from other SOs. It broadcasts available resources or resource abstractions upon capability calculation and periodic updates for near real-time availability of resources. The SO-SO Resource Advertisement uses this reference point.

So-So(-RMM), for monitoring of different resources, computing power, network bandwidth or latency, storage capacity, VMs, MEC hosts provided by the peering administrative domain. The details level depends on the agreed abstraction level. The SO NFVO-RO uses this reference point.

In the reference architecture (Figure 11), the sub-interface IF2-S and IF2-C are defined to perform network service-related operations between MPOs of different administrative domains. The IF2-RC, IF2-RT, IF2-RMon sub-interfaces are defined to regulated interactions between Catalogue and Topology Management components. Their mapping to the sub-interfaces defined in 5G-TRANSFORMER are summarized as follows:

The IF2-S sub-interface maps to So-So(-LCM) and So-So(-MON).

The IF2-C sub-interface maps to So-So(-CAT).

The IF2-RC, IF2-RT, IF2-RMon sub-interfaces map to So-So-RM, So-So-RAM, So-So-RT respectively.

6.3.3. Vs-So Interface (IF1)

This interface is based on ETSI GS-NFV IFA 013 for the VS requesting network services from the SO. Accordingly, the 5G-TRANSFORMER identified the following sub-interfaces at the level of Vs-So interactions (i.e., IF1-x interfaces regulating tenant-MPO interactions).

Vs-So(-LCM). It deals with the NFV network service lifecycle management (LCM) and it is based on the IFA 013 NS Lifecycle Management Interface. It offers primitives to instantiate, terminate, query, update or re-configure network services or receive notifications about their lifecycle.

Vs-So(-MON). It deals with the monitoring (MON) of network services and VNFs through queries or subscriptions and notifications about performance metrics, VNF indicators and network services or VNFs failures. It maps to IF1-S sub-interface of the reference architecture.

Vs-So(-CAT). It deals with the catalogue (CAT) management of Network Service Descriptors (NSDs), VNF packages, including their VNF Descriptors (VNFDs), and Application Packages, including their Application Descriptors (AppDs). It offers primitives for on-boarding, removal, updates, queries and enabling/disabling of descriptors and packages. It maps to IF1-C sub-interface of the reference architecture.

In the reference architecture (Figure 11), the sub-interface IF1-S and IF1-C are defined to build request to perform network service-related operations including requesting the instantiation, update and termination of the requested network services. The IF1-S sub-interface maps to Vs-So(-LCM) and Vs-So(-MON), while the IF1-C sub-interface maps to Vs-So(-CAT) defined in 5G-TRANSFORMER architecture.

7. Multi-domain orchestration and Open Source

Before reviewing current state of the open source projects it should be explicitly mentioned that term "federation" is quite ambiguous and used in multiple contexts across the industry. For example, federation is the approach used at certain software projects to achieve high availability and enable reliable non-interrupted operation and service delivery. One of the distinguishing features of this federation type is that all federated instances are managing the same piece of the infrastructure or resources set. However, this document is focused on another federation type, where multiples independent instances of the orchestration/management software establish certain relationships and expose available resources and capabilities in the particular domain to consumers at another domain. Besides sharing resource details, multi-domain federation requires various management information synchronization, such authentication/ authorization data, run-time policies, connectivity details and so on. This kind of functionality and appropriate implementation approaches at the relevant open source projects are in scope of current section.

At this moment several open source industry projects were formed to develop integrated NFV orchestration platform. The most known of them are ONAP [onap], OSM [osm] and Cloudify [cloudify]. While all these projects have different drivers, motivations, implementation approach and technology stack under the hood, all of them are considering multi-VIM deployment scenario, i.e. all these software

platforms are capable to deploy NFV service over different virtualized infrastructures, like public or private providers. Additionally OSM and Cloudify orchestration platforms have capabilities to manage interconnection among managed VIMs using appropriate plugins or drivers. However, despite the fact that typical Telco/Carrier infrastructure has multiple domains (both technology and administrative), none of these orchestration projects is focused on a service federation use case development.

In the meantime, as an acknowledgement of the challenges, emerged during exploitation of the federation use cases Multisite project emerged under OPNFV umbrella [opnfv]. Considering OpenStack-based VIM deployments spanned across multiple regions as a general use case, this project initially was focusing on a gaps identification in the key OpenStack projects which lacks capabilities for multi-site deployment. During several development phases of this OPNFV project, number of gaps were identified and submitted as a blueprints for the development into the appropriate OpenStack projects. Further several demo scenarios were delivered to trial OpenStack as the open source VIM which is capable to support multisite NFV clouds. While Multisite OPNFV project was focusing on a resource and VIM layer only, there are multiple viable outputs which might be considered during implementation of the federation use cases on the upper layers.

As a summary it can be stated that it is still early days for the technology implemented in a referenced NFV orchestration projects and federation use case in not on a radar for these projects for the moment. However, it is expected that upon maturity of the federation as a viable market use case appropriate feature set in the reviewed projects will be developed.

8. IANA Considerations

N/A.

9. Security Considerations

TBD.

10. Acknowledgments

This work is supported by 5G-PPP 5GEx, an innovation action project partially funded by the European Community under the H2020 Program (grant agreement no. 671636). This work is also supported by 5G-PPP 5G-TRANSFORMER, a research and innovation action project partially funded by the European Community under the H2020 Program (grant agreement no. 761536). The views expressed here are those of the

authors only. The European Commission is not liable for any use that may be made of the information in this presentation.

11. Informative References

[cloudify] "Cloudify", <https://cloudify.co/>.

[etsi_nvf_ifa009] "Report on Architectural Options, ETSI GS NFV-IFA 009 V1.1.1", July 2016.

[etsi_nvf_ifa028] "Report on architecture options to support multiple administrative domains, ETSI GR NFV-IFA 028 V3.1.1", January 2018.

[etsi_nvf_whitepaper] "Network Functions Virtualisation (NFV). White Paper 2", October 2014.

[etsi_nvf_whitepaper_5g] "Network Functions Virtualisation (NFV). White Paper on "Network Operator Perspectives on NFV priorities for 5G"", February 2017.

[ngmn_5g_whitepaper] "5G White Paper", February 2015.

[ngmn_slicing] "Description of Network Slicing Concept", January 2016.

[onap] "ONAP project", <https://www.onap.org/>.

[opnfv] "OPNFV Multisite project", <https://wiki.opnfv.org/display/multisite/Multisite>.

[osm] "Open Source MANO project", <https://osm.etsi.org/>.

Authors' Addresses

Carlos J. Bernardos (editor)
Universidad Carlos III de Madrid
Av. Universidad, 30
Leganes, Madrid 28911
Spain

Phone: +34 91624 6236
Email: cjbc@it.uc3m.es
URI: http://www.it.uc3m.es/cjbc/

Luis M. Contreras
Telefonica I+D
Ronda de la Comunicacion, S/N
Madrid 28050
Spain

Email: luismiguel.conterasmurillo@telefonica.com

Ishan Vaishnavi
Huawei Technologies Dusseldorf GmBH
Riesstrasse 25,
Munich 80992
Germany

Email: Ishan.vaishnavi@huawei.com

Robert Szabo
Ericsson
Konyves Kaman krt. 11
Budapest, EMEA 1097
Hungary

Phone: +36703135738
Email: robert.szabo@ericsson.com

Josep Mangues-Bafalluy
CTTC
Av. Carl Friecrish Gauss, 7
Castelldefels, EMEA 08860
Spain

Email: josep.mangues@cttc.cat

Xi Li
NEC
Kurfuersten-Anlage 36
Heidelberg 69115
Germany

Email: Xi.Li@neclab.eu

Francesco Paolucci
SSSA
Via Giuseppe Moruzzi, 1
Pisa 56121
Italy

Phone: +395492124
Email: fr.paolucci@santannapisa.it

Andrea Sgambelluri
SSSA
Via Giuseppe Moruzzi, 1
Pisa 56121
Italy

Phone: +395492132
Email: a.sgambelluri@santannapisa.it

Barbara Martini
SSSA
Via Giuseppe Moruzzi, 1
Pisa 56121
Italy

Email: barbara.martini@cnit.it

Luca Valcarenghi
SSSA
Via Giuseppe Moruzzi, 1
Pisa 56121
Italy

Email: luca.valcarenghi@santannapisa.it

Giada Landi
Nextworks
Via Livornese, 1027
Pisa 56122
Italy

Email: g.landi@nextworks.it

Dmitriy Andrushko
MIRANTIS

Email: dandrushko@mirantis.com

Alain Mourad
InterDigital Europe

Email: Alain.Mourad@InterDigital.com
URI: http://www.InterDigital.com/

Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 18, 2017

X. Cai
C. Meirosu
Ericsson
G. Mirsky

November 14, 2016

Recursive Monitoring Language in Network Function Virtualization (NFV) Infrastructures
draft-cai-nfvrg-recursive-monitor-03

Abstract

Network Function Virtualization (NFV) poses a number of monitoring challenges; one potential solution to these challenges is a recursive monitoring language. This document presents a set of requirements for such a recursive monitoring language.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 18, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 2
1.1. Conventions used in this document 3
1.1.1. Terminology 3
1.1.2. Requirements Language 4
2. Requirements towards NFV Monitoring Language 4
3. Sample Use Cases 4
4. Overview of the Recursive Language 5
5. Formal Syntax 8
6. Requirements for Using the Language 9
7. Sample Query Scripts 9
7.1. Query End to End Delay Between Network Functions 9
7.2. Query the CPU Usage of Network Functions 10
8. IANA Considerations 11
9. Security Considerations 11
10. Acknowledgements 12
11. References 12
11.1. Normative References 12
11.2. Informative References 12
Authors' Addresses 13

1. Introduction

This document discusses a recursive monitoring query language to support monitoring real-time properties of NFV infrastructures (e.g., defined in ETSI [ETSI-ARC] and UNIFY [I-D.unify-nfvrg-recursive-programming]). A network service can be constructed of Virtual Network Functions (VNFs) or Physical Network Functions (PNFs) interconnected through a Network Function Forwarding Graph (VNFFG). A single VNF, in turn, can consist of interconnected elements; in other words, VNFFGs can be nested.

Service operators and developers are interested in monitoring the performance of a service contained within an VNFFG (as above) or any part of it. For example, an operator may want to measure the CPU or memory usage of an entire network service and the network delay cross a VNF which consists of multiple VMs, instead of only individual virtual or physical entities.

In existing systems, this is usually done by mapping the performance metrics of VNFs to primitive network functions or elements, statically and manually when the virtualized service is deployed. However, in the architecture defined in ETSI [ETSI-ARC] and UNIFY [I-D.unify-nfvrg-recursive-programming] a multi-layer hierarchical

architecture is adopted, and the VNF and associated resources, expressed VNFFGs, may be composed recursively in different layers of the architecture. This will pose greater challenges for performance queries for a specific service, as the mapping of performance metrics from the service layer (highest layer) to the infrastructure (lowest layer) is more complex than an infrastructure with a single layer of orchestration. We argue that it is important to have an automatic and dynamic way to decompose performance queries in this environment in a recursive way, following the different abstraction levels expressed in the NF-NFs at hierarchical architecture layers. Hence, we propose using a declarative language such as Datalog [Green-2013] to perform recursive queries based on input in form of the resource graph depicted as VNFFG. By reusing the VNFFG models and monitoring database already deployed in NFV infrastructure, the language can hide the complexity of the multilayer network architecture with limited extra effort and resources. Even for single layer NFV architectures, using such language can simplify performance queries and enable a more dynamic performance decomposition and aggregation for the service layer.

Recursive query languages can support many DevOps [I-D.unify-nfvrg-devops] processes, most notably observability and troubleshooting tasks relevant for both operators and developer roles, e.g. for high-level troubleshooting where various information from different sources need to be retrieved. Additionally, the query language might be used by specific modules located in the control and orchestration layers, e.g. a module realizing infrastructure embedding of VNFFGs might query monitoring data for an up-to-date picture of current resource usage. Also scaling modules of specific network functions might take advantage of the flexible query engine pulling of monitoring information on demand (e.g. resource usage, traffic trends, etc.), as complement to relying on devices and/or elements to push this information based on pre-defined thresholds.

1.1. Conventions used in this document

1.1.1. Terminology

ETSI - European Telecommunication Standards Institute

VNFFG - Network Function Forwarding Graph

NFV - Network Function Virtualization

PNF - Physical Network Function

SG - Service Graph

VNF - Virtual Network Function

1.1.2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Requirements towards NFV Monitoring Language

Following are the requirements for a language to express constructs and actions of monitoring NFV infrastructures:

- The network service MAY consist of VNFs which contain inter-connected elements and be described by nested VNFFGs. The language MUST support recursive query.

- The language is used by the service operators or developers to monitor the high-level performance of the network service. Declarative language could provide better description on the monitoring task rather than the procedure and imperative language. The language MUST be declarative.

3. Sample Use Cases

In Figure 1, the Service Graph (SG) and corresponding VNFFGs of a network service is illustrated. The service consists of two Network Functions NF1 and NF2, which consists of (VNF1-1, VNF1-2) and (VNF2-1, VNF2-2) respectively. In VNF1-1 and VNF2-2 there are recursively nested VNFs VNF1-3 and VNF2-3.

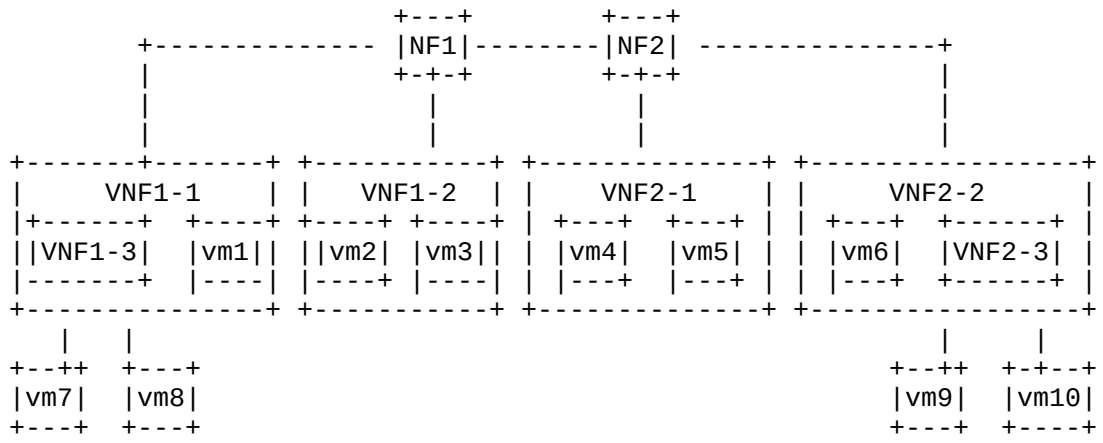


Figure 1: The sample VNFFG of network service

Two use cases of the recursive monitoring query are described below.

First, consider the use case where an operator of the network service wants to query the end to end delay from network function NF1 to Network Function NF2 in the service graph. Here the end to end delay of two network functions are defined as the delay between ingress node of source network function and egress node of destination network function. After running a querying script, the delay between NF1 and NF2 in service layer should be mapped recursively to the delay between two specific virtual machines (vm7 and vm10) in the NFV infrastructure.

Second, consider the use case where an operator wants to measure the CPU usage of network function NF1 in order to dynamically scale in/ out this function. Several types of CPU usage of a network function can be defined. For example, average CPU usage is the average value of measured CPU usage of all nodes belongs to the network function. Maximum CPU usage is the measured usage of the node that has the highest CPU load. To get either the average or maximum CPU usage, the query language to recursively identify all nodes (i.e., vm1, vm2, vm3, vm7 and vm8) of NF1, then retrieve the measured CPU usage of these nodes from somewhere and return the mean or maximum value to the operator.

4. Overview of the Recursive Language

In this section we describe the recursive monitoring language. The query language proposed here is based on Datalog, which is a declarative logic programming language that provides recursive query

capability. The simple and clear semantics of Datalog allow better query sepcifiation, understanding and maintenance. In addition, the neat formuation of its recursive query makes it fit well in the recursive based architecture. Datalog has been successfully used in cloud computing in recent years, e.g., the OpenStack [OpenStack] policy engine Congress [OpenStack-Congress]. In addition, there are many open source or commerial Datalog interpreter avaible now, e.g., python based pyDatalog [pyDatalog], java based IRIS [IRIS], LogicBlox [LogicBlox], and etc.

As like other Datalog based language, the recursive monitoring query program consists of a set of declarative Datalog rules and a query. A rule has the form:

```
h <= p1, p2, ..., pn
```

which can be defined as "p1 and p2 and ... and pn implies h". "h" is the head of the rule, and "p1, p2, ..., pn" is a list of literals that constitutes the body of the rule. Literals "p(x1, ..., xi, ..., xn)" are either predicates applied to arguments "xi" (variables and constants), or function symbols applied to arguments. The program is said to be recursive if a cycle exists through the predicates, i.e., predicate appearing both in the head and body of the same rule. The order in which the rules are presented in a program is semantically irrelevant. The commas separating the predicates in a rule are logical conjuncts (AND); the order in which predicates appear in a rule body has no semantic significance, i.e. no matter in what order rules been processed, the result is atomic, i.e. the same. The names of predicates, function symbols and constants begin with a lower-case letter, while variable names begin with an upper-case letter. A variable appearing in the head is called distinguished variable while a variable appearing in the body is called non-distinguished variable. The head is true of the distinguished variables if there exist values of the non-distinguished variables that make all sub goals of the body true. In every rule, each variable stands for the same value. Thus, variables can be considered as placeholders for values. Possible values are those that occur as constants in some rule/fact of the program itself. In the program, a query is of the form "query(m, y1, ..., yn)", in which "query" is a predicate contains arguments "m" and "yi". "m" represents the monitoring function to be queried, e.g., end to end delay, average CPU usage, and etc. "yi" is the arguments for the query function. The meaning of a query given a set of Datalog rules and facts is the set of all facts of query() that are given or can be inferred using the rules in the program. The predicates can be divided into two categories: extensional database predicates (EDB predicates), which contain ground facts, meaning it only has constant arguments; and intentional

database predicates (IDB predicates), which correspond to derived facts computed by Datalog rules.

In order to perform a recursive monitoring query, the resource graph described in the VNFFG needs be transformed so it is represented as a set of Datalog ground facts which are used by the rules in the program. The following keywords can be defined to represent the VNFFG graph into Datalog facts, which are then used in the query scripts:

sub(x, y) which represents 'y'; is an element of the directly descend sub-layer of 'x';

link(x, y) which represents that there is a direct link between elements 'x' and 'y';

node(z) which represents a node in VNFFG.

If the NFV environment adopts standardized specification or templates to define the VNFs and their connectivity graph, e.g., OASIS TOSCA [TOSCA-Simple-Profile-NFV-v1.0] and ETSI NFV MANO [ETSI-ARC], various keywords shall be defined to convert constructs included in these specifications or templates into Datalog facts.

For example, the Organization for the Advancement of Structured Information Standards (OASIS) has defined a simple profile for NFV with TOSCA [TOSCA-Simple-Profile-NFV-v1.0], an language to describe the topology and orchestration of cloud applications. TOSCA NFV data model supports layered structures. A top layer template is Network Service Description (NSD) which contains the templates of VNFD (VNF Descriptors), VLD (Virtual Link Descriptor), VNFFGD (VNF Forwarding Graph Descriptor), etc., which may also contain substitution templates. For example VNFD is composed of templates of VDU (Virtualization Deployment Unit), VLD, etc. For TOSCA NFV profile, the following keywords could be defined as an example:

node(id, type) which represents a node (e.g., VNF, PNF) in NSD with given 'id' and 'type';

sub(x, y) which represents a node 'y' belongs to a substitution template. For example, if a VNF (vnf1) node contains one VDU (vdu1) and one Connection Point (CP) (cp11), it can be denoted as 'sub(vnf1, vdu1)' and 'sub(vnf1, cp11)';

vbind(x, y) which represents a node 'y' (e.g., CP) is associated with the node 'x' (e.g., VDU);

vlink(x, y) which denotes a Connection Points 'y' belongs to Virtual Link 'x'. CP represents the virtual and/or physical interfaces of the VNFs in TOSCA NFV profile;

forwarding path(p1 , p2 , ..., pn) represents a network forwarding path which consists of an ordered list of CPs.

In addition, a set of functions calls can be defined in order to support the monitoring query. The function call will start with "fn_"; in the syntax and may include 'boolean' predicates, arithmetic computations and some other simple operation. The function calls can be provided by the query engine or developers.

If the sub-VNFFGs of a network service are provided by different NFV infrastructure providers and not available to the provider who attempts to measure some aspect of the VNFFG due to some reason, e.g., security, additional extensions to the language and query engine would be required (this is called a distributed query). This scenario is not considered in this draft and is left for further study.

5. Formal Syntax

The following syntax specification describes the Datalog based reursive monitoring language and uses the augmented Backus-Naur Form (BNF) as described in [RFC2234].

```
<program>         ::= <statement>*
<statement>       ::= <rule> | < fact>
<rule>            ::= [<rule-identifier>] <head> <= <body>
<fact>            ::= [<fact-identifier>]<clause> |
                       <fact_predicate>(<terms>)
<head>            ::= <clause>
<body>            ::= <clause>
<clause>          ::= <atom> | <atom>, <clause>
<atom>            ::= <predicate> ( <terms> )
<predicate>       ::= <lowercase-letter><string>
<fact_predicate>  ::= ("sub"; | "node" |
                       "link")( <terms> )
<terms>           ::= <term> | <term>, <terms>
<term>            ::= <VARIABLE> | <constant>
<constant>        ::= <lowercase-letter><string>
<VARIABLE>        ::= <Uppercase-letter><string>
<fact-identifier> ::= "F"<integer>
<rule-identifier> ::= "R"<integer>
```

6. Requirements for Using the Language

To utilize the recursive monitoring language a query engine has to be deployed into NFV infrastructure. Some basic functions are required for the query engine.

The query engine MUST provide the capability to parse and interpret the query scripts which are written with the language.

The query engine MUST be able to retrieve the VNFFG created by NFV infrastructure and translate them into Datalog based ground facts.

The query engine MUST be able to query the database in which the monitoring results of primitive metric are stored.

An interface between query engine and the users of the language (e.g., developer or network service operator) MUST be defined to exchange the query scripts and query results.

7. Sample Query Scripts

According to the defined language, the sample query scripts for the above mentioned use cases are illustrated in this section. Some example query scripts are illustrated in this section.

7.1. Query End to End Delay Between Network Functions

Two kinds of delay between network functions are discussed here: end-to-end delay and hop-by-hop delay. Here end to end delay is defined as the delay between the ingress node in the lowest layer of the source network function and the egress node in the lowest layer of the destination network function. And the hop by hop delay is defined as the aggregation of the delay of each segment which consists of the path from the source to the destination network function.

The scripts to query the end to end delay from NF1 to NF2 as illustrated in Figure 1 contains both the ground facts and IDB predicates:

```
F1: sub(NF1, VNF1-1, VNF1-2), sub(NF2, VNF2-1, VNF2-2),
sub(VNF1-1, VNF1-3, vm1), sub(VNF1-2, vm2, vm3),
sub(VNF1-3, vm7, vm8), sub(VNF2-1, vm4, vm5),
sub(VNF2-2, vm6, VNF2-3), sub(VNF2-3, vm9, vm10)
F2: link(NF1, NF2), link (VNF1-3, vm1), link(vm2, vm3),
link(vm3, vm4), link(vm4,vm5), link(vm5,vm6),
link(vm6, VNF2-3), link(vm7, vm8), link(vm9, vm10)
R1: child(X,Y) <= sub(X,Z), child(Z,Y)
R2: child(X,Y) <= sub(X,Y)
R3: leaf(X,Y) <= child(X,Y), ~sub(Y,Z)
R4: in_leaf(X, Y) <= leaf(X, Y) & ~link(M, Y)
R5: out_leaf(X, Y) <= leaf(X, Y) & ~link(Y, M)
R6: e2e_delay(S,D,P) <= link(S,D), P == f_e2e_delay(in_leaf(S,Y),
out_leaf(D,Z))
query(e2e_delay, NF1, NF2)
```

F1-F2 are used to translate the VNFFG in Figure x into ground facts. R1-R5 are used to traversal the VNFFG recursively to get the ingress node of VNF1 and egress node of VNF2. R1-R2 can recursively traverse the graphs and determine all child nodes (i.e., VNF1-1, VNF1-3, VNF1-2, vm1, vm2, vm3, vm7, vm8, VNF2-1, VNF2-2, VNF2-3, vm4, vm5, vm6, vm9, vm10 in Figure 1). R3 is used to find out all leaf nodes (i.e., virtual machines). In the example, they include all virtual machines. R4 and R5 are used to get the ingress and egress nodes of NF1 and NF2 respectively, i.e., vm7 and vm10. In R6 the delay for a given source and destination network functions is measured by function f_e2e_delay. R1-R6 can be stored into a library of the query engine as a template e2e_delay, so that the users only need to send a simple query request, e.g. e2e_delay NF1 NF2, to the query engine to measure the end to end delay between NF1 and NF2.

The recursion is controlled by the Datalog engine. It combines the F1-2 rules (i.e., the ground facts) to conceptually build, internally, a multi-rooted tree structure indicating the relationships between the different elements in the VNFFG. It then uses R1-3 as the primary means to determine how to traverse this tree. The R4-5 rules are special in the sense that they allow selecting very specific leafs of the tree. Recursivity in this context refers to the capability to automatically traverse components of the VNFFG located at different hierarchical levels in a NFV architecture.

7.2. Query the CPU Usage of Network Functions

A set of example scripts to query the CPU usage (maximum and average usage) of a given network function are included below:

```
F1: sub(NF1, VNF1-1, VNF1-2), sub(NF2, VNF2-1, VNF2-2),
sub(VNF1-1, VNF1-3, vm1), sub(VNF1-2, vm2, vm3),
sub(VNF1-3, vm7, vm8), sub(VNF2-1, vm4, vm5),
sub(VNF2-2, vm6, VNF2-3), sub(VNF2-3, vm9, vm10)
R1: child(X,Y) <= sub(X,Z), child(Z,Y)
R2: child(X,Y) <= sub(X,Y)
R3: leaf(X,Y) <= child(X,Y), ~sub(Y,Z)
R4: max_cpu(X,C) <= leaf(X,Y), C == f_max_cpu(leaf(X,Y))
R5: mean_cpu (X,C) <= leaf(X, Y), C == f_mean_cpu(leaf(X,Y))
Query(max_cpu, NF1)
```

F1 is used to translate the VNFFG in Figure x into ground facts. R1-R3 are used to traversal the VNFFG recursively to get all child nodes of NF1 in Figure x. R1-R2 recursively traversal the graphs and figure out all child nodes of NF1(i.e., VNF1-3, vm1, vm2, vm3, vm7, vm8). R3 is used to figure out all leaf nodes of NF1(i.e., vm1, vm2, vm3, vm7, vm8). In R4, the maximum CPU usage is calculated by function f_max_cpu. In R6, the average CPU usage is calculated by function f_mean_cpu.

Here only the query scripts for network delay and CPU usage are illustrated. But the language can also be applied to other performance metrics like throughput.

More advanced queries are possible by defining them in the template library. Related to the examples presented above, such a function could determine not only the maximum but the TopN CPU usage values for a particular type of VNF instances, or for the all the VNF instances that are part of a chain, and also return the identifiers of the VNF instances that generated these values. The results could be used as input to the orchestration that could in turn decide how to address a particular situation (for example, by consolidating the bottom M lightly used instances, or by scaling some of the TopN utilized instances).

8. IANA Considerations

TBD

9. Security Considerations

TBD

10. Acknowledgements

Authors deeply appreciate thorough review and insightful comments by Russ White.

11. References

11.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

11.2. Informative References

[ETSI-ARC] "Architectural Framework v1.1.1", ETSI , October 2013.

[Green-2013] Green, T., Huang, S., Zhou, W., and B. Loo, "Datalog and Recursive Query Processing", Foundations and Trends in Databases Vol. 5, No. 2, November 2013.

[I-D.unify-nfvrg-devops] Meirosu, C., Manzalini, A., Steinert, R., Marchetto, G., Papafili, I., Pentikousis, K., and S. Wright, "DevOps for Software-Defined Telecom Infrastructures", draft-unify-nfvrg-devops-03 (work in progress), October 2015.

[I-D.unify-nfvrg-recursive-programming] Szabo, R., Qiang, Z., and M. Kind, "Towards recursive virtualization and programming for network and cloud resources", draft-unify-nfvrg-recursive-programming-02 (work in progress), October 2015.

[IRIS] "IRIS Reasoner (online)", http://www.iris-reasoner.org/ .

[LogicBlox] "LogicBlox (online)", http://www.logicblox.com/ .

[OpenStack] "OpenStack (online)", http://www.openstack.org/ .

[OpenStack-Congress] "OpenStack Congress (online)", https://wiki.openstack.org/wiki/Congress .

[pyDatalog]
"pyDatalog (online)", https://sites.google.com/site/pydatalog/ .

[RFC2234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, DOI 10.17487/RFC2234, November 1997, <http://www.rfc-editor.org/info/rfc2234>.

[TOSCA-Simple-Profile-NFV-v1.0]
"TOSCA simple profile for network functions virtualization (NFV) version 1.0", ETSI , November 2016.

Authors' Addresses

Xuejun Cai
Ericsson

Email: xuejun.cai@ericsson.com

Catalin Meirosu
Ericsson

Email: catalin.meirosu@ericsson.com

Greg Mirsky

Email: gregimirsky@gmail.com>

NFVRG
Internet-Draft
Intended status: Informational
Expires: March 6, 2019

CJ. Bernardos
UC3M
A. Rahman
InterDigital
JC. Zuniga
SIGFOX
LM. Contreras
TID
P. Aranda
UC3M
P. Lynch
Ixia

September 2, 2018

Network Virtualization Research Challenges
draft-irtf-nfvrg-gaps-network-virtualization-10

Abstract

This document describes open research challenges for network virtualization. Network virtualization is following a similar path as previously taken by cloud computing. Specifically, cloud computing popularized migration of computing functions (e.g., applications) and storage from local, dedicated, physical resources to remote virtual functions accessible through the Internet. In a similar manner, network virtualization is encouraging migration of networking functions from dedicated physical hardware nodes to a virtualized pool of resources. However, network virtualization can be considered to be a more complex problem than cloud computing as it not only involves virtualization of computing and storage functions but also involves abstraction of the network itself. This document describes current research and engineering challenges in network virtualization including guaranteeing quality-of-service, performance improvement, supporting multiple domains, network slicing, service composition, device virtualization, privacy and security, separation of control concerns, network function placement and testing. In addition, some proposals are made for new activities in IETF/IRTF that could address some of these challenges. This document is a product of the Network Function Virtualization Research Group (NFVRG).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 6, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction and scope 3
2. Terminology 4
3. Background 5
3.1. Network Function Virtualization 5
3.2. Software Defined Networking 7
3.3. ITU-T functional architecture of SDN 12
3.4. Multi-access Edge Computing 13
3.5. IEEE 802.1CF (OmniRAN) 14
3.6. Distributed Management Task Force 14
3.7. Open Source initiatives 14
4. Network Virtualization Challenges 16
4.1. Introduction 16
4.2. Guaranteeing quality-of-service 16
4.2.1. Virtualization Technologies 17
4.2.2. Metrics for NFV characterization 17
4.2.3. Predictive analysis 18
4.2.4. Portability 19
4.3. Performance improvement 19
4.3.1. Energy Efficiency 19

4.3.2. Improved link usage 20
4.4. Multiple Domains 20
4.5. 5G and Network Slicing 21
4.5.1. Virtual Network Operators 22
4.5.2. Extending Virtual Networks and Systems to the Internet of Things 23
4.6. Service Composition 24
4.7. End-user device virtualization 25
4.8. Security and Privacy 26
4.9. Separation of control concerns 27
4.10. Network Function placement 28
4.11. Testing 28
4.11.1. Changes in methodology 28
4.11.2. New functionality 30
4.11.3. Opportunities 31
5. Technology Gaps and Potential IETF Efforts 31
6. NFVRG focus areas 32
7. IANA Considerations 33
8. Security Considerations 33
9. Acknowledgments 33
10. Informative References 34
Authors' Addresses 39

1. Introduction and scope

The telecommunications sector is experiencing a major revolution that will shape the way networks and services are designed and deployed for the next few decades. In order to cope with continuously increasing demand and cost, network operators are taking lessons from the IT paradigm of cloud computing. This new approach of virtualizing network functions will enable multi-fold advantages by moving communication services from bespoke hardware in the operator's core network to Commercial off-the-shelf (COTS) equipment distributed across datacenters.

Some of the network virtualization mechanisms that are being considered include: sharing of network infrastructure to reduce costs, virtualization of core and edge servers/services running in data centers as a way of supporting their load-aware elastic dimensioning, and dynamic energy policies to reduce the electricity consumption.

This document presents research and engineering challenges in network virtualization that need to be addressed in order to achieve these goals, spanning from pure research and engineering/standards space. The objective of this memo is to document the technical challenges and corresponding current approaches and to expose requirements that should be addressed by future research and standards work.

This document represents the consensus of the NFV Research Group. It has been reviewed by the Research Group members active in the specific areas of work covered by the document.

2. Terminology

The following terms used in this document are defined by the ETSI Network Function Virtualization (NVF) Industrial Study Group (ISG) [etsi_gs_nfv_003], the ONF [onf_tr_521] and the IETF [RFC7426] [RFC7665]:

Application Plane - The collection of applications and services that program network behavior.

Control Plane (CP) - The collection of functions responsible for controlling one or more network devices. CP instructs network devices with respect to how to process and forward packets. The control plane interacts primarily with the forwarding plane and, to a lesser extent, with the operational plane.

Forwarding Plane (FP) - The collection of resources across all network devices responsible for forwarding traffic.

Management Plane (MP) - The collection of functions responsible for monitoring, configuring, and maintaining one or more network devices or parts of network devices. The management plane is mostly related to the operational plane (it is related less to the forwarding plane).

NFV Infrastructure (NFVI): totality of all hardware and software components which build up the environment in which VNFs are deployed.

NFV Management and Orchestration (NFV-MANO): functions collectively provided by NFVO, VNFM, and VIM.

NFV Orchestrator (NFVO): functional block that manages the Network Service (NS) lifecycle and coordinates the management of NS lifecycle, VNF lifecycle (supported by the VNFM) and NFVI resources (supported by the VIM) to ensure an optimized allocation of the necessary resources and connectivity.

Operational Plane (OP) - The collection of resources responsible for managing the overall operation of individual network devices.

Physical Network Function (PNF): Physical implementation of a Network Function in a monolithic realization.

Service Function Chain (SFC): for a given service, the abstracted view of the required service functions and the order in which they are to be applied. This is somehow equivalent to the Network Function Forwarding Graph (NF-FG) at ETSI.

Service Function Path (SFP): the selection of specific service function instances on specific network nodes to form a service graph through which an SFC is instantiated.

Virtualized Infrastructure Manager (VIM): functional block that is responsible for controlling and managing the NFVI compute, storage and network resources, usually within one infrastructure operator's Domain.

Virtualized Network Function (VNF): implementation of a Network Function that can be deployed on a Network Function Virtualization Infrastructure (NFVI).

Virtualized Network Function Manager (VNFM): functional block that is responsible for the lifecycle management of VNF.

3. Background

This section briefly describes some basic background technologies, as well as other standards developing organizations and open source initiatives working on network virtualization or related topics.

3.1. Network Function Virtualization

The ETSI ISG NFV is a working group which, since 2012, aims to evolve quasi-standard IT virtualization technology to consolidate many network equipment types into industry standard high volume servers, switches, and storage. It enables implementing network functions in software that can run on a range of industry standard server hardware and can be moved to, or loaded in, various locations in the network as required, without the need to install new equipment. The ETSI NFV is one of the predominant NFV reference framework and architectural footprints [nfv_sota_research_challenges]. The ETSI NFV framework architecture framework is composed of three domains (Figure 1):

- Virtualized Network Function, running over the NFVI.
- NFV Infrastructure (NFVI), including the diversity of physical resources and how these can be virtualized. NFVI supports the execution of the VNFs.
- NFV Management and Orchestration, which covers the orchestration and life-cycle management of physical and/or software resources

that support the infrastructure virtualization, and the life-cycle management of VNFs. NFV Management and Orchestration focuses on all virtualization specific management tasks necessary in the NFV framework.

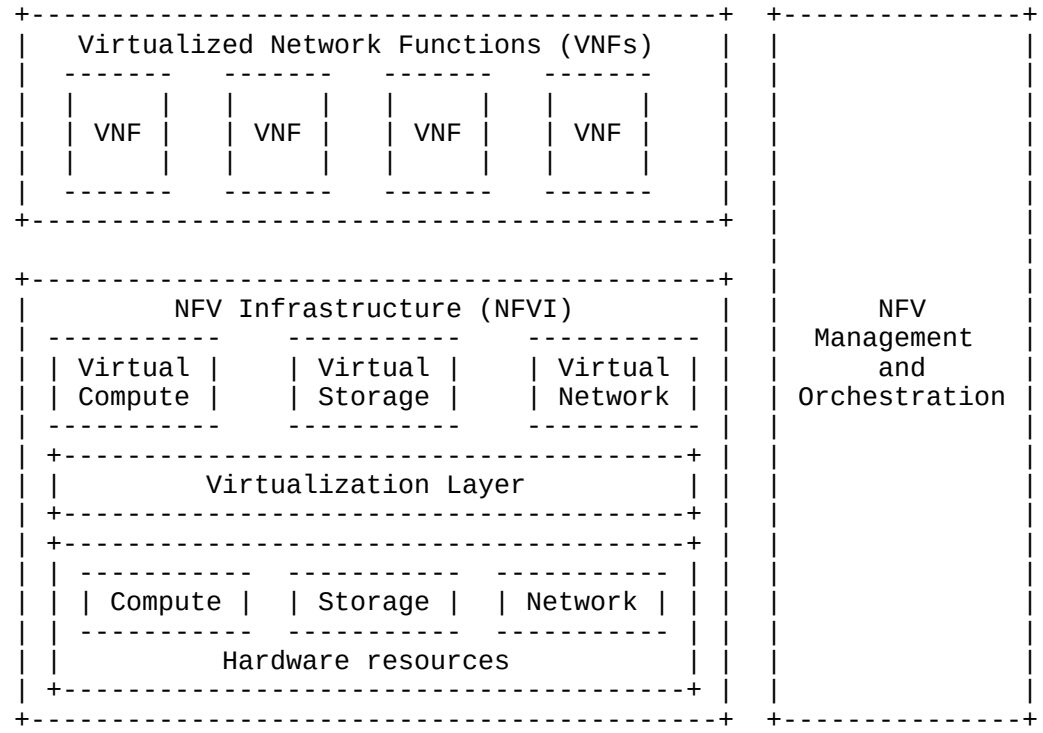


Figure 1: ETSI NFV framework

The NFV architectural framework identifies functional blocks and the main reference points between such blocks. Some of these are already present in current deployments, whilst others might be necessary additions in order to support the virtualization process and consequent operation. The functional blocks are (Figure 2):

- Virtualized Network Function (VNF).
- Element Management (EM).
- NFV Infrastructure, including: Hardware and virtualized resources, and Virtualization Layer.
- Virtualized Infrastructure Manager(s) (VIM).

- o NFV Orchestrator.
- o VNF Manager(s).
- o Service, VNF and Infrastructure Description.
- o Operations and Business Support Systems (OSS/BSS).

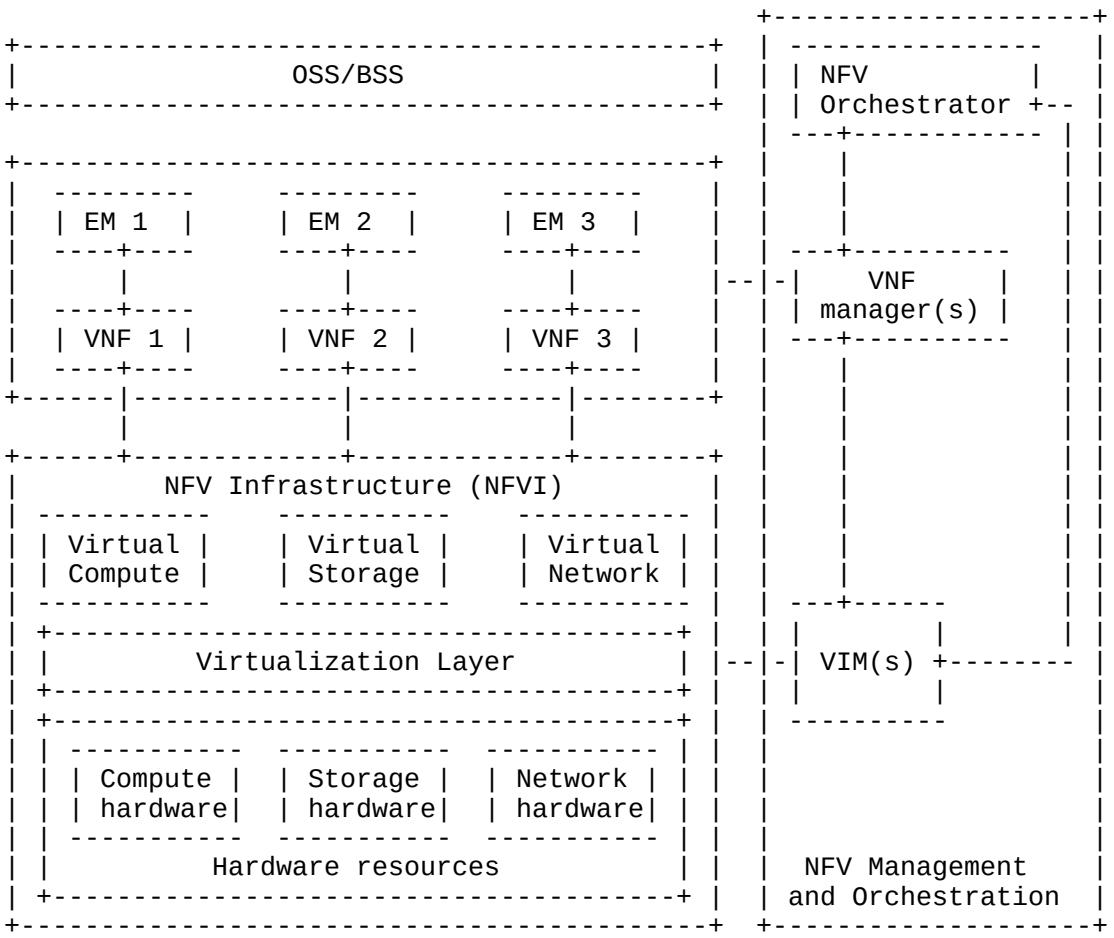


Figure 2: ETSI NFV reference architecture

3.2. Software Defined Networking

The Software Defined Networking (SDN) paradigm pushes the intelligence currently residing in the network elements to a central controller implementing the network functionality through software.

In contrast to traditional approaches, in which the network's control plane is distributed throughout all network devices, with SDN the control plane is logically centralized. In this way, the deployment of new characteristics in the network no longer requires complex and costly changes in equipment or firmware updates, but only a change in the software running in the controller. The main advantage of this approach is the flexibility it provides operators to manage their network, i.e., an operator can easily change its policies on how traffic is distributed throughout the network.

One of the most well known protocols for the SDN control plane between the central controller and the networking elements is the OpenFlow protocol (OFP), which is maintained and extended by the Open Network Foundation (ONF: https://www.opennetworking.org/). Originally this protocol was developed specifically for IEEE 802.1 switches conforming to the ONF OpenFlow Switch specification. As the benefits of the SDN paradigm have reached a wider audience, its application has been extended to more complex scenarios such as Wireless and Mobile networks. Within this area of work, the ONF is actively developing new OFP extensions addressing three key scenarios: (i) Wireless backhaul, (ii) Cellular Evolved Packet Core (EPC), and (iii) Unified access and management across enterprise wireless and fixed networks.

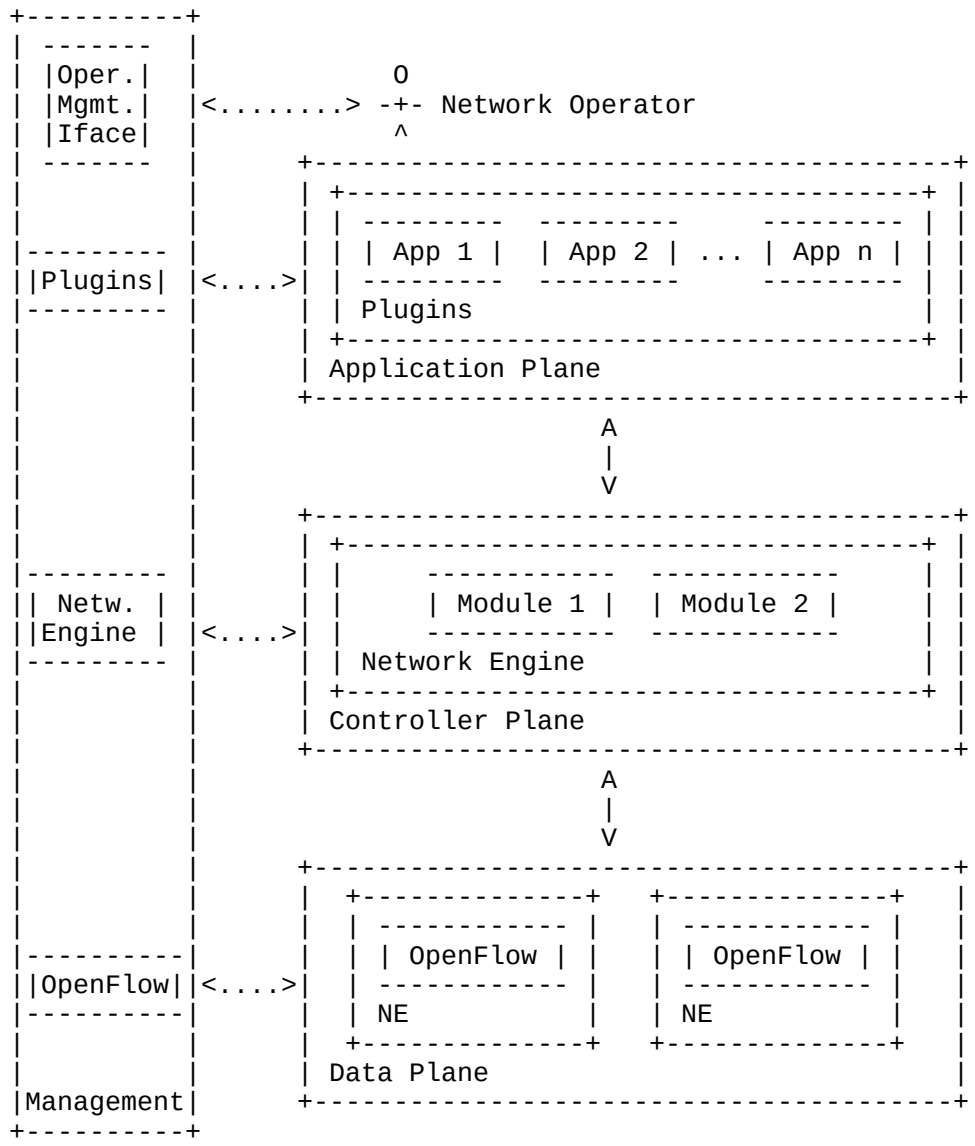


Figure 3: High level SDN ONF architecture

Figure 3 shows the blocks and the functional interfaces of the ONF architecture, which comprises three planes: Data, Controller, and Application. The Data plane comprehends several Network Entities (NE), which expose their capabilities toward the Controller plane via a Southbound API. The Controller plane includes several cooperating modules devoted to the creation and maintenance of an abstracted

resource model of the underlying network. Such model is exposed to the applications via a Northbound API where the Application plane comprises several applications/services, each of which has exclusive control of a set of exposed resources.

The Management plane spans its functionality across all planes performing the initial configuration of the network elements in the Data plane, the assignment of the SDN controller and the resources under its responsibility. In the Controller plane, the Management needs to configure the policies defining the scope of the control given to the SDN applications, to monitor the performance of the system, and to configure the parameters required by the SDN controller modules. In the Application plane, Management configures the parameters of the applications and the service level agreements. In addition to these interactions, the Management plane exposes several functions to network operators which can easily and quickly configure and tune the network at each layer.

In RFC7426 [RFC7426], the IRTF Software-Defined Networking Research Group (SDNRG) documented a layer model of an SDN architecture, since this has been a controversial discussion topic: what exactly is SDN? what is the layer structure of the SDN architecture? how do layers interface with each other? etc.

Figure 4 reproduces the figure included in RFC7426 [RFC7426] to summarize the SDN architecture abstractions in the form of a detailed, high-level schematic. In a particular implementation, planes can be collocated with other planes or can be physically separated.

In SDN, a controller manipulates controlled entities via an interface. Interfaces, when local, are mostly API invocations through some library or system call. However, such interfaces may be extended via some protocol definition, which may use local inter-process communication (IPC) or a protocol that could also act remotely; the protocol may be defined as an open standard or in a proprietary manner.

SDN expands multiple planes: Forwarding, Operational, Control, Management and Applications. All planes mentioned above are connected via interfaces. Additionally, RFC7426 [RFC7426] considers four abstraction layers: the Device and resource Abstraction Layer (DAL), the Control Abstraction Layer (CAL), the Management Abstraction Layer (MAL) and the Network Services Abstraction Layer (NSAL).

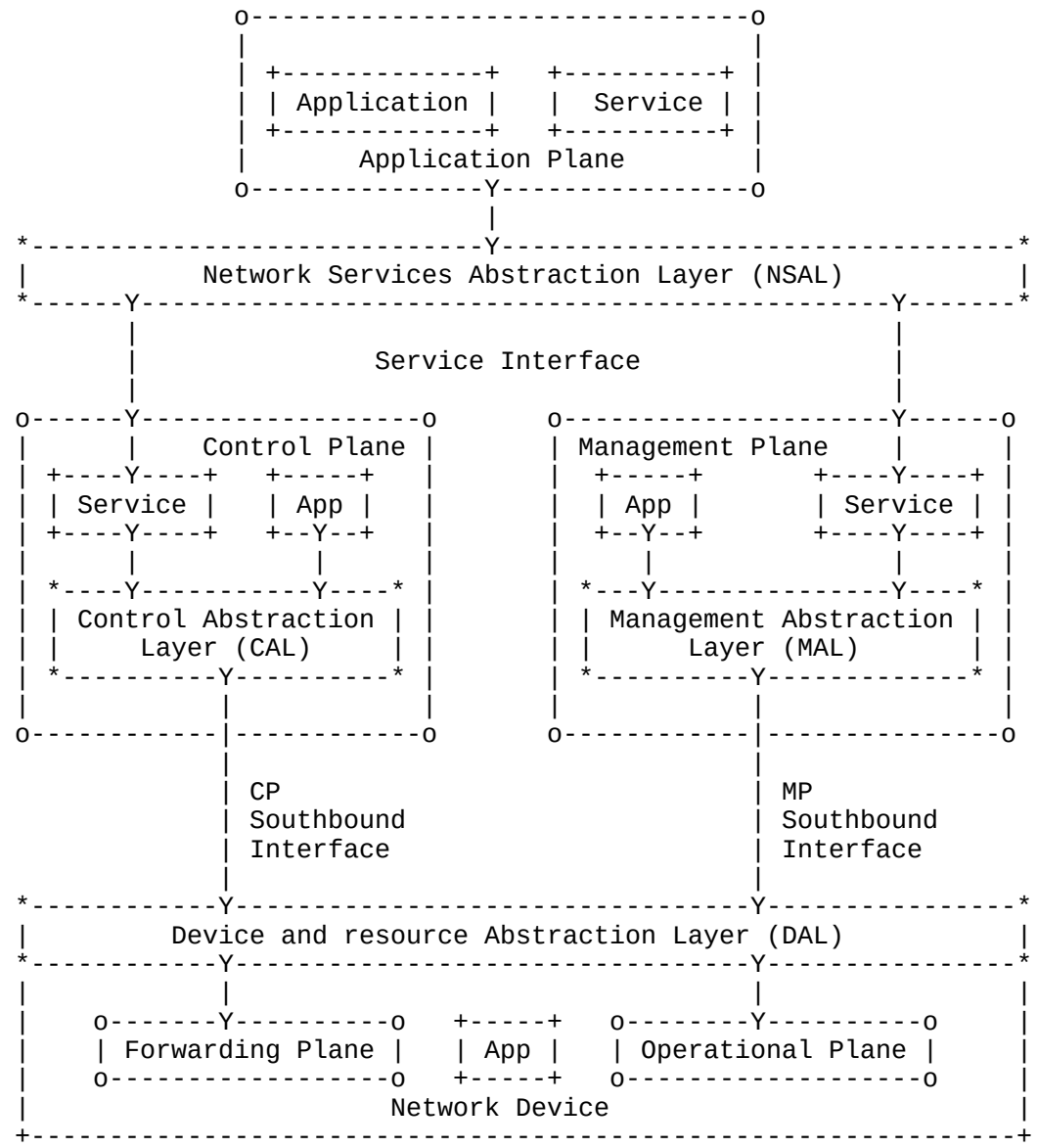


Figure 4: SDN Layer Architecture

While SDN is often directly associated to OpenFlow, this is just one (relevant) example of a southbound protocol between the central controller and the network entities. Other relevant examples of protocols in the SDN family are NETCONF [RFC6241], RESTCONF [RFC8040] and ForCES [RFC5810].

3.3. ITU-T functional architecture of SDN

The Telecommunication standardization sector of the International Telecommunication Union (ITU) -- the ITU-T -- has also looked into SDN architectures, defining a slightly modified one from what other SDOs have done. ITU-T provides in the recommendation ITU-T Y.3302 [itu-t-y.3302] a functional architecture of SDN with descriptions of functional components and reference points. The described functional architecture is intended to be used as an enabler for further studies on other aspects such as protocols and security as well as being used to customize SDN in support of appropriate use cases (e.g., cloud computing, mobile networks). This recommendation is based on ITU-T Y.3300 [itu-t-y.3300] and ITU-T Y.3301 [itu-t-y.3301]. While the first describes the framework of SDN (including definitions, objectives, high-level capabilities, requirements and the high-level architecture of SDN), the second describes more detailed requirements.

Figure 5 shows the SDN functional architecture defined by the ITU-T. It is a layered architecture composed of the SDN application layer (SDN-AL), the SDN control layer (SDN-CL) and the SDN resource layer (SDN-RL). It also has multi-layer management functions (MMF), which provides functionalities for managing the functionalities of SDN layers, i.e., SDN-AL, SDN-CL and SDN-RL. MMF interacts with these layers using MMFA, MMFC, and MMFR reference points.

The SDN-AL enables a service-aware behavior of the underlying network in a programmatic manner. The SDN-CL provides programmable means to control the behavior of SDN-RL resources (such as data transport and processing), following requests received from the SDN-AL according to MMF policies. The SDN-RL is where the physical or virtual network elements perform transport and/or processing of data packets according to SDN-CL decisions.

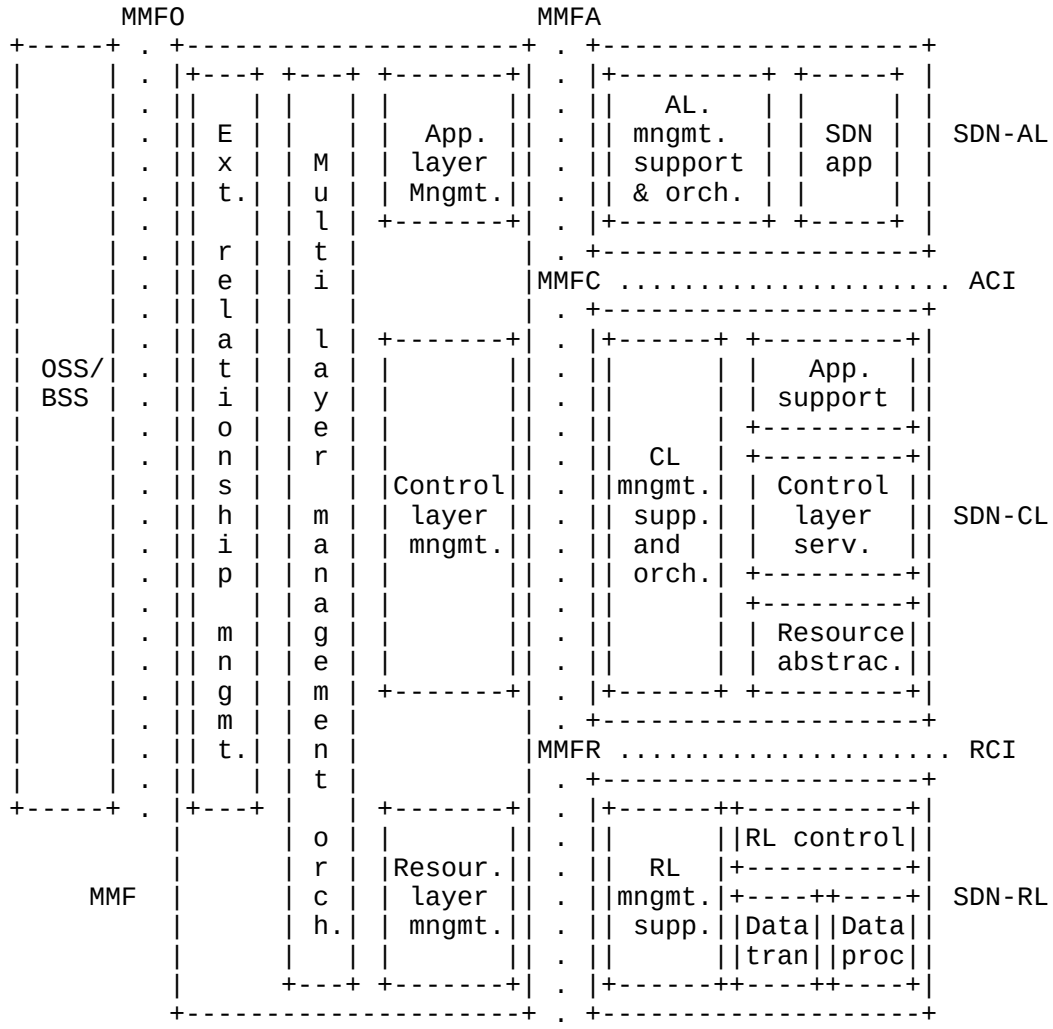


Figure 5: ITU-T SDN functional architecture

3.4. Multi-access Edge Computing

Multi-access Edge Computing (MEC) -- formerly known as Mobile Edge Computing -- capabilities deployed in the edge of the mobile network can facilitate the efficient and dynamic provision of services to mobile users. The ETSI ISG MEC working group, operative from end of 2014, intends to specify an open environment for integrating MEC capabilities with service providers' networks, including also applications from 3rd parties. These distributed computing capabilities will make available IT infrastructure as in a cloud

environment for the deployment of functions in mobile access networks. It can be seen then as a complement to both NFV and SDN.

3.5. IEEE 802.1CF (OmniRAN)

The IEEE 802.1CF Recommended Practice [omniran] specifies an access network, which connects terminals to their access routers, utilizing technologies based on the family of IEEE 802 Standards (e.g., 802.3 Ethernet, 802.11 Wi-Fi, etc.). The specification defines an access network reference model, including entities and reference points along with behavioral and functional descriptions of communications among those entities.

The goal of this project is to help unifying the support of different interfaces, enabling shared network control and use of SDN principles, thereby lowering the barriers to new network technologies, to new network operators, and to new service providers.

3.6. Distributed Management Task Force

The DMTF (https://www.dmtf.org/) is an industry standards organization working to simplify the manageability of network-accessible technologies through open and collaborative efforts by some technology companies. The DMTF is involved in the creation and adoption of interoperable management standards, supporting implementations that enable the management of diverse traditional and emerging technologies including cloud, virtualization, network and infrastructure.

There are several DMTF initiatives that are relevant to the network virtualization area, such as the Open Virtualization Format (OVF), for VNF packaging; the Cloud Infrastructure Management Interface (CIM), for cloud infrastructure management; the Network Management (NETMAN), for VNF management; and, the Virtualization Management (VMAN), for virtualization infrastructure management.

3.7. Open Source initiatives

The Open Source community is especially active in the area of network virtualization and orchestration. We next summarize some of the active efforts:

- OpenStack. OpenStack is a free and open-source cloud-computing software platform. OpenStack software controls large pools of compute, storage, and networking resources throughout a datacenter, managed through a dashboard or via the OpenStack API.

- Kubernetes. Kubernetes is an open-source system for automating deployment, scaling and management of containerized applications. Kubernetes can schedule and run application containers on clusters of physical or virtual machines. Kubernetes allows: (i) Scale on the fly, (ii) Limit hardware usage to required resources only, (iii) Load balancing Monitoring, and (iv) Efficient lifecycle management.

- OpenDayLight. OpenDaylight (ODL) is a highly available, modular, extensible and scalable 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.

- ONOS. The ONOS (Open Network Operating System) project is an open source community hosted by The Linux Foundation. The goal of the project is to create a SDN operating system for communications service providers that is designed for scalability, high performance and high availability.

- OpenContrail. OpenContrail is an Apache 2.0-licensed project that is built using standards-based protocols and provides all the necessary components for network virtualization-SDN controller, virtual router, analytics engine, and published northbound APIs. It has an extensive REST API to configure and gather operational and analytics data from the system.

- OPNFV. OPNFV is a carrier-grade, integrated, open source platform to accelerate the introduction of new NFV products and services. By integrating components from upstream projects, the OPNFV community aims at conducting performance and use case-based testing to ensure the platform’s suitability for NFV use cases. The scope of OPNFV’s initial release is focused on building NFV Infrastructure (NFVI) and Virtualized Infrastructure Management (VIM) by integrating components from upstream projects such as OpenDaylight, OpenStack, Ceph Storage, KVM, Open vSwitch, and Linux. These components, along with application programmable interfaces (APIs) to other NFV elements form the basic infrastructure required for Virtualized Network Functions (VNF) and Management and Network Orchestration (MANO) components. OPNFV’s goal is to (i) increase performance and power efficiency, (ii) improve reliability, availability, and serviceability, and (iii) deliver comprehensive platform instrumentation.

- OSM. Open Source Mano (OSM) is an ETSI-hosted project to develop an Open Source NFV Management and Orchestration (MANO) software stack aligned with ETSI NFV. OSM is based on components from

previous projects, such Telefonica's OpenMANO or Canonical's Juju, among others.

- OpenBaton. OpenBaton is a ETSI NFV compliant Network Function Virtualization Orchestrator (NFVO). OpenBaton was part of the OpenSDNCore project started with the objective of providing a compliant implementation of the ETSI NFV specification.

- ONAP. ONAP (Open Network Automation Platform) is an open source software platform that delivers capabilities for the design, creation, orchestration, monitoring, and life cycle management of: (i) Virtual Network Functions (VNFs), (ii) The carrier-scale Software Defined Networks (SDNs) that contain them, and (iii) Higher-level services that combine the above. ONAP (derived from the AT&T's ECOMP) provides for automatic, policy-driven interaction of these functions and services in a dynamic, real-time cloud environment.

- SONA. SONA (Simplified Overlay Network Architecture) is an extension to ONOS to have a almost full SDN network control in OpenStack for virtual tenant network provisioning. Basically, SONA is an SDN-based network virtualization solution for cloud DC.

Among the main areas that are being developed by the former open source activities that relate to network virtualization research, we can highlight: policy-based resource management, analytics for visibility and orchestration, service verification with regards to security and resiliency.

4. Network Virtualization Challenges

4.1. Introduction

Network Virtualization is changing the way the telecommunications sector will deploy, extend and operate their networks. These new technologies aim at reducing the overall costs by moving communication services from specific hardware in the operators' core to server farms scattered in datacenters (i.e. compute and storage virtualization). In addition, the networks interconnecting the functions that compose a network service are fundamentally affected in the way they route, process and control traffic (i.e. network virtualization).

4.2. Guaranteeing quality-of-service

Achieving a given quality-of-service in an NFV environment with virtualized and distributed computing, storage and networking functions is more challenging than providing the equivalent in

discrete non-virtualized components. For example, ensuring a guaranteed and stable forwarding data rate has proven not to be straightforward when the forwarding function is virtualized and runs on top of COTS server hardware [openmano_dataplane] [I-D.mlk-nfvrg-nfv-reliability-using-cots] [etsi_nvf_whitepaper_3]. Again, the comparison point is against a router or forwarder built on optimized hardware. We next identify some of the challenges that this poses.

4.2.1. Virtualization Technologies

The issue of guaranteeing a network quality-of-service is less of an issue for "traditional cloud computing" because the workloads that are treated there are servers or clients in the networking sense and hardly ever process packets. Cloud computing provides hosting for applications on shared servers in a highly separated way. Its main advantage is that the infrastructure costs are shared among tenants and that the cloud infrastructure provides levels of reliability that can not be achieved on individual premises in a cost-efficient way [intel_10_differences_nfv_cloud]. NFV has very strict requirements posed in terms of performance, stability and consistency. Although there are some tools and mechanisms to improve this, such as Enhanced Performance Awareness (EPA), Single Root I/O Virtualization (SR-IOV), Non-Uniform Memory Access (NUMA), Data Plane Development Kit (DPDK), etc, these are still unsolved challenges. One open research issue is finding out technologies that are different from VM and more suitable for dealing with network functionalities.

Lately, a number of light-weight virtualization technologies including containers, unikernels (specialized VMs) and minimalistic distributions of general-purpose OSes have appeared as virtualization approaches that can be used when constructing an NFV platform. [I-D.natarajan-nfvrg-containers-for-nfv] describes the challenges in building such a platform and discusses to what extent these technologies, as well as traditional VMs, are able to address them.

4.2.2. Metrics for NFV characterization

Another relevant aspect is the need for tools for diagnostics and measurement suited for NFV. There is a pressing need to define metrics and associated protocols to measure the performance of NFV. Specifically, since NFV is based on the concept of taking centralized functions and evolving it to highly distributed SW functions, there is a commensurate need to fully understand and measure the baseline performance of such systems.

The IP Performance Metrics (IPPM) WG defines metrics that can be used to measure the quality and performance of Internet services and

applications running over transport layer protocols (e.g., TCP, UDP) over IP. It also develops and maintains protocols for the measurement of these metrics. While the IPPM WG is a long running WG that started in 1997, at the time of writing it does not have a charter item or active drafts related to the topic of network virtualization. In addition to using IPPM metrics to evaluate the QoS, there is a need for specific metrics for assessing the performance of network virtualization techniques.

The Benchmarking Methodology Working Group (BMWG) is also performing work related to NFV metrics. For example, [RFC8172] investigates additional methodological considerations necessary when benchmarking VNFs instantiated and hosted in general-purpose hardware, using bare-metal hypervisors or other isolation environments such as Linux containers. An essential consideration is benchmarking physical and virtual network functions in the same way when possible, thereby allowing direct comparison.

As stated in the document [RFC8172], there is a clear motivation for the work on performance metrics for NFV [etsi_gs_nfv_per_001], that is worth replicating here: "I'm designing and building my NFV Infrastructure platform. The first steps were easy because I had a small number of categories of VNFs to support and the VNF vendor gave HW recommendations that I followed. Now I need to deploy more VNFs from new vendors, and there are different hardware recommendations. How well will the new VNFs perform on my existing hardware? Which among several new VNFs in a given category are most efficient in terms of capacity they deliver? And, when I operate multiple categories of VNFs (and PNFs) *concurrently* on a hardware platform such that they share resources, what are the new performance limits, and what are the software design choices I can make to optimize my chosen hardware platform? Conversely, what hardware platform upgrades should I pursue to increase the capacity of these concurrently operating VNFs?"

Lately, there are also some efforts looking into VNF benchmarking. The selection of an NFV Infrastructure Point of Presence to host a VNF or allocation of resources (e.g., virtual CPUs, memory) needs to be done over virtualized (abstracted and simplified) resource views [vnf_benchmarking] [I-D.rorosz-nfvrg-vbaas].

4.2.3. Predictive analysis

On top of diagnostic tools that enable an assessment of the QoS, predictive analyses are required to react before anomalies occur. Due to the SW characteristics of VNFs, a reliable diagnosis framework could potentially enable the prevention of issues by a proper diagnosis and then a reaction in terms of acting on the potentially

impacted service (e.g., migration to a different compute node, scaling in/out, up/down, etc).

4.2.4. Portability

Portability in NFV refers to the ability to run a given VNF on multiple NFVIs, that is, guaranteeing that the VNF would be able to perform its functions with a high and predictable performance given that a set of requirements on the NFVI resources is met. Therefore, portability is a key feature that, if fully enabled, would contribute to making the NFV environment achieve a better reliability than a traditional system. Implementing functionality in SW over "commodity" infrastructure should make it much easier to port/move functions from one place to another. However this is not yet as ideal as it sounds, and there are aspects that are not fully tackled. The existence of different hypervisors, specific hardware dependencies (e.g., EPA related) or state synchronization aspects are just some examples of trouble-makers for portability purposes.

The ETSI NFV ISG is doing work in relation to portability. [etsi_gs_nfv_per_001] provides a list of minimal features which the VM Descriptor and Compute Host Descriptor should contain for the appropriate deployment of VM images over an NFVI (i.e. a "telco datacenter"), in order to guarantee high and predictable performance of data plane workloads while assuring their portability. In addition, the document provides a set of recommendations on the minimum requirements which HW and hypervisor should have for a "telco datacenter" suitable for different workloads (data-plane, control-plane, etc.) present in VNFs. The purpose of this document is to provide the list of VM requirements that should be included in the VM Descriptor template, and the list of HW capabilities that should be included in the Compute Host Descriptor (CHD) to assure predictable high performance. ETSI NFV assumes that the MANO Functions will make the mix & match. There are therefore still several research challenges to be addressed here.

4.3. Performance improvement

4.3.1. Energy Efficiency

Virtualization is typically seen as a direct enabler of energy savings. Some of the enablers for this that are often mentioned [nfv_sota_research_challenges] are: (i) the multiplexing gains achieved by centralizing functions in data centers reduce the overall energy consumed, (ii) the flexibility brought by network programmability enables to switch off infrastructure as needed in a much easier way. However there is still a lot of room for

improvement in terms of virtualization techniques to reduce the power consumption, such as enhanced hypervisor technologies.

Some additional examples of research topics that could enable energy savings are [nfv_sota_research_challenges]:

- Energy aware scaling (e.g., reductions in CPU speeds and partially turning off some hardware com- ponents to meet a given energy consumption target.
- Energy-aware function placement.
- Scheduling and chaining algorithms, for example adapting the network topology and operating parameters to minimize the operation cost (e.g., tracking energy costs to identify the cheapest prices).

Note that it is also important to analyze the trade-off between energy efficiency and network performance.

4.3.2. Improved link usage

The use of NFV and SDN technologies can help improve link usage. SDN has already shown that it can greatly increase average link utilization (e.g., Google example [google_sdn_wan]). NFV adds more complexity (e.g., due to service function chaining / VNF forwarding graphs) which need to be considered. Aspects like the ones described in [I-D.bagnulo-nfvrg-topology] on NFV data center topology design have to be carefully looked at as well.

4.4. Multiple Domains

Market fragmentation has resulted in a multitude of network operators each focused on different countries and regions. This makes it difficult to create infrastructure services spanning multiple countries, such as virtual connectivity or compute resources, as no single operator has a footprint everywhere. Cross-domain orchestration of services over multiple administrations or over multi-domain single administrations will allow end-to-end network and service elements to mix in multi-vendor, heterogeneous technology and resource environments [multi-domain_5GEx].

For the specific use case of ’Network as a Service’, it becomes even more important to ensure that Cross Domain Orchestration also takes care of hierarchy of networks and their association, with respect to provisioning tunnels and overlays.

Multi-domain orchestration is currently an active research topic, which is being tackled, among others, by ETSI NFV ISG and the 5GEx project (https://www.5gex.eu/) [I-D.bernardos-nfvrg-multidomain] [multi-domain_5GEx].

Another side of the multi-domain problem is the integration/ harmonization of different management domains. A key example comes from Multi-access Edge Computing, which, according to ETSI, comes with its own MANO system, and would require to be integrated if interconnected to a generic NFV system.

4.5. 5G and Network Slicing

From the beginning of all 5G discussions in the research and industry fora, it has been agreed that 5G will have to address much more use cases than the preceding wireless generations, which first focused on voice services, and then on voice and high speed packet data services. In this case, 5G should be able to handle not only the same (or enhanced) voice and packet data services, but also new emerging services like tactile Internet and IoT. These use cases take the requirements to opposite extremes, as some of them require ultra-low latency and higher-speed, whereas some others require ultra-low power consumption and high delay tolerance.

Because of these very extreme 5G use cases, it is envisioned that selective combinations of radio access networks and core network components will have to be combined into a given network slice to address the specific requirements of each use case.

For example, within the major IoT category, which is perhaps the most disrupting one, some autonomous IoT devices will have very low throughput, will have much longer sleep cycles (and therefore high latency), and a battery life time exceeding by a factor of thousands that of smart phones or some other devices that will have almost continuous control and data communications. Hence, it is envisioned that a customized network slice will have to be stitched together from virtual resources or sub-slices to meet these requirements.

The actual definition of network slice from an IP infrastructure viewpoint is currently undergoing intense debate [I-D.geng-coms-problem-statement] [I-D.gdmb-netslices-intro-and-ps] [I-D.defoy-netslices-3gpp-network-slicing] [ngmn_5G_whitepaper]. Network slicing is a key for introducing new actors in existing market at low cost -- by letting new players rent "blocks" of capacity, if the new business model enables performance that meets the application needs (e.g., broadcasting updates to many sensors with satellite broadcasting capabilities). However, more work needs to be done to define the basic architectural approach of how network

slices will be defined and formed. For example, is it mostly a matter of defining the appropriate network models (e.g. YANG) to stitch the network slice from existing components. Or do end-to-end timing, synchronization and other low level requirements mean that more fundamental research has to be done.

4.5.1. Virtual Network Operators

The widespread use/discussion/practice of system and network virtualization technologies has led to new business opportunities, enlarging the offer of IT resources with virtual network and computing resources, among others. As a consequence, the network ecosystem now differentiates between the owner of physical resources, the Infrastructure Provider (InP), and the intermediary that conforms and delivers network services to the final customers, the Virtual Network Operator (VNO).

VNOs aim to exploit the virtualized infrastructures to deliver new and improved services to their customers. However, current network virtualization techniques offer poor support for VNOs to control their resources. It has been considered that the InP is responsible for the reliability of the virtual resources but there are several situations in which a VNO requires to gain a finer control on its resources. For instance, dynamic events, such as the identification of new requirements or the detection of incidents within the virtual system, might urge a VNO to quickly reform its virtual infrastructure and resource allocation. However, the interfaces offered by current virtualization platforms do not offer the necessary functions for VNOs to perform the elastic adaptations they require to tackle with their dynamic operation environments.

Beyond their heterogeneity, which can be resolved by software adapters, current virtualization platforms do not have common methods and functions, so it is difficult for the virtual network controllers used by the VNOs to actually manage and control virtual resources instantiated on different platforms, not even considering different InPs. Therefore it is necessary to reach a common definition of the functions that should be offered by underlying platforms to give such overlay controllers the possibility to allocate and deallocate resources dynamically and get monitoring data about them.

Such common methods should be offered by all underlying controllers, regardless of being network-oriented (e.g. ODL, ONOS, Ryu) or computing-oriented (e.g. OpenStack, OpenNebula, Eucalyptus). Furthermore, it is also important for those platforms to offer some "PUSH" function to report resource state, avoiding the need for the VNO’s controller to "POLL" for such data. A starting point to get

proper notifications within current REST APIs could be to consider the protocol proposed by the WEBPUSH WG [RFC8030].

Finally, in order to establish a proper order and allow the coexistence and collaboration of different systems, a common ontology regarding network and system virtualization should be defined and agreed, so different and heterogeneous systems can understand each other without requiring to rely on specific adaptation mechanisms that might break with any update on any side of the relation.

4.5.2. Extending Virtual Networks and Systems to the Internet of Things

The Internet of Things (IoT) refers to the vision of connecting a multitude of automated devices (e.g. lights, environmental sensors, traffic lights, parking meters, health and security systems, etc.) to the Internet for purposes of reporting, and remote command and control of the device. This vision is being realized by a multi-pronged approach of standardization in various forums and complementary open source activities. For example, in the IETF, support of IoT web services has been defined by an HTTP-like protocol adapted for IoT called CoAP [RFC7252], and lately a group has been studying the need to develop a new network layer to support IP applications over Low Power Wide Area Networks (LPWAN).

Elsewhere, for 5G cellular evolution there is much discussion on the need for supporting virtual "network slices" for the expected massive numbers of IoT devices. A separate virtual network slice is considered necessary for different 5G IoT use cases because devices will have very different characteristics than typical cellular devices like smart phones [ngmn_5G_whitepaper], and the number of IoT devices is expected to be at least one or two orders of magnitude higher than other 5G devices (see Section 4.5).

The specific nature of the IoT ecosystem, particularly reflected in the Machine-to-Machine (M2M) communications, leads to the creation of new and highly distributed systems which demand location-based network and computing services. A specific example can be represented by a set of "things" that suddenly require to set-up a firewall to allow external entities to access their data while outsourcing some computation requirements to more powerful systems relying on cloud-based services. This representative use case exposes important requirements for both NFV and the underlying cloud infrastructures.

In order to provide the aforementioned location-based functions integrated with highly distributed systems, the so called fog infrastructures should be able to instantiate VNFs, placing them in the required place, e.g. close to their consumers. This requirement

implies that the interfaces offered by virtualization platforms must support the specification of location-based resources, which is a key function in those scenarios. Moreover, those platforms must also be able to interpret and understand the references used by IoT systems to their location (e.g., "My-AP", "5BLDG+2F") and also the specification of identifiers linked to other resources, such as the case of requiring the infrastructure to establish a link between a specific AP and a specific virtual computing node. In summary, the research gap is exact localization of VNFs at far network edge infrastructure which is highly distributed and dynamic.

4.6. Service Composition

Current network services deployed by operators often involve the composition of several individual functions (such as packet filtering, deep packet inspection, load balancing). These services are typically implemented by the ordered combination of a number of service functions that are deployed at different points within a network, not necessarily on the direct data path. This requires traffic to be steered through the required service functions, wherever they are deployed [RFC7498].

For a given service, the abstracted view of the required service functions and the order in which they are to be applied is called a Service Function Chain (SFC) [sfc_challenges], which is called Network Function Forwarding Graph (NF-FG) in ETSI. An SFC is instantiated through selection of specific service function instances on specific network nodes to form a service graph: this is called a Service Function Path (SFP). The service functions may be applied at any layer within the network protocol stack (network layer, transport layer, application layer, etc.).

Service composition is a powerful means which can provide significant benefits when applied in a softwarized network environment. There are however many research challenges in this area, as for example the ones related to composition mechanisms and algorithms to enable load balancing and improve reliability. The service composition should also act as an enabler to gather information across all hierarchies (underlays and overlays) of network deployments which may span across multiple operators, for faster serviceability thus facilitating accomplishing aforementioned goals of "load balancing and improve reliability".

As described in [dynamic_chaining], different algorithms can be used to enable dynamic service composition that optimizes a QoS-based utility function (e.g., minimizing the latency per-application traffic flows) for a given composition plan. Such algorithms can consider the computation capabilities and load status of resources

executing the VNF instances, either deduced through estimations from historical usage data or collected through real-time monitoring (i.e., context-aware selection). For this reason, selections should include references to dynamic information on the status of the service instance and its constituent elements, i.e., monitoring information related to individual VNF instances and links connecting them as well as derived monitoring information at the chain level (e.g., end-to-end delay). At runtime, if one or more VNF instances are no more available or QoS degrades below a given threshold, the service selection task can be rerun to perform service substitution.

There are different research directions that relate to the previous point. For example, the use of Integer Linear Programming (ILP) techniques can be explored to optimize the management of diverse traffic flows. Deep machine learning can also be applied to optimize service chains using information parameters such as some of the ones mentioned above. Newer scheduling paradigms, like co-flows, can also be used.

The SFC working group is working on an architecture for service function chaining [RFC7665] that includes the necessary protocols or protocol extensions to convey the Service Function Chain and Service Function Path information to nodes that are involved in the implementation of service functions and Service Function Chains, as well as mechanisms for steering traffic through service functions.

In terms of actual work items, the SFC WG is has not yet considered working on the management and configuration of SFC components related to the support of Service Function Chaining. This part is of special interest for operators and would be required in order to actually put SFC mechanisms into operation. Similarly, redundancy and reliability mechanisms for service function chaining are currently not dealt with by any WG in the IETF. While this was the main goal of the VNFpool BoF efforts, it still remains unaddressed.

4.7. End-user device virtualization

So far, most of the network softwarization efforts have focused on virtualizing functions of network elements. While virtualization of network elements started with the core, mobile networks architectures are now heavily switching to also virtualize radio access network (RAN) functions. The next natural step is to get virtualization down at the level of the end-user device (e.g., virtualizing a smartphone) [virtualization_mobile_device]. The cloning of a device in the cloud (central or local) bears attractive benefits to both the device and network operations alike (e.g., power saving at the device by offloading computational-heaving functions to the cloud, optimized networking -- both device-to-device and device-to-infrastructure) for

service delivery through tighter integration of the device (via its clone in the networking infrastructure). This is, for example, being explored by the European H2020 ICIRRUS project (www.icirrus-5gnet.eu).

4.8. Security and Privacy

Similar to any other situation where resources are shared, security and privacy are two important aspects that need to be taken into account.

In the case of security, there are situations where multiple service providers will need to coexist in a virtual or hybrid physical/ virtual environment. This requires attestation procedures amongst different virtual/physical functions and resources, as well as ongoing external monitoring. Similarly, different network slices operating on the same infrastructure can present security problems, for instance if one slice running critical applications (e.g. support for a safety system) is affected by another slice running a less critical application. In general, the minimum common denominator for security measures on a shared system should be equal or higher than the one required by the most critical application. Multiple and continuous threat model analysis, as well as DevOps model are required to maintain a certain level of security in an NFV system. Simplistically, DevOps is a process that combines multiple functions into single cohesive teams in order to quickly produce quality software. It typically relies on also applying the Agile development process, which focuses on (among many things) dividing large features into multiple, smaller deliveries. One part of this is to immediately test the new smaller features in order to get immediate feedback on errors so that if present, they can be immediately fixed and redeployed.

On the other hand, privacy refers to concerns about the control of personal data and the decision of what to reveal to whom. In this case, the storage, transmission, collection, and potential correlation of information in the NFV system, for purposes not originally intended or not known by the user, should be avoided. This is particularly challenging, as future intentions and threats cannot be easily predicted, and still can be applied on data collected in the past. Therefore, well-known techniques such as data minimization, using privacy features as default, and allowing users to opt in/out should be used to prevent potential privacy issues.

Compared to traditional networks, NFV will result in networks that are much more dynamic (in function distribution and topology) and elastic (in size and boundaries). NFV will thus require network operators to evolve their operational and administrative security

solutions to work in this new environment. For example, in NFV the network orchestrator will become a key node to provide security policy orchestration across the different physical and virtual components of the virtualized network. For highly confidential data, for example, the network orchestrator should take into account if certain physical hardware (HW) of the network is considered more secure (e.g., because it is located in secure premises) than other HW.

Traditional telecom networks typically run under a single administrative domain controlled by (exactly) one operator. With NFV, it is expected that in many cases, the telecom operator will now become a tenant (running the VNFs), and the infrastructure (NFVI) may be run by a different operator and/or cloud service provider (see also Section 4.4). Thus, there will be multiple administrative domains involved, making security policy coordination more complex. For example, who will be in charge of provisioning and maintaining security credentials such as public and private keys? Also, should private keys be allowed to be replicated across the NFV for redundancy reasons? Alternatively, it can be investigated how to develop a mechanism that avoid such a security policy coordination, this making the system more robust.

On a positive note, NFV may better defense against Denial of Service (DoS) attacks because of the distributed nature of the network (i.e. no single point of failure) and the ability to steer (undesirable) traffic quickly [etsi_gs_nfv_sec_001]. Also, NFVs which have physical HW which is distributed across multiple data centers will also provide better fault isolation environments. This holds true in particular if each data center is protected separately via firewalls, DMZs and other network protection techniques.

SDN can also be used to help improve security by facilitating the operation of existing protocols, such as Authentication, Authorization and Accounting (AAA). The management of AAA infrastructures, namely the management of AAA routing and the establishment of security associations between AAA entities, can be performed using SDN, as analyzed in [I-D.marin-sdnrg-sdn-aaa-mng].

4.9. Separation of control concerns

NFV environments offer two possible levels of SDN control. One level is the need for controlling the NFVI to provide connectivity end-to-end among VNFs or among VNFs and PNFs (Physical Network Functions). A second level is the control and configuration of the VNFs themselves (in other words, the configuration of the network service implemented by those VNFs), taking advantage of the programmability brought by SDN. Both control concerns are separated in nature.

However, interaction between both could be expected in order to optimize, scale or influence each other.

Clear mechanisms for such interaction are needed in order to avoid malfunctioning or interference concerns. These ideas are considered in [etsi_gs_nfv_eve005] and [I-D.irtf-sdnrg-layered-sdn]

4.10. Network Function placement

Network function placement is a problem in any kind of network telecommunications infrastructure. Moreover, the increased degree of freedom added by network virtualization makes this problem even more important, and also harder to tackle. Deciding where to place virtual network functions is a resource allocation problem which needs to (or may) take into consideration quite a few aspects: resiliency, (anti-)affinity, security, privacy, energy efficiency, etc.

When several functions are chained (typical scenario), placement algorithms become more complex and important (as described in Section 4.6). While there has been research on the topic [nfv_piecing] [dynamic_placement][vnf-p], this still remains an open challenges that requires more attention. Multi-domain also adds another component of complexity to this problem that has to be considered.

4.11. Testing

The impacts of network virtualization on testing can be divided into 3 groups:

1. Changes in methodology.

2. New functionality.

3. Opportunities.

4.11.1. Changes in methodology

The largest impact of NFV is the ability to isolate the System Under Test (SUT). When testing Physical Network Functions (PNF), isolating the SUT means that all the other devices that the SUT communicates with are replaced with simulations (or controlled executions) in order to place the SUT under test by itself. The SUT may be comprised of one or more devices. The simulations use the appropriate traffic type and protocols in order to execute test cases.

As shown in Figure 2, NFV provides a common architecture for all functions to use. A VNF is executed using resources offered by the NFVI, which have been allocated using the MANO function. It is not possible to test a VNF by itself, without the entire supporting environment present. This fundamentally changes how to consider the SUT. In the case of a VNF (or multiple VNFs), the SUT is part of a larger architecture which is necessary in order to run the SUTs.

Isolation of the SUT therefore becomes controlling the environment in a disciplined manner. The components of the environment necessary to run the SUTs that are not part of the SUT become the test environment. In the case of VNFs which are the SUT, the NFVI and MANO become the test environment. The configurations and policies that guide the test environment should remain constant during the execution of the tests, and also from test to test. Configurations such as CPU pinning, NUMA configuration, the SW versions and configurations of the hypervisor, vSwitch and NICs should remain constant. The only variables in the testing should be those controlling the SUT itself. If any configuration in the test environment is changed from test to test, the results become very difficult, if not impossible, to compare since the test environment behavior may change the results as a consequence of the configuration change.

Testing the NFVI itself also presents new considerations. With a PNF, the dedicated hardware supporting it is optimized for the particular workload of the function. Routing hardware is specially built to support packet forwarding functions, while the hardware to support a purely control plane application (say, a DNS server, or a Diameter function) will not have this specialized capability. In NFV, the NFVI is required to support all types of potentially different workload types.

Testing the NFVI therefore requires careful consideration about what types of metrics are sought. This, in turn, depends on the workload type the expected VNF will be. Examples of different workload types are data forwarding, control plane, encryption, and authentication. All these types of expected workloads will determine the types of metrics that should be sought. For example, if the workload is control plane, then a metric such as jitter is not useful, but dropped packets are critical. In a multi-tenant environment, the NFVI could support various types of workloads. In this case, testing with a variety of traffic types while measuring the corresponding metrics simultaneously becomes necessary.

Test beds for any type of testing for an NFV-based system will be largely similar to previously used test architectures. The methods are impacted by virtualization, as described above, but the design of

test beds are similar as in the past. There are two main new considerations:

- Since networking is based on software, which has lead to greater automation in deployment, the test system should also be deployable with the rest of the system in order to fully automate the system. This is especially relevant in a DevOps environment supported by a CI/CD tool chain (see Section 4.11.3 below).

- In any performance test bed, the test system should not share the same resources as the System Under Test (SUT). While multi-tenenacy is a reality in virtualization, having the test system share resources with the SUT will impact the measured results in a performance test bed. The test system should be deployed on a separate platform in order to not to impact the resources available to the SUT.

4.11.2. New functionality

NFV presents a collection of new functionality in order to support the goal of software networking. Each component on the architecture shown in Figure 2 has an associated set of functionality that allows VNFs to run: onboarding, lifecycle management for VNFs and Networks Services (NS), resource allocation, hypervisor functions, etc.

One of the new capabilities enabled by NFV is VNFFG (VNF Forwarding Graphs). This refers to the graph that represents a Network Service by chaining together VNFs into a forwarding path. In practice, the forwarding path can be implemented in a variety of ways using different networking capabilities: vSwitch, SDN, SDN with a northbound application, and the VNFFG might use tunneling protocols like VXLAN. The dynamic allocation and implementation of these networking paths will have different performance characteristics depending on the methods used. The path implementation mechanism becomes a variable in the network testing of the NSs. The methodology used to test the various mechanisms should largely remain the same, and as usual, the test environment should remain constant for each of the tests, focusing on varying the path establishment method.

Scaling refers to the change in allocation of resources to a VNF or NS. It happens dynamically at run-time, based on defined policies and triggers. The triggers can be network, compute or storage based. Scaling can allocate more resources in times of need, or reduce the amount of resources allocated when the demand is reduced. The SUT in this case becomes much larger than the VNF itself: MANO controls how scaling is done based on policies, and then allocates the resources

accordingly in the NFVI. Essentially, the testing of scaling includes the entire NFV architecture components into the SUT.

4.11.3. Opportunities

Softwarization of networking functionality leads to softwarization of test as well. As Physical Network Functions (PNF) are being transformed into VNFs, so have the test tools. This leads to the fact that test tools are also being controlled and executed in the same environment as the VNFs are. This presents an opportunity to include VNF-based test tools along with the deployment of the VNFs supporting the services of the service provider into the host data centers. Tests can therefore be automatically executed upon deployment in the target environment, for each deployment, and each service. With PNFs, this was very difficult to achieve.

This new concept helps to enable modern concepts like DevOps and Continuous Integration and Continuous Deployment in the NFV environment. The CI/CD pipeline supports this concept. It consists of a series of tools, among which immediate testing is an integral part, to deliver software from source to deployment. The ability to deploy the test tools themselves into the production environment stretches the CI/CD pipeline all the way to production deployment, allowing a range of tests to be executed. The tests can be simple, with a goal of verifying the correct deployment and networking establishment, but can also be more complex, like testing VNF functionality.

5. Technology Gaps and Potential IETF Efforts

Table 1 correlates the open network virtualization research areas identified in this document to potential IETF and IRTF groups that could address some aspects of them. An example of a specific gap that the group could potentially address is identified in parenthetical beside the group name.

Open Research Area	Potential IETF/IRTF Group
1-Guaranteeing QoS	IPPM WG (Measurements of NFVI)
2-Performance improvement	SFC WG, NFVRG (energy driven orchestration)
3-Multiple Domains	NFVRG (multi-domain orchestration)
4-Network Slicing	NVO3 WG, NETSLICES bar BoF (multi-tenancy support)
5-Service Composition	SFC WG (SFC Mgmt and Config)
6-End-user device virtualization	N/A
7-Security	N/A
8-Separation of control concerns	NFVRG (separation between transport control and services)
9-Testing	NFVRG (testing of scaling)
10-Function placement	NFVRG, SFC WG (VNF placement algorithms and protocols)

Table 1: Mapping of Open Research Areas to Potential IETF Groups

6. NFVRG focus areas

Table 2 correlates the currently identified NFVRG topics of interests/focus areas to the open network virtualization research areas enumerated in this document. This can help the NFVRG in identifying and prioritizing research topics. The current list of NFVRG focus points is the following:

- Re-architecting functions, including aspects such as new architectural and design patterns (e.g., containerization, statelessness, serverless, control/data plane separation), SDN integration, and proposals on programmability.

- New management frameworks, considering aspects related to new OAM mechanisms (e.g., configuration control, hybrid descriptors) and lightweight MANO proposals.

- Techniques to guarantee low latency, resource isolation, and other dataplane features, including hardware acceleration, functional offloading to dataplane elements (including NICs), and related approaches.

- Measurement and benchmarking, addressing both internal measurements and external applications.

NFVRG Focus Point	Open Research Area
1-Re-architecting functions	- Performance improvem.
	- Network Slicing
	- Guaranteeing QoS
	- Security
	- End-user device virt.
	- Separation of control
2-New management frameworks	- Multiple Domains
	- Service Composition
	- End-user device virt.
3-Low latency, resource isolation, etc	- Performance improvem.
	- Separation of control
4-Measurement and benchmarking	- Guaranteeing QoS
	- Testing

Table 2: Mapping of NFVRG Focus Points to Open Research Areas

7. IANA Considerations

N/A.

8. Security Considerations

This is an informational document, which therefore does not introduce any security threat. Research challenges and gaps related to security and privacy have been included in Section 4.8.

9. Acknowledgments

The authors want to thank Dirk von Hugo, Rafa Marin, Diego Lopez, Ramki Krishnan, Kostas Pentikousis, Rana Pratap Sircar, Alfred Morton, Nicolas Kuhn, Saumya Dikshit, Fabio Giust, Evangelos Haleplidis, Angeles Vazquez-Castro, Barbara Martini, Jose Saldana and Gino Carrozzo for their very useful reviews and comments to the document. Special thanks to Pedro Martinez-Julia, who provided text for the network slicing section.

The authors want to also thank Dave Oran and Michael Welzl for their very detailed IRSG reviews.

The work of Carlos J. Bernardos and Luis M. Contreras is partially supported by the H2020 5GEx (Grant Agreement no. 671636) and 5G-TRANSFORMER (Grant Agreement no. 761536) projects.

10. Informative References

[dynamic_chaining]
Martini, B. and F. Paganelli, "A Service-Oriented Approach for Dynamic Chaining of Virtual Network Functions over Multi-Provider Software-Defined Networks", Future Internet vol. 8, no. 2, June 2016.

[dynamic_placement]
Clayman, S., Maini, E., and A. Galis, "The dynamic placement of virtual network functions", 2014 IEEE Network Operations and Management Symposium (NOMS) pp. 1-9, May 2014.

[etsi_gs_nfv_003]
ETSI NFV ISG, "Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV", ETSI GS NFV 003 V1.2.1 NFV 003, December 2014, <http://www.etsi.org/deliver/etsi_gs/NFV/001_099/003/01.02.01_60/gs_NFV003v010201p.pdf>.

[etsi_gs_nfv_eve005]
ETSI NFV ISG, "Network Functions Virtualisation (NFV); Ecosystem; Report on SDN Usage in NFV Architectural Framework", ETSI GS NFV-EVE 005 V1.1.1 NFV-EVE 005, December 2015, <http://www.etsi.org/deliver/etsi_gs/NFV-EVE/001_099/005/01.01.01_60/gs_NFV-EVE005v010101p.pdf>.

[etsi_gs_nfv_per_001]
ETSI GS NFV-PER 001 V1.1.2, "Network Functions Virtualisation (NFV); NFV Performance & Portability Best Practises", ETSI GS NFV-PER 001 V1.1.2 NFV-PER 001, December 2014, <http://www.etsi.org/deliver/etsi_gs/NFV-PER/001_099/001/01.01.02_60/gs_NFV-PER001v010102p.pdf>.

[etsi_gs_nfv_sec_001]
ETSI GS NFV-SEC 001 V1.1.1, "Network Functions Virtualisation (NFV); NFV Security; Problem Statement", ETSI GS NFV-SEC 001 V1.1.1 NFV-SEC 001, October 2014, <http://www.etsi.org/deliver/etsi_gs/NFV-SEC/001_099/001/01.01.01_60/gs_NFV-SEC001v010101p.pdf>.

[etsi_nvf_whitepaper_3]
"Network Functions Virtualisation (NFV). White Paper 3", October 2014.

[google_sdn_wan]
Sushant Jain et al., "B4: experience with a globally-deployed Software Defined WAN", Proceedings of the ACM SIGCOMM 2013 , August 2013.

[I-D.bagnulo-nfvrg-topology]
Bagnulo, M. and D. Dolson, "NFVI PoP Network Topology: Problem Statement", draft-bagnulo-nfvrg-topology-01 (work in progress), March 2016.

[I-D.bernardos-nfvrg-multidomain]
Bernardos, C., Contreras, L., Vaishnavi, I., Szabo, R., Li, X., Paolucci, F., Sgambelluri, A., Martini, B., Valcarenghi, L., Landi, G., Andrushko, D., and A. Mourad, "Multi-domain Network Virtualization", draft-bernardos-nfvrg-multidomain-04 (work in progress), March 2018.

[I-D.defoy-netslices-3gpp-network-slicing]
Foy, X. and A. Rahman, "Network Slicing - 3GPP Use Case", draft-defoy-netslices-3gpp-network-slicing-02 (work in progress), October 2017.

[I-D.gdmb-netslices-intro-and-ps]
Galis, A., Dong, J., kiran.makhijani@huawei.com, k., Bryant, S., Boucadair, M., and P. Martinez-Julia, "Network Slicing - Introductory Document and Revised Problem Statement", draft-gdmb-netslices-intro-and-ps-02 (work in progress), February 2017.

[I-D.geng-coms-problem-statement]
Geng, L., Slawomir, S., Qiang, L., kiran.makhijani@huawei.com, k., Galis, A., and L. Contreras, "Problem Statement of Common Operation and Management of Network Slicing", draft-geng-coms-problem-statement-04 (work in progress), March 2018.

[I-D.irtf-sdnrg-layered-sdn]
Contreras, L., Bernardos, C., Lopez, D., Boucadair, M., and P. Iovanna, "Cooperating Layered Architecture for SDN", draft-irtf-sdnrg-layered-sdn-01 (work in progress), October 2016.

[I-D.marin-sdnrg-sdn-aaa-mng]
Lopez, R. and G. Lopez-Millan, "Software-Defined Networking (SDN)-based AAA Infrastructures Management", draft-marin-sdnrg-sdn-aaa-mng-00 (work in progress), November 2015.

[I-D.mlk-nfvrg-nfv-reliability-using-cots]
Mo, L. and B. Khasnabish, "NFV Reliability using COTS Hardware", draft-mlk-nfvrg-nfv-reliability-using-cots-01 (work in progress), October 2015.

[I-D.natarajan-nfvrg-containers-for-nfv]
natarajan.sriram@gmail.com, n., Krishnan, R., Ghanwani, A., Krishnaswamy, D., Willis, P., Chaudhary, A., and F. Huici, "An Analysis of Lightweight Virtualization Technologies for NFV", draft-natarajan-nfvrg-containers-for-nfv-03 (work in progress), July 2016.

[I-D.rorosz-nfvrg-vbaas]
Rosa, R., Rothenberg, C., and R. Szabo, "VNF Benchmark-as-a-Service", draft-rorosz-nfvrg-vbaas-00 (work in progress), October 2015.

[intel_10_differences_nfv_cloud]
Torre, P., "Discover the Top 10 Differences Between NFV and Cloud Environments", November 2015, <https://software.intel.com/en-us/videos/discover-the-top-10-differences-between-nfv-and-cloud-environments>.

[itu-t-y.3300]
ITU-T, "Y.3300: Framework of software-defined networking", ITU-T Recommendation Y.3300 (06/14), June 2014, <http://www.itu.int/rec/T-REC-Y.3300-201406-I/en>.

[itu-t-y.3301]
ITU-T, "Y.3301: Functional requirements of software-defined networking", ITU-T Recommendation Y.3301 (09/16), September 2016, <http://www.itu.int/rec/T-REC-Y.3301-201609-I/en>.

[itu-t-y.3302]
ITU-T, "Y.3302: Functional architecture of software-defined networking", ITU-T Recommendation Y.3302 (01/17), January 2017, <http://www.itu.int/rec/T-REC-Y.3302-201701-I/en>.

[multi-domain_5GEx]
Bernardos, C., Geroe, B., Di Girolamo, M., Kern, A., Martini, B., and I. Vaishnavi, "5GEx: Realizing a Europe wide Multi-domain Framework for Software Defined Infrastructures", Transactions on Emerging Telecommunications Technologies vol. 27, no. 9, pp. 1271-1280, September 2016.

[nfv_piecing]
Luizelli, M., Bays, L., and L. Buriol, "Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions", 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM) pp. 98-106, May 2015.

[nfv_sota_research_challenges]
Mijumbi, R., Serrat, J., Gorricho, J-L., Bouten, N., De Turck, F., and R. Boutaba, "Network Function Virtualization: State-of-the-art and Research Challenges", IEEE Communications Surveys & Tutorials Volume: 18, Issue: 1, September 2015.

[ngmn_5G_whitepaper]
"NGMN 5G. White Paper", February 2015.

[omniran] IEEE 802.1CF, "Recommended Practice for Network Reference Model and Functional Description of IEEE 802 Access Network", Draft 1.0 , December 2017.

[onf_tr_521]
ONF, "SDN Architecture, Issue 1.1", ONF TR-521 TR-521, February 2016, <https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR-521_SDN_Architecture_issue_1.1.pdf>.

[openmano_dataplane]
Lopez, D., "OpenMANO: The Dataplane Ready Open Source NFV MANO Stack", March 2015, <https://www.ietf.org/proceedings/92/slides/slides-92-nfvrg-7.pdf>.

[RFC5810] Doria, A., Ed., Hadi Salim, J., Ed., Haas, R., Ed., Khosravi, H., Ed., Wang, W., Ed., Dong, L., Gopal, R., and J. Halpern, "Forwarding and Control Element Separation (ForCES) Protocol Specification", RFC 5810, DOI 10.17487/RFC5810, March 2010, <https://www.rfc-editor.org/info/rfc5810>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7426] Haleplidis, E., Ed., Pentikousis, K., Ed., Denazis, S., Hadi Salim, J., Meyer, D., and O. Koufopavlou, "Software-Defined Networking (SDN): Layers and Architecture Terminology", RFC 7426, DOI 10.17487/RFC7426, January 2015, <https://www.rfc-editor.org/info/rfc7426>.

[RFC7498] Quinn, P., Ed. and T. Nadeau, Ed., "Problem Statement for Service Function Chaining", RFC 7498, DOI 10.17487/RFC7498, April 2015, <https://www.rfc-editor.org/info/rfc7498>.

[RFC7665] Halpern, J., Ed. and C. Pignataro, Ed., "Service Function Chaining (SFC) Architecture", RFC 7665, DOI 10.17487/RFC7665, October 2015, <https://www.rfc-editor.org/info/rfc7665>.

[RFC8030] Thomson, M., Damaggio, E., and B. Raymor, Ed., "Generic Event Delivery Using HTTP Push", RFC 8030, DOI 10.17487/RFC8030, December 2016, <https://www.rfc-editor.org/info/rfc8030>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8172] Morton, A., "Considerations for Benchmarking Virtual Network Functions and Their Infrastructure", RFC 8172, DOI 10.17487/RFC8172, July 2017, <https://www.rfc-editor.org/info/rfc8172>.

[sfc_challenges] Medhat, A., Taleb, T., Elmangoush, A., Carella, G., Covaci, S., and T. Magedanz, "Service Function Chaining in Next Generation Networks: State of the Art and Research Challenges", IEEE Communications Magazine vol. 55, no. 2, pp. 216-223, February 2017.

[virtualization_mobile_device] William D. Sproule, "Virtualization of Mobile Device User Experience", Patent US 9.542.062 B2 , January 2017.

[vnf-p] Moens, H. and Filip De Turck, "VNF-P: A model for efficient placement of virtualized network functions", 10th International Conference on Network and Service Management (CNSM) and Workshop pp. 418-423, 2014.

[vnf_benchmarking] Rosa, R., Rothenberg, C., and R. Szabo, "A VNF Testing Framework Design, Implementation and Partial Results", November 2016, <https://www.ietf.org/proceedings/97/slides/slides-97-nfvrg-06-vnf-benchmarking-00.pdf>.

Authors' Addresses

Carlos J. Bernardos
Universidad Carlos III de Madrid
Av. Universidad, 30
Leganes, Madrid 28911
Spain

Phone: +34 91624 6236
Email: cjbc@it.uc3m.es
URI: http://www.it.uc3m.es/cjbc/

Akbar Rahman
InterDigital Communications, LLC
1000 Sherbrooke Street West, 10th floor
Montreal, Quebec H3A 3G4
Canada

Email: Akbar.Rahman@InterDigital.com
URI: http://www.InterDigital.com/

Juan Carlos Zuniga
SIGFOX
425 rue Jean Rostand
Labege 31670
France

Email: j.c.zuniga@ieee.org
URI: http://www.sigfox.com/

Luis M. Contreras
Telefonica I+D
Ronda de la Comunicacion, S/N
Madrid 28050
Spain

Email: luismiguel.contrerasmurillo@telefonica.com

Pedro Aranda
Universidad Carlos III de Madrid
Av. Universidad, 30
Leganes, Madrid 28911
Spain

Email: pedroandres.aranda@uc3m.es

Pierre Lynch
Ixia

Email: plynch@ixiacom.com

NFV Research Group
Internet-Draft
Intended status: Informational
Expires: March 11, 2017

N. Figueira
Brocade
R. Krishnan
Dell
D. Lopez
Telefonica
S. Wright
AT&T
D. Krishnaswamy
IBM

September 7, 2016

Policy Architecture and Framework for NFV Infrastructures
draft-irtf-nfvrg-nfv-policy-arch-04

Abstract

A policy architecture and framework is discussed to support NFV environments, where policies are used to enforce business rules and to specify resource constraints in a number of subsystems. This document approaches the policy framework and architecture from the perspective of overall orchestration requirements for services involving multiple subsystems. The framework extends beyond common orchestration constraints across compute, network, and storage subsystems to include energy conservation. This document also analyses policy scope, global versus local policies, static versus dynamic versus autonomic policies, policy actions and translations, policy conflict detection and resolution, interactions among policies engines, and a hierarchical policy architecture/framework to address the demanding and growing requirements of NFV environments. These findings may also be applicable to cloud infrastructures in general.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire in May 2015.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Table of Contents

1. Introduction 3
2. Policy Intent Statement versus Subsystem Actions and Configurations 4
3. Global vs Local Policies 5
4. Static vs Dynamic vs Autonomic Policies 7
5. Hierarchical Policy Framework 7
6. Policy Conflicts and Resolution 9
6.1. Soft vs Hard Policy Constraints 11
7. Policy Pub/Sub Bus 12
7.1 Pub/Sub Bus Name Space 16
8. Examples 17
8.1 Establishment of a Multipoint Ethernet Service 17
8.2 Policy-Based NFV Placement and Scheduling 20
8.2.1 Policy Engine Role in NFV Placement and Scheduling . . . 20
8.2.2 Policy-based NFV Placement and Scheduling with OpenStack 21
9. Summary 25
10. IANA Considerations 25
11. Security Considerations 25
12. Contributors 26

13. References 26
13.1. Normative References 26
13.2. Informative References 26
Acknowledgements 28
Authors' Addresses 28

1. Introduction

This document discusses the policy architecture and framework to support Network Function Virtualization (NFV) [11] infrastructures. In these environments, policies are used to enforce business rules and to specify resource constraints, e.g., energy constraints, in a number of subsystems, e.g., compute, storage, network, and etc., and across subsystems. These subsystems correspond to the different "infrastructure domains" identified by the NFV ISG Infrastructure Working Group [8][10][7].

The current work in the area of policy for NFV is mostly considered in the framework of general cloud services, and typically focused on individual subsystems and addressing very specific use cases or environments. For example, [11] addresses network subsystem policy for network virtualization, [19] and [20] are open source projects in the area of network policy as part of the OpenDaylight [21] software defined networking (SDN) controller framework, [18] specifies an information model for network policy, [13] focuses on placement and migration policies for distributed virtual computing, [24] is an open source project in OpenStack [22] to address policy for general cloud environments.

This document approaches policy, policy framework, and policy architecture for NFV services from the perspective of overall orchestration requirements for services involving multiple subsystems, and can be applied to the general case of any cloud-based service. The analysis extends beyond common orchestration constraints across compute, network, and storage subsystems to also include energy conservation constraints applicable to NFV and other environments. The analysis in this document also extends beyond a single virtual Point of Presence (vPoP) or administrative domain to include multiple data centers and networks forming hierarchical domain architectures [12]. The focus of this document is not general policy theory, which has already been intensively studied and documented on numerous publications over the past 10 to 15 years (see [18], [27], [17], [28], and [3] to name a few). This document's purpose is to discuss and document a policy architecture that uses known policy concepts and theories to address the unique requirements of NFV services including multiple vPoPs and networks forming hierarchical domain architectures [12].

With the above goals, this document analyses policy scope, global versus local policies, static versus dynamic versus autonomic policies, policy actions and translations of actions, policy conflict detection and resolution (which can be relevant to resource management in service chains [16]), the interactions among policies engines from the different vPoPs and network subsystems, and a hierarchical policy architecture/framework to address the demanding and growing requirements of NFV environments. These findings may also be applicable to cloud infrastructures in general.

2. Policy Intent Statement versus Subsystem Actions and Configurations

Policies define which states of deployment are in compliance with the policy, and, by logic negation, which ones are not. The compliance statement in a policy may define specific actions, e.g., "a given customer is [not allowed to deploy VNF X]", where VNF refers to a Virtual Network Function, or quasi-specific actions, e.g., "a given customer [must be given platinum treatment]." Quasi-specific actions differ from the specific ones in that the former requires an additional level of translation or interpretation, which will depend on the subsystems where the policy is being evaluated, while the latter does not require further translation or interpretation.

In the previous examples, "VNF X" defines a specific VNF type, i.e., "X" in this case, while "platinum treatment" could be translated to an appropriate resource type depending on the subsystem. For example, in the compute subsystem this could be translated to servers of a defined minimum performance specification, while in the network subsystem this could be translated to a specific Quality of Service (QoS) level treatment.

The actions defined in a policy may be translated to subsystem configurations. For example, when "platinum treatment" is translated to a specific QoS level treatment in a networking subsystem, one of the outcomes (there can be multiple ones) of the policy could be the configuration of network elements (physical or virtual) to mark that customer’s traffic to a certain DSCP (DiffServ Code Point) level (Figure 1). Some may refer to the QoS configuration above as a policy in itself, e.g., [27], [28], [4], and [17]. In this document, such domain configurations are called policy enforcement technologies to set them apart from the actual policy intent, e.g., "a given customer must be given platinum treatment" as in the above example.

Describing intent using a high-level policy language instead of directly describing configuration details allows for the decoupling of the desired intent from the actual configurations, which are subsystem dependent, as shown in the previous example (Figure 1). The

translation of a policy into appropriate subsystem configurations requires additional information that is usually subsystem and technology dependent. Therefore, policies should not be written in terms of policy enforcement technologies. Policies should be translated at the subsystems using the appropriate policy provides a few examples where the policy "a given customer must be given platinum treatment" is translated to appropriate configurations at the respective subsystems.

The above may sound like a discussion about "declarative" versus "imperative" policies. We are actually postulating that "imperative policy" is just a derived subsystem configuration using an appropriate policy enforcement technology to support an actually intended policy.

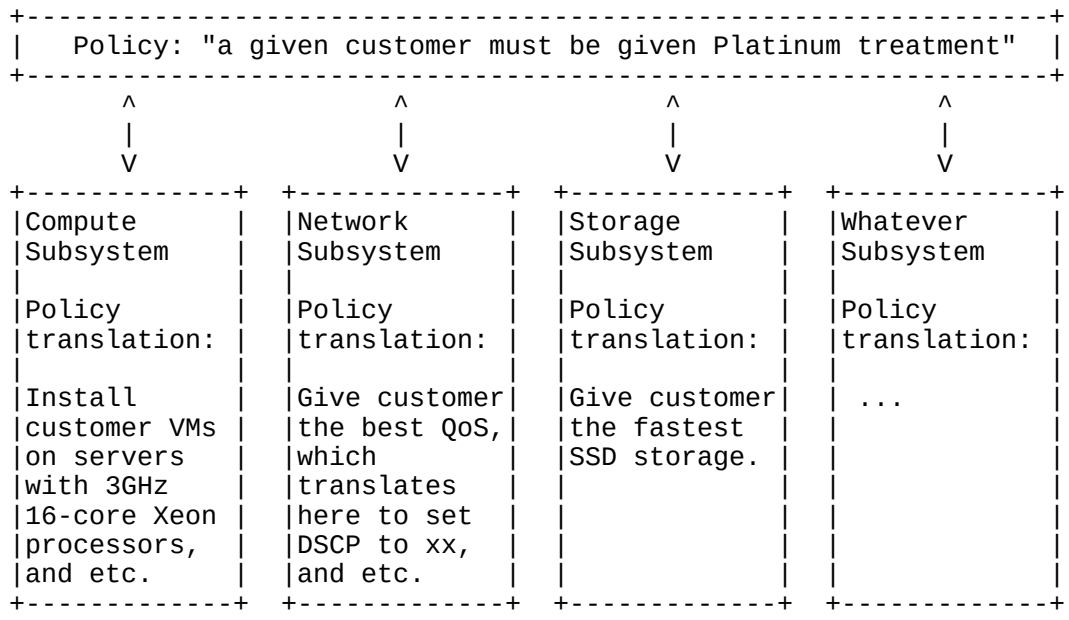


Figure 1: Example of Subsystem Translations of Policy Actions

3. Global vs Local Policies

Some policies may be subsystem specific in scope, while others may have broader scope and interact with multiple subsystems. For example, a policy constraining certain customer types (or specific customers) to only use certain server types for VNF or Virtual Machine (VM) deployment would be within the scope of the compute subsystem. A policy dictating that a given customer type (or specific

customers) must be given "platinum treatment" could have different implications on different subsystems. As shown in Figure 1, that "platinum treatment" could be translated to servers of a given performance specification in a compute subsystem and storage of a given performance specification in a storage subsystem.

Policies with broader scope, or global policies, would be defined outside affected subsystems and enforced by a global policy engine (Figure 2), while subsystem-specific policies or local policies, would be defined and enforced at the local policy engines of the respective subsystems.

Examples of sub-system policies can include thresholds for utilization of sub-system resources, affinity/anti-affinity constraints with regard to utilization or mapping of sub-system resources for specific tasks, network services, or workloads, or monitoring constraints regarding under-utilization or over-utilization of sub-system resources.

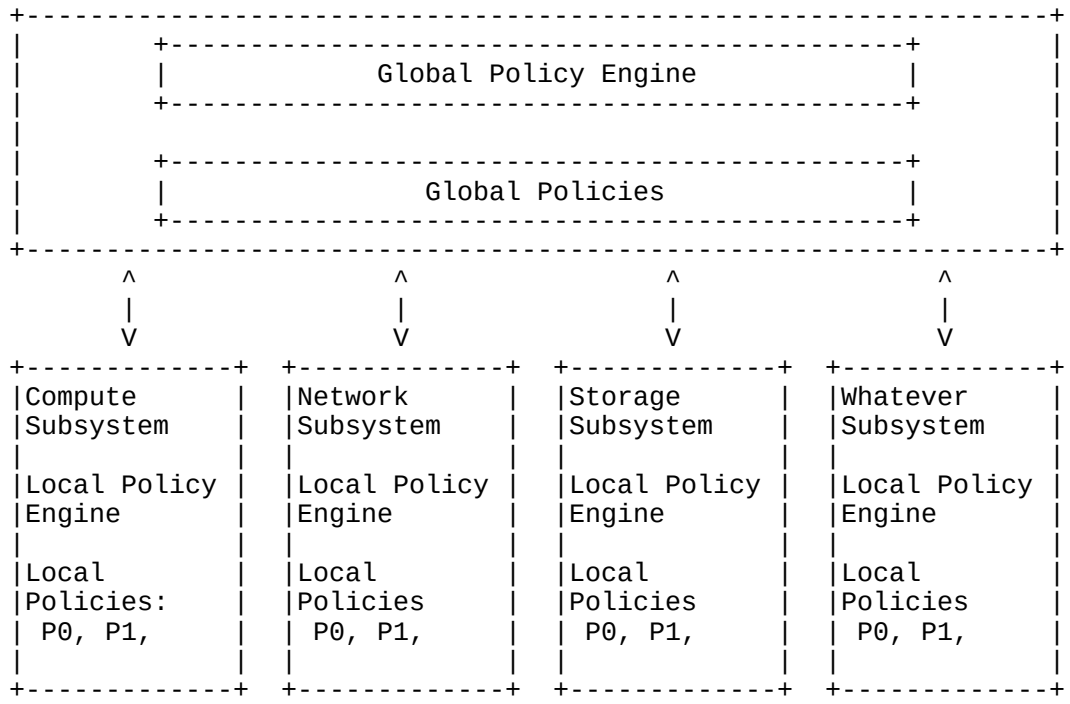


Figure 2: Global versus Local Policy Engines

4. Static vs Dynamic vs Autonomic Policies

Policies can be defined based on a diverse set of constraints and sources, e.g., an operator’s energy cost reduction policy based on the time-varying energy rates imposed by a utility company supplying energy to a given region or an operator’s "green" policy based on the operator’s green operational requirements, which may drive a reduction in energy consumption despite of potentially increased energy costs.

While "static" policies can be imposed based on past learned behavior in the systems, "dynamic" policies can be define that would override "static" policies due to dynamically varying constraints. For example, if a system needs to provided significantly additional scaling of users in a given geographic area due to a sporting or a concert event at that location, then a dynamic policy can be superimposed to temporarily override a static policy to support additional users at that location for a certain period of time. Alternatively, if energy costs significantly rise on a particular day, then an energy cost threshold could be dynamically raised to avoid policy violation on that day.

Support for autonomic policies may also be required, such as an auto-scaling policy that allows a resource (compute/storage/network/energy resource) to be scaled up or down as needed. For example, a policy specifying that a VNF can be scaled up or down to accommodate traffic needs.

5. Hierarchical Policy Framework

So far, we have referenced compute, network, and storage as subsystems examples. However, the following subsystems may also support policy engines and subsystem specific policies:

- SDN Controllers, e.g., OpenDaylight [21].
- OpenStack [22] components such as, Neutron, Cinder, Nova, and etc.
- Directories, e.g., LDAP, ActiveDirectory, and etc.
- Applications in general, e.g., standalone or on top of OpenDaylight or OpenStack.
- Physical and virtual network elements, e.g., routers, firewalls, application delivery controllers (ADCs), and etc.
- Energy subsystems, e.g., OpenStack Neat [25].

Therefore, a policy framework may involve a multitude of subsystems. Subsystems may include other lower level subsystems, e.g., Neutron [26] would be a lower level subsystem in the OpenStack subsystem. In

other words, the policy framework is hierarchical in nature, where the policy engine of a subsystem may be viewed as a higher level policy engine by lower level subsystems. In fact, the global policy engine in Figure 2 could be the policy engine of a Data Center subsystem and multiple Data Center subsystems could be grouped in a region containing a region global policy engine. In addition, one could define regions inside regions, hierarchically, as shown in Figure 3.

Metro and wide-area network (WAN) used to interconnect data centers would also be independent subsystems with their own policy engines.

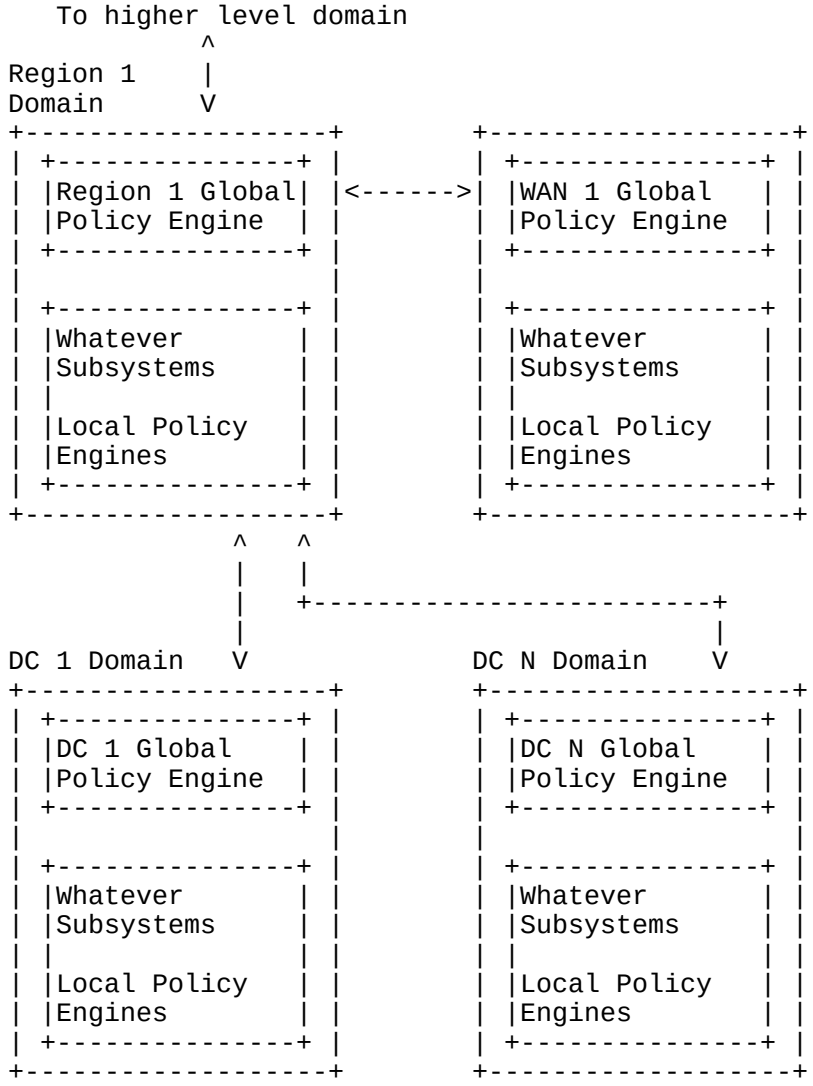


Figure 3: A Hierarchical Policy Framework

6. Policy Conflicts and Resolution

Policies should be stored in databases accessible by the policy engines. For example, the local policies defined for the Compute subsystem in Figure 2 would be stored in a database accessible by the local policy engine in that subsystem.

As a new policy is added to a subsystem, the subsystem’s policy engine should perform conflict checks. For example, a simple conflict would be created if a new policy states that "customer A must not be allowed to use VNF X", while an already existing policy states that "customer A is allowed to use VNF X". In this case, the conflict should be detected and an appropriate policy conflict resolution mechanism should be initiated.

The nature of the policy conflict resolution mechanism would depend on how the new policy is being entered into the database. If an administrator is manually attempting to enter that policy, the conflict resolution could entail a warning message and rejection of the new policy. The administrator would then decide whether or not to replace the existing policy with the new one.

When policies are batched for later inclusion in the database, the administrator should run a preemptive conflict resolution check on those policies before committing to include them in the database at a future time. However, running a preemptive conflict resolution check does not guarantee that there will be no conflicts at the time the batched policies are actually included in the database, since other policies could have been added in the interim that cause conflicts with those batched policies.

To avoid conflicts between batched policies waiting for later inclusion in the database and new policies being immediately added to the database, one could run a preemptive conflict resolution check against database policies and also batched policies every time new policies are added to the database. However, this may not be sufficient in case of separate administrative domains. A region administration could define batched polices to be pushed to the Compute subsystem of a Data Center at a later time. However, the Compute subsystem may be a separate administrative domain from that of the region administrative domain. In this case, the Compute subsystem may not be allowed to run preemptive policy conflict checks against the batched policies defined at the region administrative domain. Thus, there is a need for a reactive policy conflict resolution mechanism besides preemptive techniques.

The above discussions implicitly assumed that policies are individually evaluated for conflicts and individually committed without regard to other policies. However, a set of policies could be labeled as part of a same "Commit Group", where the whole set of policies in the Commit Group must be committed for a desired result to be obtained. In this case, the conflict resolution mechanism would need to verify that none of the policies in the Commit Group conflicts with currently committed policies before the Commit Group is added (in other words, committed) to the policy database.

The Commit Group conflict detection mechanism and subsequent addition to the database should be implemented as an atomic process, i.e., no changes to the policy database should be allowed by other processes until either the whole Commit Group is checked and committed or a conflict is detected and the process stopped, to avoid multiple writers issues.

The above described atomic Commit Group conflict detection and policy commit mechanism would eliminate the need for Commit Group rollback. A rollback could be required if policies in a Commit Group were to be checked for conflicts and committed one by one, since the detection of a subsequent policy conflict in the Commit Group would require the rollback of previously committed policies in that group.

6.1. Soft vs Hard Policy Constraints

Policies at any level in the policy hierarchy can be either soft or hard. A soft policy imposes a soft constraint in a system that can be violated without causing any catastrophic failure in the system. A hard policy imposes a hard constraint in the system that must not be violated. An example of a soft constraint is the degree of underutilization (for example, a 40% utilization threshold could be used as a soft constraint) of compute servers with regard to CPU utilization. In such a case, when this soft constraint is violated, the system continues to function, although it may consume more energy (due to a non-linear dependence of the energy utilization as a function of the CPU utilization) compared to a task allocation across multiple servers after workload consolidation is performed. Alternatively, a soft constraint could be violated if the network bandwidth exceeds a certain fraction (say 80%) of the available bandwidth, the energy utilization exceeds a certain value, or the memory utilization falls below a certain value (say 30%). An example of a hard constraint could be to disallow a desired mean CPU utilization across allocated workloads in a compute subsystem to exceed a certain high threshold (such as 90%), or to disallow the number of CPUs allocated for a set of workloads to exceed a certain value, or to disallow the network bandwidth across workloads to exceed a certain value (say 98% of the maximum available bandwidth).

When considering policy conflicts, violation of policies across hard policy constraints is undesirable, and must be avoided. A conflict resolution could be possible by relaxing one or more of the hard constraints to an extent that is mutually satisfactory to the imposers of the policies. Alternatively, as discussed earlier, a new hard policy that conflicts with existing policies is not admitted into the system. Violation of policies across soft policy constraints or between one or more hard policy constraints and one or more soft policy constraints can be allowed, such that one or more soft policy

constraints are violated without hard constraints being violated. Despite soft constraints being violated, it is desirable to have a region of operating conditions that would allow the system to operate. For admission of new policy constraints, whether hard or soft, one should ensure that the overall system has a feasible region for operation given the existing constraints, and the new constraints under consideration. When such a feasible region is not possible, one can consider relaxing one or more of the existing or new constraints to allow such policies to be admitted.

7. Policy Pub/Sub Bus

In the previous section, we considered policy conflicts within a same level subsystem. For example, new local policies added to the Compute subsystem conflicting with existing local policies at that subsystem. However, more subtle conflicts are possible between global and local policies.

A global policy may conflict with subsystems' local policies. Consider the following Compute subsystem local policy: "Platinum treatment must be provided using server of type A."

The addition of the Global policy "Platinum treatment must be provided using server subtype A-1" would intrude into the Compute subsystem by redefining the type of server to be used for a particular service treatment. While one could argue that such global policy should not be permitted, this is an event that requires detection and proper resolution. A possible resolution is for the Compute subsystem to import the more restrictive policy into its local database. The original local policy would remain in the database as is along with the new restrictive policy. The local policy engine would then enforce the more restricted form of the policy after this policy change, which could make already existing resource allocations non-compliant and requiring corrective actions, e.g., Platinum treatment being currently provided by a server of type A instead of a server of type A-1.

If the new Global policy read "Platinum treatment must be provided using server of types A or B" instead, the Compute subsystem would not need to do anything different, since the Compute subsystem has a more restrictive local policy in place, i.e., "Platinum treatment must be provided using server of type A."

The above examples demonstrate the need for subsystems to subscribe to policy updates at the Global policy level. A policy publication/subscription (pub/sub) bus would be required as shown in Figure 4.

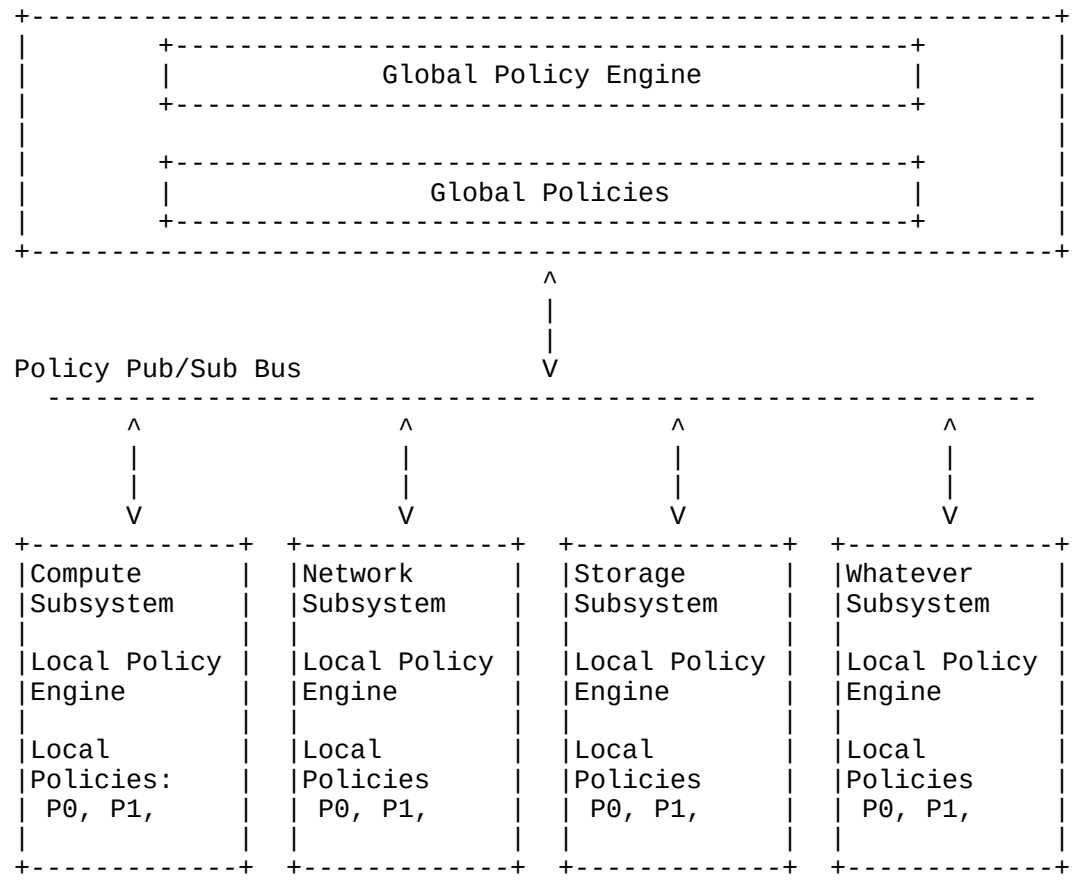


Figure 4: A Policy Pub/Sub Bus

A policy conflict may force policies to change scope. Consider the following existing policies in a Data Center:

Compute subsystem policy: "Platinum treatment requires a server of type A or B."

Storage subsystem policy: "Platinum treatment requires a server storage of type X or Y."

Now consider the outcome of adding the following new Global policy: "Platinum treatment requires a server of type A when storage of type X is used or a server of type B when storage of type Y is used."

This new Global policy intrudes into the Compute and Storage subsystems. Again, one could argue that such global policy should not

be permitted. Nevertheless, this is an event that would require detection and proper resolution. This Global policy causes a conflict because the Compute and Storage subsystems can no longer independently define whether to use a server of type A or B or storage of type X or Y, respectively. If the Compute subsystem selects server of type A for a customer and the Storage subsystem selects storage of type Y for that same customer service the Global policy is violated. In conclusion, if such global policy is permitted, the Compute and Storage subsystems can no longer make such selections. A possible conflict resolution is for the Compute and Storage subsystems to relegate policy enforcement for such resources to the Global policy engine. In this example, the Global Policy engine would need to coordinate with the Compute and Storage subsystems the selection of appropriate resource types to satisfy that policy.

That suggests that the policy pub/sub bus should in fact be an integral part of the northbound service interfaces (NBI) of the subsystems in the hierarchy. Such issue was analyzed in [12], where the concepts of service capability, service availability, and service instantiation were introduced to enable a higher-level subsystem to properly select services and resources from lower-level subsystems to satisfy existing policies.

The above example demonstrates again the need for subsystems to subscribe to policy updates at the higher policy level (the Global policy level in this example) as shown in Figure 4.

If, as demonstrated, a Global policy may "hijack" or "nullify" local policies of subsystems, what exactly makes the scope of a policy local versus global then?

Proposition: A Local Policy does not affect the compliance state imposed by global Policies or the local policies of other subsystems.

The above non-exhaustive examples demonstrate that global and local policies may conflict in subtle ways. Policy conflicts will also policy framework requires a policy pub/sub bus between all levels to allow for conflict detection, conflict information propagation, and conflict resolution (Figure 5).

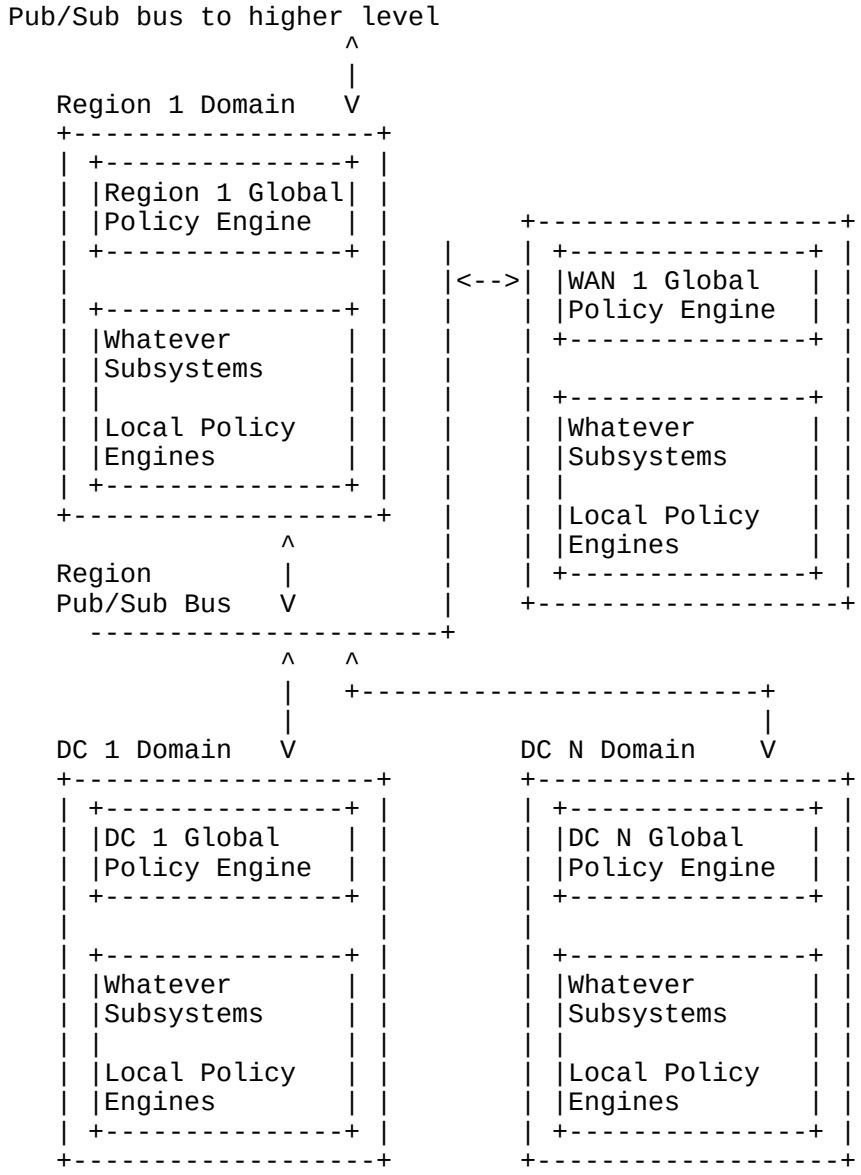


Figure 5: Pub/Sub Bus - Hierarchical Policy Framework

7.1 Pub/Sub Bus Name Space

As described above, a higher tier policy engine would communicate policies to lower tier policy engines using a policy pub/sub bus. Conversely, lower tier policy engines would communicate their configured policies and services to the higher tier policy engine using the same policy pub/sub bus. Such communications require each policy pub/sub bus to have a pre-defined/pre-configured policy "name space". For example, a pub/sub bus could define services using the name space "Platinum", "Gold", and "Silver". A policy could then be communicated over that pub/sub bus specifying a Silver service requirement.

In a hierarchical policy framework, a policy engine may use more than one policy pub/sub bus, e.g., a policy pub/sub bus named "H" to communicate with a higher tier policy engine and a policy pub/sub bus named "L" to communicate with lower tier policy engines. As the name spaces of policy pub/sub buses H and L may be different, the policy engine would translate policies defined using the policy pub/sub bus H name space to policies defined using the policy pub/sub bus L name space, and vice-versa. For example, suppose that the policy pub/sub bus H name space defines service levels named Platinum, Gold, and Silver and that the policy pub/sub bus L name space does not define such service levels, but defines QoS levels High, Medium, and Low. The policy engine would translate a policy to support Silver service, which is written using the policy pub/sub bus H name space, to an appropriate policy (or set of policies) written using the policy pub/sub bus L name space, e.g., QoS level Low.

The described policy framework does not preclude use of a single/same name space throughout the hierarchy. However, to promote scalability and limit complexity, the name spaces of higher tier policy pub/sub buses should be limited to support higher level policies, since the higher the degree of specificity allowed at the higher tiers of the policy hierarchy the higher the operational complexity.

8. Examples

8.1 Establishment of a Multipoint Ethernet Service

Consider a service provider with an NFV infrastructure (NFVI) with multiple vPoPs, where each vPoP is a separate administrative domain. A customer "Z" requests the creation of a "multipoint Silver Ethernet service" between three of its sites, which are connected to service provider's vPoPs A, B, and C. The customer request is carried out using a service provider self-service web portal, which offers customers multiple service type options, e.g., point-to-point and multipoint Ethernet services, and multiple service levels per service type, e.g., Platinum, Gold, and Silver Ethernet services, where the different service levels may represent different service specifications in terms of QoS, latency, and etc. The web portal relays the request to a service provider's OSS/BSS. The service request is stored as a service policy that reads as: "multipoint Silver Ethernet service between vPoPs A, B, and C for customer Z".

The OSS/BSS subsystem would communicate the service request and requirements as a policy to a global NFV Orchestrator (NFVO) subsystem using the name space of the pub/sub bus between these two subsystems (see Section 7.1). For example, the OSS/BSS could translate "Silver" service level into a policy defined using a Network Service (NS) Flavor ID, as defined by the name space of pub/sub bus between the OSS/BSS and the NFVO.

The service provider's vPoP NFV infrastructure architecture may vary depending on the size of each vPoP and other specific needs of the service provider. For example, a vPoP may have a local NFVO subsystem and one or more local Virtual Infrastructure Manager (VIM) subsystems (as in Figure 6). In this case, the global NFVO subsystem would communicate the service request and requirements as a policy to the local NFVOs of vPoPs A, B, and C.

At each vPoP, the local NFVO (and VNF Managers) would carry out the requested service policy based on the local configurations of respective subsystems and current availability of resources. For example, the requested service may translate in vPoP A to use a specific vCE (virtual customer edge) VNF type, say vCE_X, while in vPoP B it may translate to use a different vCPE VNF type, say vCPE_Y, due to local subsystem configurations (refer to Section 2 for a discussion on subsystem actions and configurations). Similarly, the local VIM interaction with the vPoP's compute, network, and storage subsystems may lead to local configurations of these subsystems driven by the translation of the policies received by the respective subsystems (see Section 3 for a discussion on global versus local policies). Note that the original policy at the OSS/BSS level is

translated throughout the policy hierarchy by respective policy engines to fit the name spaces of the associated pub/sub buses in the hierarchy.

The global NFVO subsystem could also communicate a policy defining the requirements to create a multipoint Ethernet service between vPoPs A, B, and C to a WAN infrastructure management (WIM) subsystem (not shown in Figure 6). The WIM subsystem could oversee a hierarchy of other subsystems, e.g., SDN multi-domain architecture of controllers deployed as a hierarchy of network regions (see [12]). Network subsystems would translate locally received policies to local configurations (again, refer to Section 2 for a discussion on subsystem actions and configurations).

As depicted in Figure 6, policy communications would employ a policy pub/sub bus between the subsystems' policy engines in the policy hierarchy (see Section 7). The global NFVO subsystem should have visibility into the policies defined locally at each vPoP to be able to detect any potential global policy conflicts, e.g., a local vPoP administrator could add a local policy that violates or conflicts with a global policy. In addition, the global NFVO subsystem would benefit from being able to import the currently configured services at each vPoP. The global NFVO would use such information to monitor global policy conformance and also to facilitate detection of policy violations when new global policies are created, e.g., a global level administrator is about to add a new global policy that, if committed, would make certain already configured services a violation of the policy. The publication of subsystem service tables for consumption by a global policy engine is a concept used in the Congress [24] OpenStack [22] project.

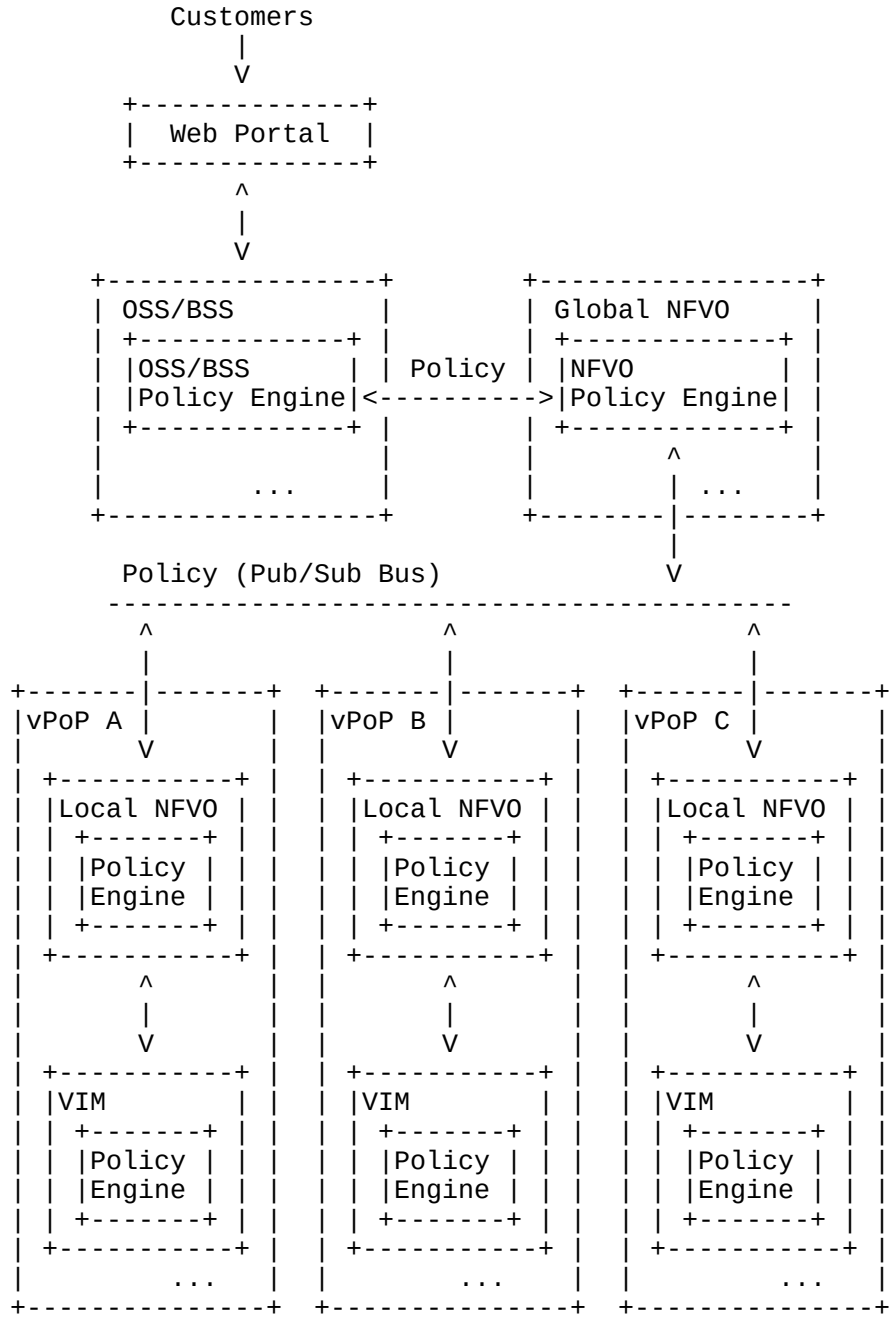


Figure 6: Simplified view of a service provider's NFV Architecture: Multipoint Ethernet Service Example

8.2 Policy-Based NFV Placement and Scheduling

One of the goals of NFV is to allow a Service Provider (SP) offer the NFV infrastructure as a service to other Service Providers as Customers - this is called NFVIaaS [6]. In this context, it may be desirable for a Service Provider to run virtual network elements (e.g., virtual routers, virtual firewalls, and etc.) as virtual machine instances inside the infrastructure of another Service Provider. In this document, we call the former a "customer SP" and the latter an "NFVIaaS SP."

There are many reasons for a customer SP to require the services of an NFVIaaS SP, including: to meet performance requirements (e.g., latency or throughput) in locations where the customer SP does not have physical data center presence, to allow for expanded customer reach, regulatory requirements, and etc.

As VNFs are virtual machines, their deployment in such NFVIaaS SPs would share some of the same placement restrictions (i.e., placement policies) as those intended for Cloud Services. However, VNF deployment will drive support for unique placement policies, given VNF’s stringent service level specifications (SLS) required/imposed by customer SPs. Additionally, NFV DCs or NFV PoPs [8] often have capacity, energy and other constraints - thus, optimizing the overall resource usage based on policy is an important part of the overall solution.

This section describes an example [15] of a global policy written in Datalog [3] applicable to compute to promote energy conservation for the NFVIaaS use case in an OpenStack framework. The goal of that global policy is to address the energy efficiency requirements described in the ETSI NFV Virtualization Requirements [9].

A related energy efficiency use case using analytics-driven policies in the context of OpenStack Congress [24] policy as a service was presented and demonstrated at the Vancouver OpenStack summit [14], where the Congress policy engine delegated VM placement to a VM placement engine that migrated under-utilized VMs to save energy.

8.2.1 Policy Engine Role in NFV Placement and Scheduling

A policy engine may facilitate policy-based resource placement and scheduling of VMs in an NFVIaaS environment. In this role, a policy engine (Figure 7) would determine optimized placement and scheduling choices based on the constraints specified by current resource placement and scheduling policies. The policy engine would evaluate such policies based on event triggers or programmable timers.

In one instantiation, a policy engine would interface with a "Measurement Collector" (e.g., OpenStack Ceilometer [23]) to periodically retrieve instantaneous per-server CPU utilization, in order to compute a table of per-server average CPU utilization. In an alternative instantiation, the Measurement Collector could itself compute per-server average CPU utilization and provide that information to the policy engine. The latter approach would reduce overhead, since it would avoid too frequent pulling of stats from the Measurement Collector.

Other average utilization parameters such as VM CPU utilization, VM Memory utilization, VM disk read IOPS, Network utilization/latency, and etc. could also be used by the policy engine to enforce other types of placement policies.

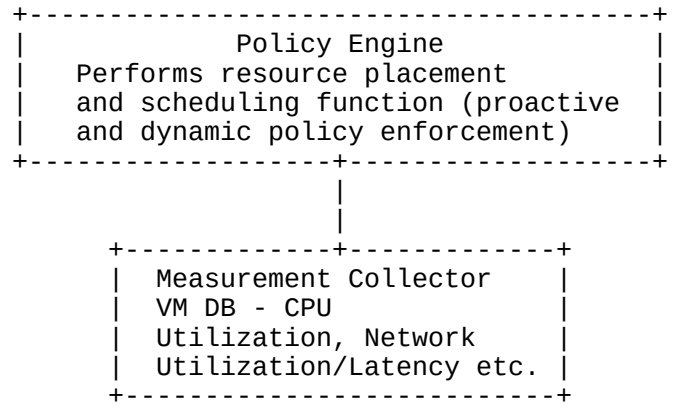


Figure 7: NFVIaaS Architecture for Policy Based Resource Placement and Scheduling

In the ETSI NFV Architectural Framework [7], the Policy Engine is part of the Orchestrator and the Measurement Collector is part of the Virtual Infrastructure Manager (VIM).

8.2.2 Policy-based NFV Placement and Scheduling with OpenStack

Consider an NFVIaaS SP that owns a multitude of mini NFV data centers managed by OpenStack [22] where:

- The Policy Engine function is performed by OpenStack Congress [24].

- The Measurement Collector function is performed by OpenStack Celiometer [23].

- The Policy Engine has access to the OpenStack Nova database that stores records of mapping of virtual machines to physical servers.

An exemplary mini NFV DC configuration is used in this example, as described below:

- 210 physical servers in 2U rack server configuration spread over 10 racks.

- 4 types of physical servers each with a different system configuration and from a particular manufacturer. It is possible that the servers are all from the same or different manufacturers. For the purpose of this example, a server "type 1" is described. Server type 1 has 32 virtual CPUs and 128GB DRAM from manufacturer x. Assume 55 physical servers of type 1 per mini NFV DC.

- 2 types of instances large.2 and large.3, which are described in Table 1. Each parameter has a minimum guarantee and a maximum usage limit.

Instance Type	Virtual CPU Units Minimum Guarantee /Maximum Usage	Memory (GB) Minimum Guarantee /Maximum Usage	Minimum/Maximum Physical Server Utilization (%)
large.2	0/4	0/16	0/12.5
large.3	0/8	0/32	0/25

Table 1: NFVIaaS Instance Types

For the purpose of this example, the Mini NFV DC topology is considered static -- the above topology, including the network interconnection, is available through a simple file-based interface.

Policy 1 (an exemplary NFV policy): In a descriptive language, Policy 1 is as follows - "For physical servers of type 1, there can be at most only one active physical server with average overall utilization less than 50%." Policy 1 is an example of reactive enforcement. The goal of this policy is to address the energy efficiency requirements described in the ETSI NFV Virtualization Requirements [9].

Policy 2 (another exemplary NFV policy): Policy 2 is designed to protect NFV instances from physical server failures. Policy 2 reads as follows in a descriptive language - "Not more than one VM of the same high availability (HA) group must be deployed on the same physical server". Policy 2 is an example of proactive and reactive enforcement.

Note that there may be cases where there may not be any placement solution respecting both policies given the current DC load. To avoid such cases, Policy 1 could be relaxed to: "Minimize the number of physical servers with average overall utilization less than 50%".

Policy 1 calls for the identification of servers by type. OpenStack Congress would need to support server type, average CPU utilization, and be able to support additional performance parameters (in the future) to support additional types of placement policies. OpenStack Congress would run the policy check periodically or based on trigger events, e.g., deleting/adding VMs. In case OpenStack Congress detects a violation, it would determine optimized placement and scheduling choices so that current placement and scheduling policy are not violated.

A key goal of Policy 1 is to ensure that servers are not kept under low utilization, since servers have a non-linear power profile and exhibit relatively higher power consumption at lower utilization. For example, in the active idle state as much as 30% of peak power is consumed. At the physical server level, instantaneous energy consumption can be accurately measured through IPMI standard. At a customer instance level, instantaneous energy consumption can be approximately measured using an overall utilization metric, which is a combination of CPU utilization, memory usage, I/O usage, and network usage. Hence, Policy 1 is written in terms of overall utilization and not power usage.

The following example addressed the combined effect of Policy 1 and Policy 2.

For an exemplary maximum usage scenario, 53 physical servers could be under peak utilization (100%), 1 server (server-a) could be under partial utilization (62.5%) with 2 instances of type large.3 and 1 instance of type large.2 (this instance is referred as large.2.X1), and 1 server (server-b) could be under partial utilization (37.5%) with 3 instances of type large.2. Call these three instances large.2.X2, large.2.Y and large.2.Z

One HA-group has been configured and two large.2 instances belong to this HA-group. To enforce Policy 2 large.2.X1 and large2.X2 that belong to the HA-group have been deployed in different physical servers, one in server-a and a second in server-b.

When one of the large.3 instances mapped to server-a is deleted from physical server type 1, Policy 1 will be violated, since the overall utilization of server-a falls to 37.5%, since two servers are underutilized (below 50%).

OpenStack Congress, on detecting the policy violation, would use various constraint based placement techniques to find placements for physical server type 1 to address Policy 1 violation without breaking Policy 2. Constraint based placement involves a convex optimization framework [5]. Some of the algorithms that could be considered include linear programming [1], branch and bound [2], interior point methods, equality constrained minimization, non-linear optimization, and etc.

Various new placements are described below:

1) New placement 1: Move 2 of three instances of large.2 running on server-b to server-a. Overall utilization of server-a - 62.5%. Overall utilization of server-b - 25%. large.2.X2 must not be one of the migrated instances.

2) New placement 2: Move 1 instance of large.3 to server-b. Overall utilization of server-a - 12.5%. Overall utilization of server-b - 62.5%.

A third solution consisting of moving 3 large.2 instances to server-a cannot be adopted, since this violates Policy 2. Another policy minimizing the number of migrations could allow choosing between solutions (1) and (2) above.

New placements 2 and 3 could be considered optimal, since they achieve maximal bin packing and open up the door for turning off server-a or server-b and maximizing energy efficiency.

To detect violations of Policy 1, an example of a classification rule is expressed below in Datalog, the policy language used by OpenStack Congress.

The database table exported by the Resource Placement and Scheduler for Policy 1 is described below:

- server_utilization (physical_server, overall_util): Each database entry has the physical server and the calculated average overall utilization.

- vm_host_mapping(vm, server): Each database entry gives the physical server on which VM is deployed.

- anti-affinity_group(vm, group): Each entry gives the anti-affinity group to which a VM belongs.

Policy 1 in a Datalog [3] policy language is as follows:

```
      error (physical_server) :-
        nova: node (physical_server, "type1"),
        resource placement and scheduler:
          server_utilization (physical_server, overall_util < 50)
```

Policy 2 (in Datalog policy language):

```
      error(vm) :-
        anti-affinity_group(vm1, grp1),
        anti-affinity_group(vm2, grp2),
        grp1 != grp2,
        nova: vm host mapping(vm1, server-1),
        nova:  vm host mapping(vm2,server-2),
        server-1 == server-2
```

9. Summary

This document approached the policy framework and architecture from the perspective of overall orchestration requirements for services involving multiple subsystems. The analysis extended beyond common orchestration for compute, network, and storage subsystems to also include energy conservation constraints. This document also analyzed policy scope, global versus local policies, policy actions and translations, policy conflict detection and resolution, interactions among policies engines, and a hierarchical policy architecture/framework to address the demanding and growing requirements of NFV environments, applicable as well to general cloud infrastructures.

The concept of NFV and the proposed policy architecture is applicable to service providers and also enterprises. For example, an enterprise branch office could have capacity and energy constraints similar to that of many service provider NFV vPoPs in constrained environments. This is an aspect that would be worth examining in detail in future work.

10. IANA Considerations

This draft does not have any IANA considerations.

11. Security Considerations

Security issues due to exchanging policies across different administrative domains are an aspect for further study.

12. Contributors

In addition to the authors listed on the front page, the following individuals contributed to the content of Section 8.2 ("Policy-Based NFV Placement and Scheduling"):

Tim Hinrichs
Styra
tim@styra.com

Ruby Krishnaswamy
Orange
ruby.krishnaswamy@orange.com

Arun Yerra
Dell Inc.
arun.yerra@dell.com

13. References

13.1. Normative References

13.2. Informative References

[1] Alevras, D. and Padberg, M. "Linear Optimization and Extensions: Problems and Solutions," Universitext, Springer-Verlag, 2001.

[2] Brassard, G. and Bratley, P., "Fundamentals of Algorithmics," .

[3] Ceri, S. et al., "What you always wanted to know about Datalog (and never dared to ask)," IEEE Transactions on Knowledge and Data Engineering, (Volume: 1, Issue: 1), August 2002

[4] "Common Information Model (CIM)," DTMF, http://www.dmtf.org/standards/cim

[5] "Convex Optimization," https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf

[6] ETSI GS NFV 001 v1.1.1 (2013-10): "Network Function Virtualisation (NFV); Use Cases," http://www.etsi.org/deliver/etsi_gs/NFV/001_099/001/01.01.01_60/gs_NFV001v010101p.pdf

[7] ETSI GS NFV 002 v1.2.1 (2014-12): "Network Function Virtualisation (NFV); Architectural Framework," http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_nfv002v010201p.pdf

[8] ETSI GS NFV 003 V1.2.1 (2014-12): "ETSI NFV: Terminology for Main Concepts in NFV," http://www.etsi.org/deliver/etsi_gs/NFV/001_099/003/01.02.01_60/gs_NFV003v010201p.pdf

[9] ETSI GS NFV 004 v1.1.1 (2013-10): "Network Function Virtualisation (NFV); Virtualization Requirements," http://www.etsi.org/deliver/etsi_gs/NFV/001_099/004/01.01.01_60/gs_NFV004v010101p.pdf

[10] ETSI GS NFV-INF 001 v.1.1.1 (2015-01): "Network Function Virtualisation (NFV); Infrastructure Overview," http://www.etsi.org/deliver/etsi_gs/NFV-INF/001_099/001/01.01.01_60/gs_NFV-INF001v010101p.pdf

[11] ETSI NFV White Paper: "Network Functions Virtualisation, An Introduction, Benefits, Enablers, Challenges, & Call for Action," http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[12] Figueira, N. and Krishnan, R., "SDN Multi-Domain Orchestration and Control: Challenges and Innovative Future Directions," CNC VIII: Cloud and Multimedia Applications, IEEE International Conference on Computing (ICNC), February 2015

[13] Grit, L. et al., "Virtual Machine Hosting for Networked Clusters: Building the Foundations for "Autonomic" Orchestration," Virtualization Technology in Distributed Computing, 2006. VTDC 2006.

[14] Krishnan, R. et al., "Helping Telcos go Green and save OpEx via Policy", Talk and demo at the Vancouver OpenStack summit. Video Link: https://www.openstack.org/summit/vancouver-2015/summit-videos/presentation/helping-telcos-go-green-and-save-opex-via-policy

[15] Krishnan, R. et al., "NFVIaaS Architectural Framework for Policy Based Resource Placement and Scheduling," https://datatracker.ietf.org/doc/draft-krishnan-nfvrg-policy-based-rm-nfviaas/

[16] Lee, S. et al., "Resource Management in Service Chaining", https://datatracker.ietf.org/doc/draft-irtf-nfvrg-resource-management-service-chain/

[17] Moore, B., et al., "Information Model for Describing Network Device QoS Datapath Mechanisms," RFC 3670, January 2004

[18] Moore, B. et al., "Policy Core Information Model -- Version 1 Specification," RFC 3060, February 2001

[19] "OpenDaylight Group Based Policy," https://wiki.opendaylight.org/view/Project_Proposals:Group_Based_Policy_Plugin

[20] "OpenDaylight Network Intent Composition Project," https://wiki.opendaylight.org/index.php?title=Network_Intent_Composition:Main#Friday_8AM_Pacific_Time

[21] "OpenDaylight SDN Controller," http://www.opendaylight.org/

[22] "OpenStack," http://www.openstack.org/

[23] "OpenStack Celiometer," http://docs.openstack.org/developer/ceilometer/measurements.html

[24] "OpenStack Congress," https://wiki.openstack.org/wiki/Congress

[25] "OpenStack Neat," http://openstack-neat.org/

[26] "OpenStack Neutron," https://wiki.openstack.org/wiki/Neutron

[27] "Policy Framework Working Group," IETF, http://www.ietf.org/wg/concluded/policy.html

[28] Westerinen, A. et al., "Terminology for Policy-Based Management," RFC 3198, November 2001

Acknowledgements

The authors would like to thank the following individuals for valuable discussions on some of the topics addressed in this document: Uwe Michel, Klaus Martiny, Pedro Andres Aranda Gutierrez, Tim Hinrichs, Juergen Schoenwaelder, and Tina TSOU.

Authors' Addresses

Norival Figueira
Brocade Communications Systems, Inc.
nfigueir@Brocade.com

Ram (Ramki) Krishnan
Dell
Ramki_Krishnan@Dell.com

Diego R. Lopez
Telefonica I+D
diego.r.lopez@telefonica.com

Steven Wright
AT&T
sw3588@att.com

Dilip Krishnaswamy
IBM Research
dilikris@in.ibm.com

nfvrg
Internet-Draft
Intended status: Informational
Expires: May 3, 2017

R. Szabo, Ed.
Ericsson
S. Lee, Ed.
ETRI
N. Figueira
Brocade

October 30, 2016

Policy-Based Resource Management
draft-irtf-nfvrg-policy-based-resource-management-02

Abstract

abstract to be defined

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 3, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 2
1.1. Scope 3
2. Terminology 3
3. Definitions 3
4. Requirements 4
5. Architecture Considerations 4
5.1. MANO Architecture 5
5.2. Policies in the MANO Architecture 8
5.3. Global vs Local Policies 9
5.4. Hierarchical Policy Framework 10
5.4.1. Mapping to Hierarchical Resource Orchestration . . . 12
5.5. Policy Pub/Sub Bus 13
5.5.1. Pub/sub bus in the hierarchical framework 15
5.6. Policy Intent Statement versus Subsystem Actions and Configurations 17
5.7. Static vs Dynamic vs Autonomic Policies 17
5.8. Policy Conflicts and Resolution 17
5.9. Soft vs Hard Policy Constraints 17
5.10. Operational Policies for Resource management 17
5.10.1. Operational Policies at NFVO 19
5.10.2. Operational Policies at VIM/WIM 19
6. Policy-Based Resource Management Examples 20
6.1. Policy-Based Multipoint Ethernet Service 20
6.2. Policy-Based NFV Placement 20
6.3. Policy-Based VNF-FG Management 20
6.4. Policy-Based Fault Management 22
7. Implementation Examples 28
8. Gaps and Open Questions 28
9. Conclusions 28
9.1. Relation to other IETF/IRTF activities 28
10. Acknowledgements 28
11. Contributors 28
12. IANA Considerations 29
13. Security Considerations 29
14. References 29
14.1. Normative References 29
14.2. Informative References 29
Authors' Addresses 32

1. Introduction

NFV "Point of Presence" (PoP) will be likely constrained in compute and storage capacity. Since practically all NFV PoPs are foreseen to be distributed, inter-datacenter network capacity is also a constraint. Additionally, energy is also a constraint, both as a general concern for NFV operators, and in particular for specific-

purpose NFV PoPs such as those in mobile base stations. This draft focuses on the optimized resource management and workload distribution based on policy to address such contraints.

1.1. Scope

For the first version of the draft, only the research group currently adopted drafts (i.e., [I-D.norival-nfvrg-nfv-policy-arch], [I-D.irtf-nfvrg-resource-management-service-chain], and [I-D.unify-nfvrg-recursive-programming]) are considered as inputs to this document. The initial goal is to summarize these inputs and to assess gaps and open questions.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Definitions

This document uses the terms of [ETSI-NFV-TERM]:

- MANO - Management and Orchestration: Describes the architecture framework to manage NFVI and orchestrate the allocation of resources needed by the NSs and VNFs.

- NF - Network Functions: A functional building block within a network infrastructure, which has well-defined external interfaces and a well-defined functional behavior.

- NFV Framework: The totality of all entities, reference points, information models and other constructs defined by the specifications published by the ETSI ISG NFV.

- NFVI - NFV Infrastructure: The NFV-Infrastructure is the totality of all hardware and software components which build up the environment in which VNFs are deployed.

- NFVI-PoP: A location or point of presence that hosts NFV infrastructure

- NFVO - Network Function Virtualization Orchestrator: The NFV Orchestrator is in charge of the network wide orchestration and management of NFV (infrastructure and software) resources, and realizing NFV service topology on the NFVI.

- NS - Network service: A composition of network functions and defined by its functional and behavioural specification.

- VNF - Virtualized Network Function: An implementation of an NF that can be deployed on a Network Function Virtualization Infrastructure (NFVI).

- VNF-FG - VNF Forwarding Graph: A NF forwarding graph where at least one node is a VNF.

Additionally, we use the following terms:

- NFP - Network Forwarding Path: The sequence of hardware/software switching ports and operations in the NFV network infrastructure as configured by management and orchestration that implements a logical VNF forwarding graph "link" connecting VNF "node" logical interfaces.

- Virtual Link: A set of connection points along with the connectivity relationship between them and any associated target performance metrics (e.g. bandwidth, latency, QoS). The Virtual Link can interconnect two or more entities (VNF components, VNFs, or PNFs).

- Scaling: Ability to dynamically extend/reduce resources granted to the Virtual Network function (VNF) as needed.

- NFVIaaS: NFV infrastructure as a service to other SP customers.

- SDN: Software Defined Networking.

- BSS: Business Support Systems

- OSS: Operation Support Systems

- DC: Data Center

- VM: Virtual machine

4. Requirements

tbd

5. Architecture Considerations

5.1. MANO Architecture

According to the ETSI MANO framework [ETSI-NFV-MANO], an NFVO is split into two functions (see Figure 1):

- The orchestration of NFVI resources across multiple VIMs, fulfilling the Resource Orchestration functions. The NFVO uses the Resource Orchestration functionality to provide services that support accessing NFVI resources in an abstracted manner independently of any VIMs, as well as governance of VNF instances sharing resources of the NFVI infrastructure

- The lifecycle management of Network Services, fulfilling the network Service Orchestration functions.

Similarly, a VIM is split into two functions (see Figure 1):

- Orchestrating the allocation/upgrade/release/reclamation of NFVI resources (including the optimization of such resources usage), and

- managing the association of the virtualised resources to the physical compute, storage, networking resources.

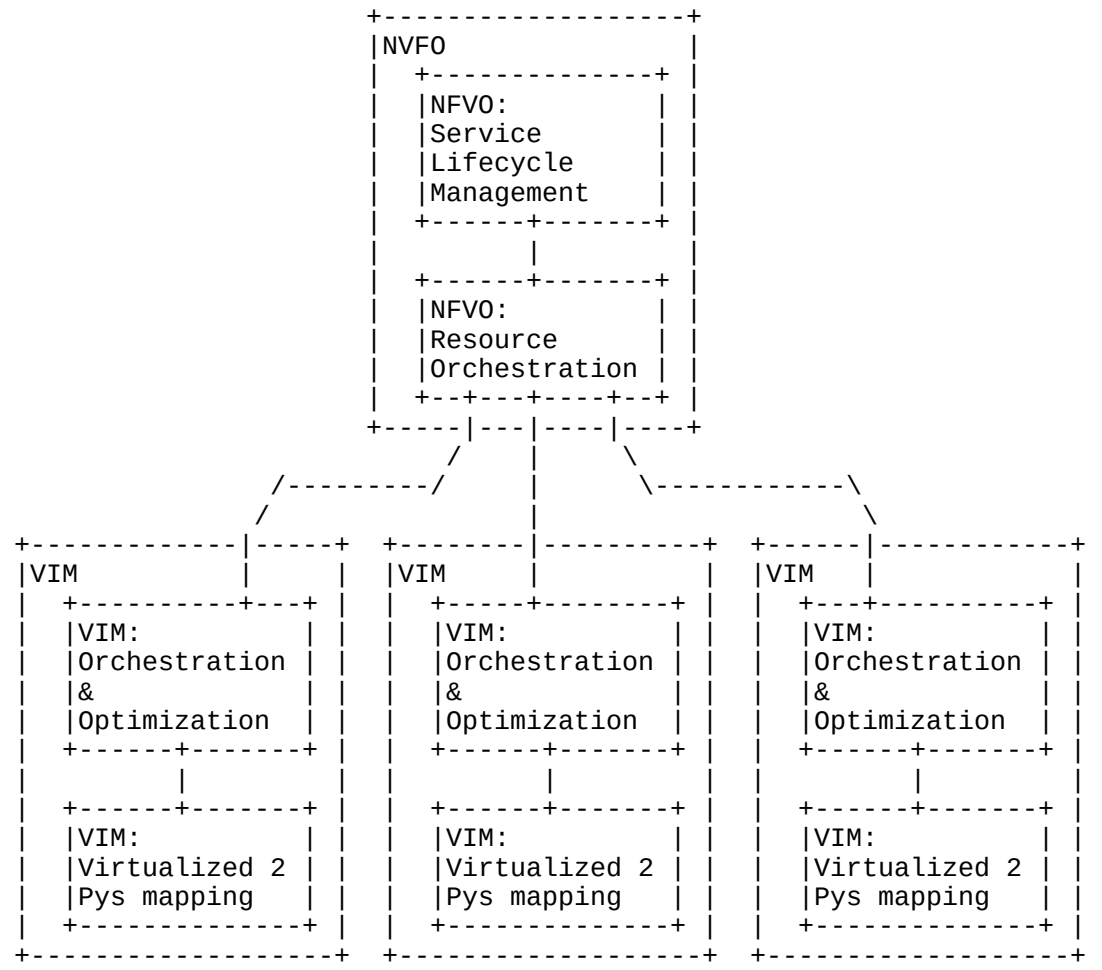


Figure 1: Functional decomposition of the NFVO and the VIM according to the ETSI MANO

In Figure 2 we show various policies mapped to the MANO architecture (see Section 5.2 for more dicussions on policies in the MANO architeture):

- Tenant Policies: Tenant policies exist whenever a domain offers a virtualization service to more than one consumer. User tenants may exists at the northbound of the NFVO. Additionally, if a VIM exposes resource services to more than one NFVO, then each NFVO may appear as a tenant (virtualization consumer) at the northbound of the VIM.

- Wherever virtualization services are produced or consumed corresponding export and import policies may exist. Export policies govern the details of resources, capabilities, costs, etc. exposed to consumers. In turn, consumers (tenants) apply import policies to filter, tweak, annotate resources and services received from their southbound domains. An entity may at the same time consume and produce virtualization services hence apply both import and export policies.

- Operational policies support the business logic realized by the domain’s ownership. They are often associated with Operations or Business Support Systems (OSS or BSS) and frequently determine operational objectives like energy optimization, utilization targets, offered services, charging models, etc. Operational policies may be split according to different control plane layers, for example, i) lifecycle and ii) resource management layers within the NFVO.

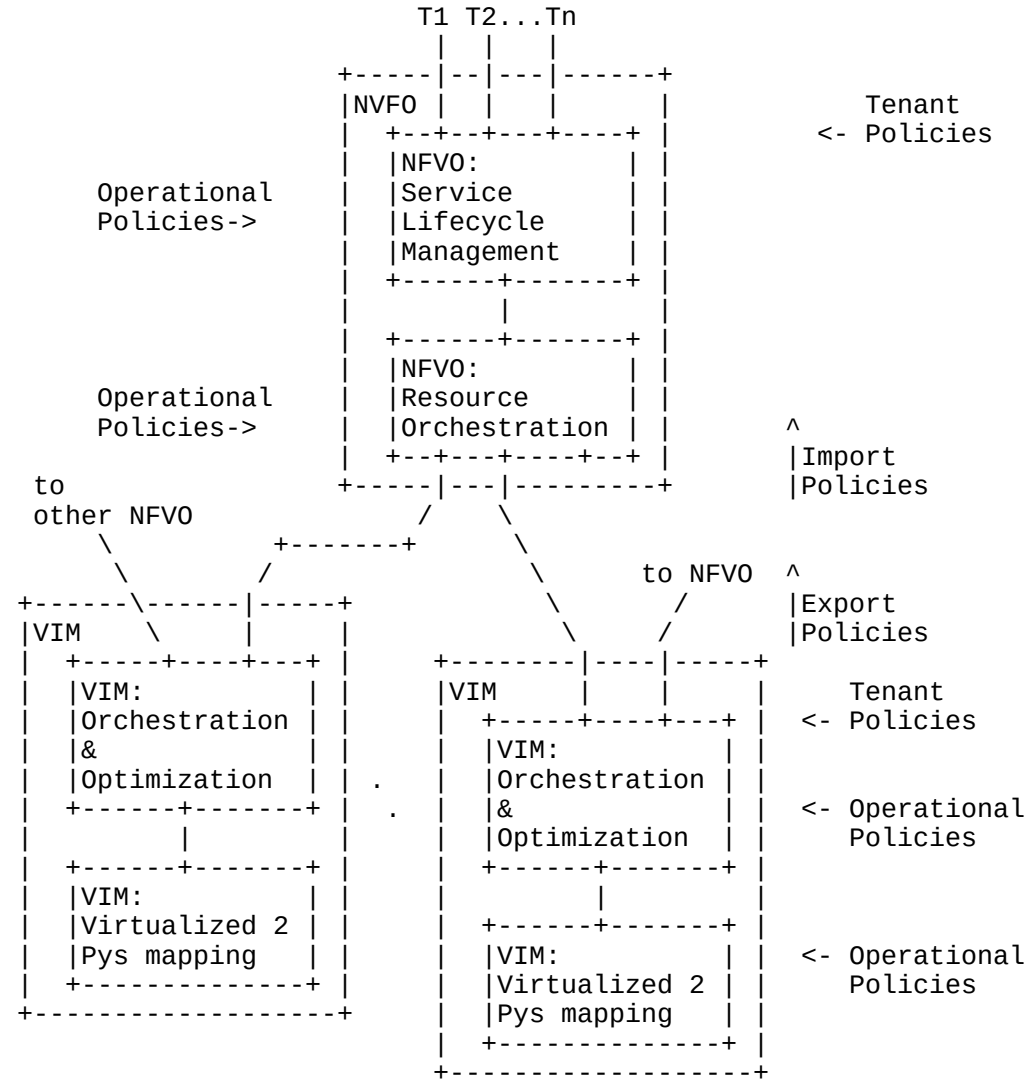


Figure 2: Policies within the MANO framework

5.2. Policies in the MANO Architecture

The current industry work in the area of policy for NFV is mostly considered in the framework of general cloud services, and typically focused on individual subsystems and addressing very specific use cases or environments. For example, [ETSI-NFV-WHITE-PAPER] addresses network subsystem policy for network virtualization, [ODL-GB-POLICY] and [ODL-NIC-PROJECT] are open source projects in the area of network

policy as part of the OpenDaylight [ODL-SDN-CONTROLLER] software defined networking (SDN) controller framework, [RFC3060] specifies an information model for network policy, [VM-HOSTING-NET-CLUSTER] focuses on placement and migration policies for distributed virtual computing, [OPENSTACK-CONGRESS] is an open source project proposal in OpenStack [OPENSTACK] to address policy for general cloud environments.

A policy framework applicable to the MANO architure must consider NFV services from the perspective of overall orchestration requirements for services involving multiple subsystems (e.g., Figure 1 and Figure 2).

While this document discusses policy atributes as applicable to the MANO architecture, the general topic of policy has already been intensively studied and documented on numerous publications over the past 10 to 15 years (see [RFC3060], [POLICY-FRAMEWORK-WG], [RFC3670], [RFC3198], and [CERI-DATALOG] to name a few). This document's purpose is to discuss and document a policy framework applicable to the MANO architecture using known policy concepts and theories to address the unique requirements of NFV services including multiple PoPs and networks forming hierarchical domain architectures [SDN-MULTI-DOMAIN].

With the above goals, this document analyses "global versus local policies" (Section 5.3), a "hierarchical policy framework" (Section 5.4) to address the demanding and growing requirements of NFV environments, a "policy pub/sub bus in the hierarchical framework" (Section 5.5), "policy intent versus subsystem actions" (Section 5.6), "static versus dynamic versus autonomic policies" (Section 5.7), "policy conflict detection and resolution" (Section 5.8), and "soft versus hard policy constraints" (Section 5.9), which can be relevant to resource management in service chains [RESOURCE-MGMT-SERVICE-CHAIN].

5.3. Global vs Local Policies

Some policies may be subsystem specific in scope, while others may have broader scope and interact with multiple subsystems. For example, a policy constraining certain customer types (or specific customers) to only use certain server types for VNF or Virtual Machine (VM) deployment would be within the scope of the compute subsystem. A policy dictating that a given customer type (or specific customers) must be given "platinum treatment" could have different implications on different subsystems. As shown in Figure 8, that "platinum treatment" could be translated to servers of a given performance specification in a compute subsystem and storage of a given performance specification in a storage subsystem.

Policies with broader scope, or global policies, would be defined outside affected subsystems and enforced by a global policy engine (Figure 3), while subsystem-specific policies or local policies, would be defined and enforced at the local policy engines of the respective subsystems.

Examples of sub-system policies can include thresholds for utilization of sub-system resources, affinity/anti-affinity constraints with regard to utilization or mapping of sub-system resources for specific tasks, network services, or workloads, or monitoring constraints regarding under-utilization or over-utilization of sub-system resources.

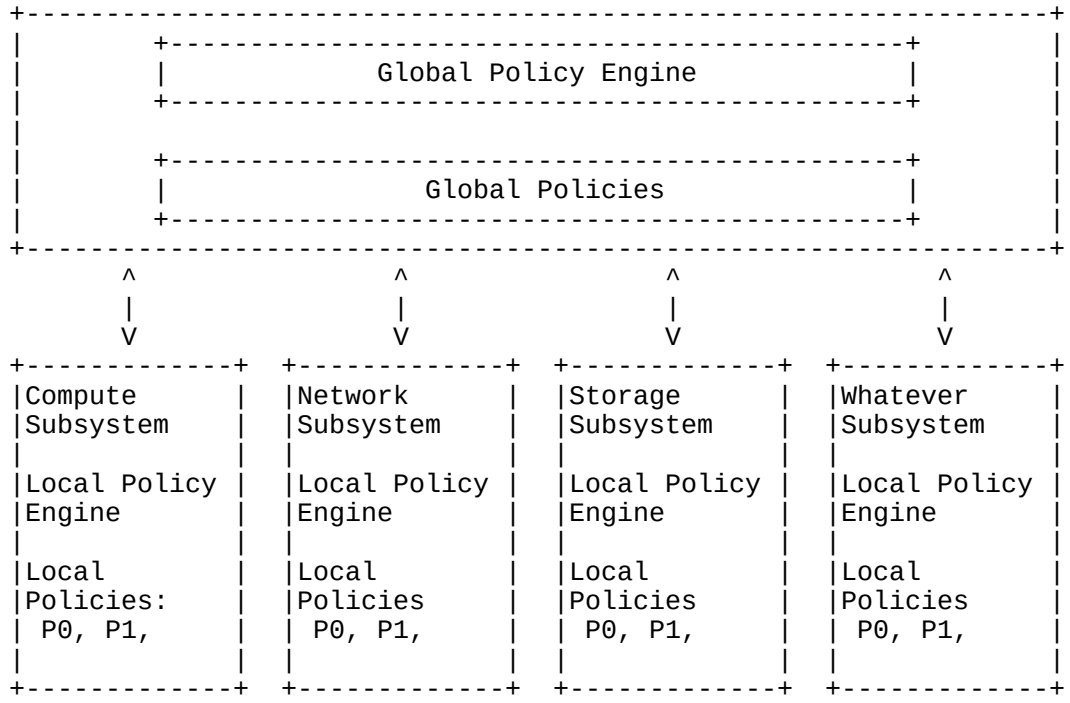


Figure 3: Global versus Local Policy Engines

5.4. Hierarchical Policy Framework

So far, we have referenced compute, network, and storage as subsystems examples. However, the following subsystems may also support policy engines and subsystem specific policies:

- SDN Controllers, e.g., OpenDaylight [ODL-SDN-CONTROLLER].

- OpenStack [OPENSTACK] components such as, Neutron, Cinder, Nova, and etc.
- Directories, e.g., LDAP, ActiveDirectory, and etc.
- Applications in general, e.g., standalone or on top of OpenDaylight or OpenStack.
- Physical and virtual network elements, e.g., routers, firewalls, application delivery controllers (ADCs), and etc.
- Energy subsystems, e.g., OpenStack Neat [OPENSTACK-NEAT].

Therefore, a policy framework may involve a multitude of subsystems. Subsystems may include other lower level subsystems, e.g., Neutron [OPENSTACK-NEUTRON] would be a lower level subsystem in the OpenStack subsystem. In other words, the policy framework is hierarchical in nature, where the policy engine of a subsystem may be viewed as a higher level policy engine by lower level subsystems. In fact, the global policy engine in Figure 3 could be the policy engine of a Data Center subsystem and multiple Data Center subsystems could be grouped in a region containing a region global policy engine. In addition, one could define regions inside regions, hierarchically, as shown in Figure 4.

Metro and wide-area network (WAN) used to interconnect data centers would also be independent subsystems with their own policy engines.

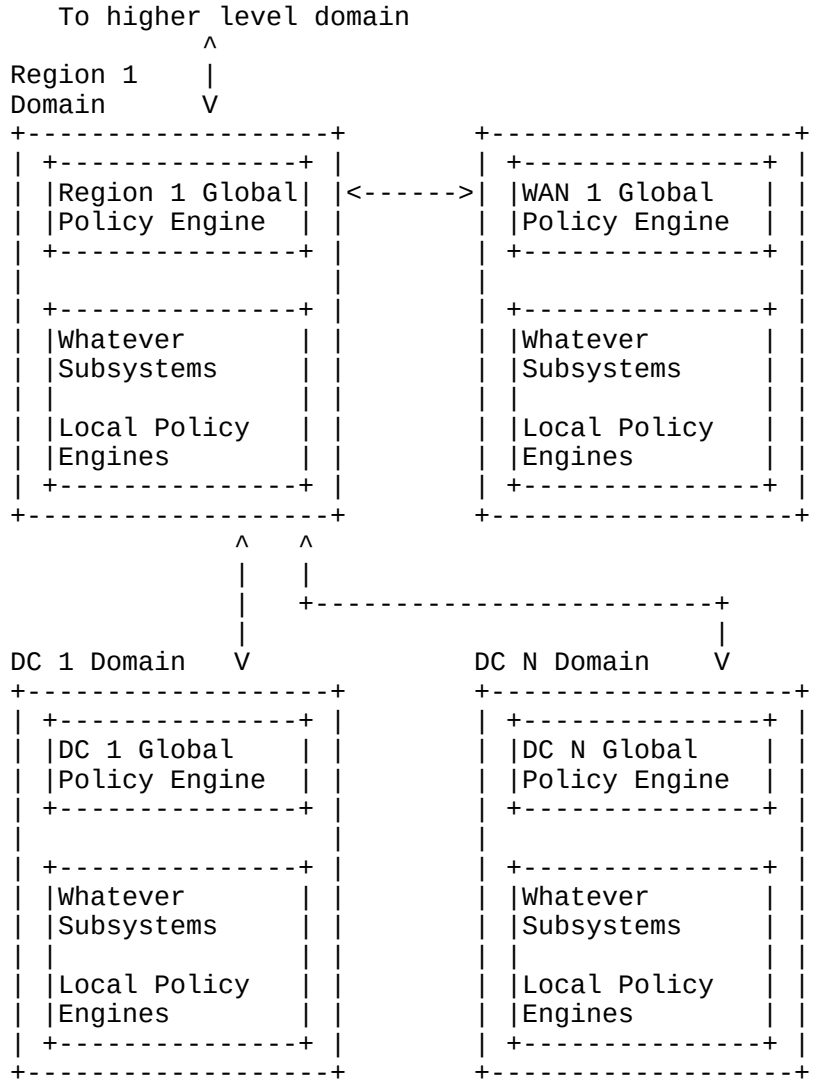


Figure 4: A Hierarchical Policy Framework

5.4.1. Mapping to Hierarchical Resource Orchestration

If the MANO framework is extended to multi layer hierarchies [I-D.unify-nfvrg-recursive-programming], then a potential mapping of the hierarchical policies to the MANO architecture is shown in Figure 5

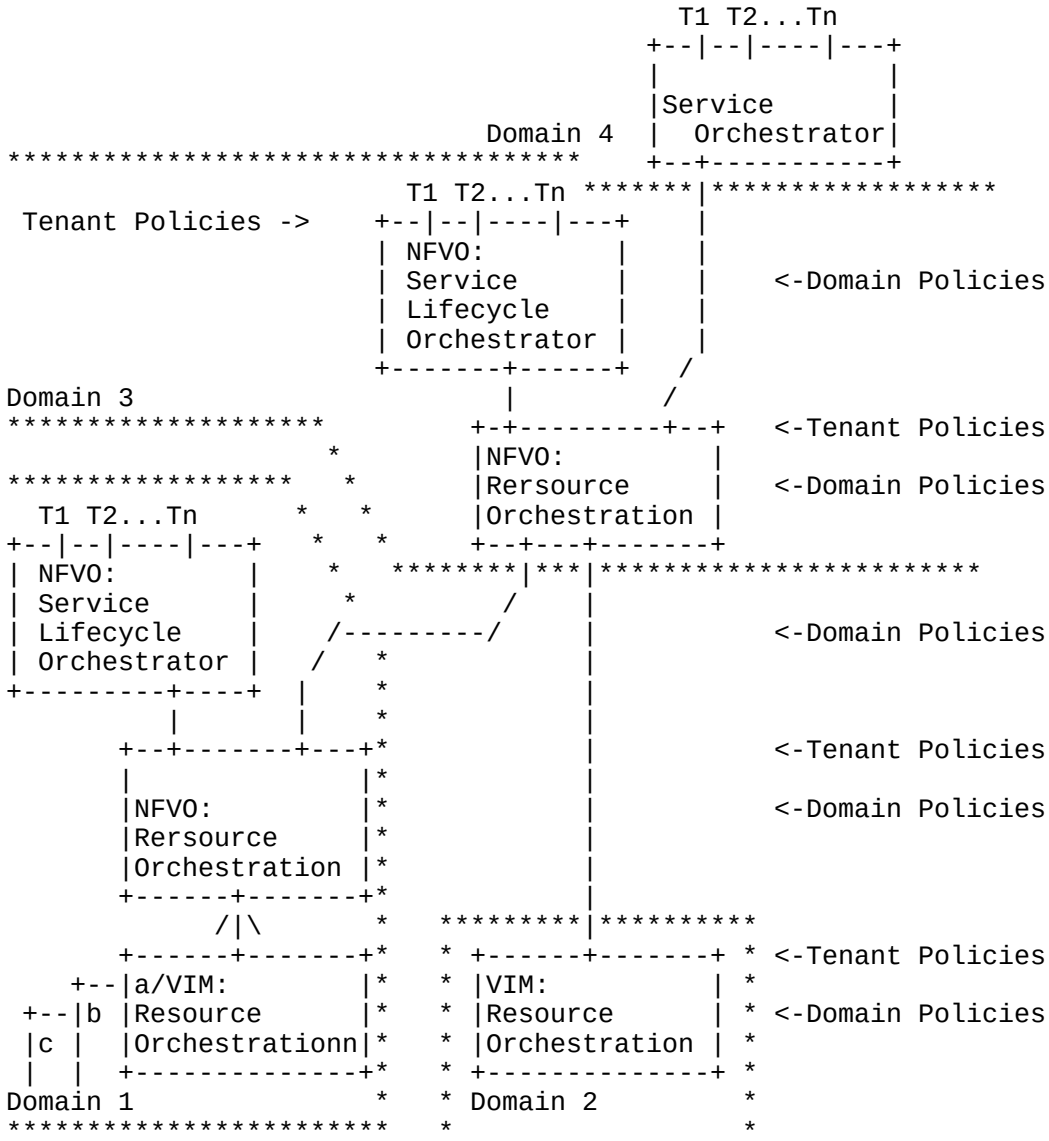


Figure 5: Policies in a Hierarchical Orchestration View

5.5. Policy Pub/Sub Bus

In [I-D.irtf-nfvrg-nfv-policy-arch] the authors argued for the need of policy subsystems to subscribe to policy updates at a higher policy level. A policy publication/subscription (pub/sub) bus would be required as shown in Figure 6.

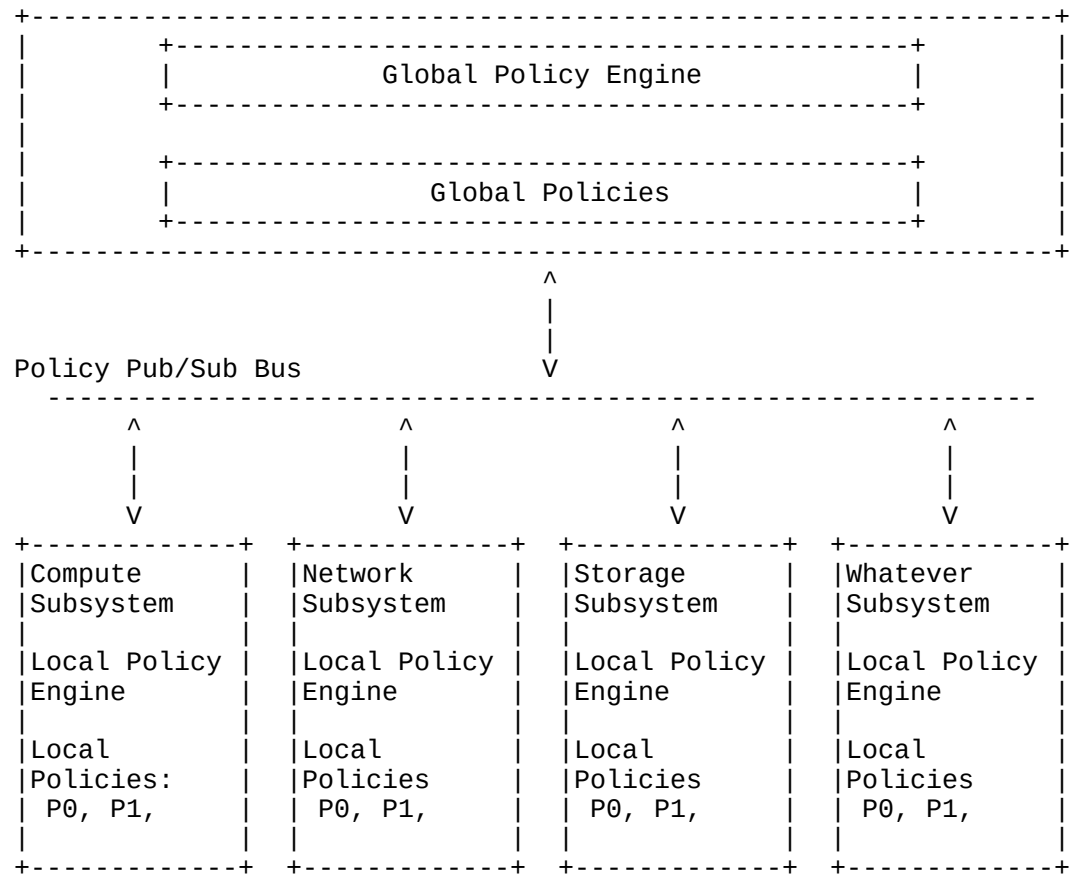


Figure 6: A Policy Pub/Sub Bus

A higher tier policy engine would communicate policies to lower tier policy engines using a policy pub/sub bus. Conversely, lower tier policy engines would communicate their configured policies and services to the higher tier policy engine using the same policy pub/sub bus. Such communications require each policy pub/sub bus to have a pre-defined/pre-configured policy "name space". For example, a pub/sub bus could define services using the name space "Platinum", "Gold", and "Silver". A policy could then be communicated over that pub/sub bus specifying a Silver service requirement.

In a hierarchical policy framework, a policy engine may use more than one policy pub/sub bus, e.g., a policy pub/sub bus named "H" to communicate with a higher tier policy engine and a policy pub/sub bus named "L" to communicate with lower tier policy engines. As the name spaces of policy pub/sub buses H and L may be different, the policy

engine would translate policies defined using the policy pub/sub bus H name space to policies defined using the policy pub/sub bus L name space, and vice-versa.

5.5.1. Pub/sub bus in the hierarchical framework

Figure 7 shows the Pub/sub bus in the hierarchical MANO framework. Policy communications would employ a policy pub/sub bus between the subsystems' policy engines in the policy hierarchy (see Section 5.4). The global NFVO subsystem should have visibility into the policies defined locally at each PoP to be able to detect any potential global policy conflicts, e.g., a local PoP administrator could add a local policy that violates or conflicts with a global policy. In addition, the global NFVO subsystem would benefit from being able to import the currently configured services at each PoP. The global NFVO would use such information to monitor global policy conformance and also to facilitate detection of policy violations when new global policies are created, e.g., a global level administrator is about to add a new global policy that, if committed, would make certain already configured services a violation of the policy. The publication of subsystem service tables for consumption by a global policy engine is a concept used in the Congress [OPENSTACK-CONGRESS] OpenStack [OPENSTACK] project.

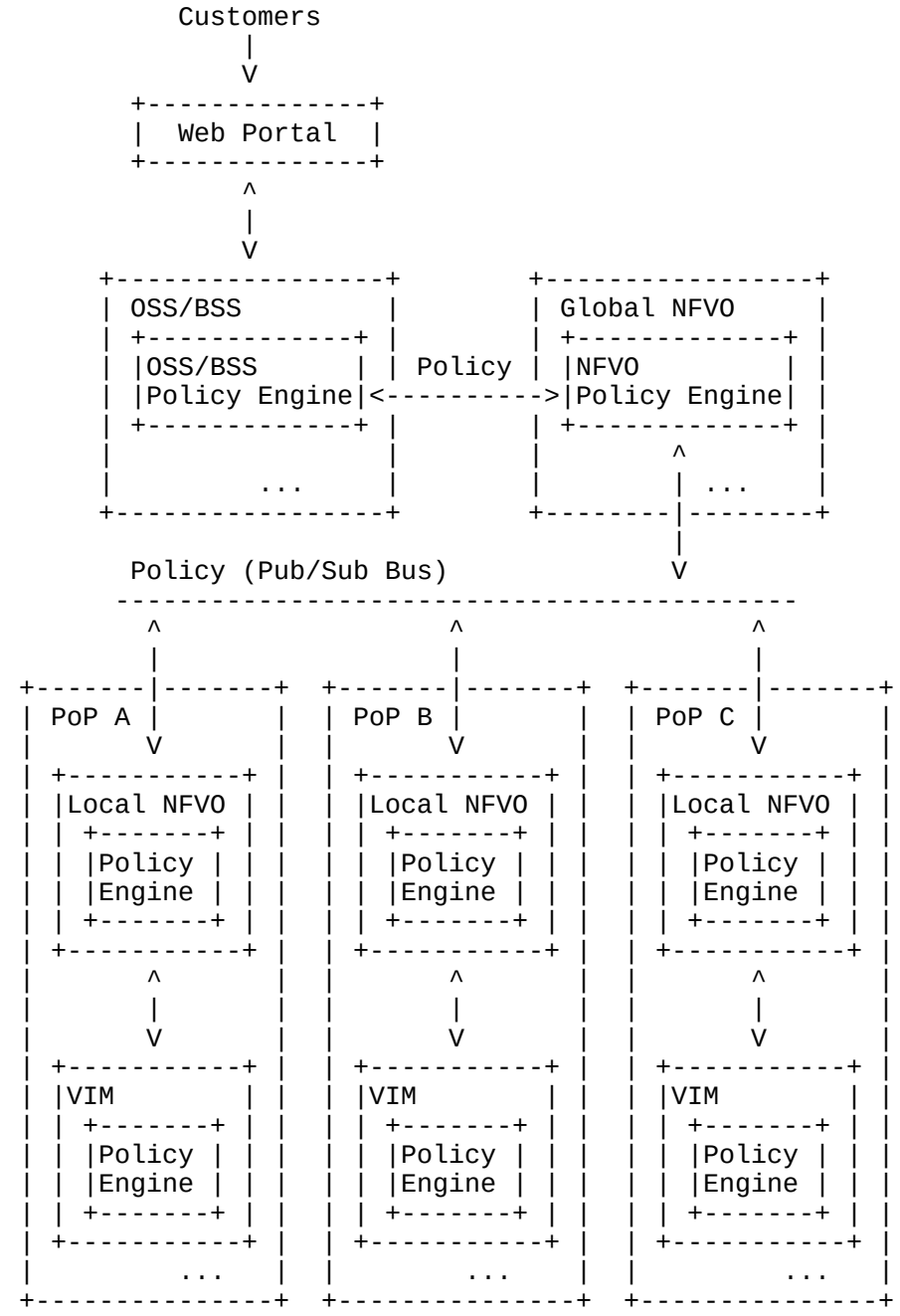


Figure 7: Pub/sub bus in the hierarchical MANO framework

5.6. Policy Intent Statement versus Subsystem Actions and Configurations

Content to be merged

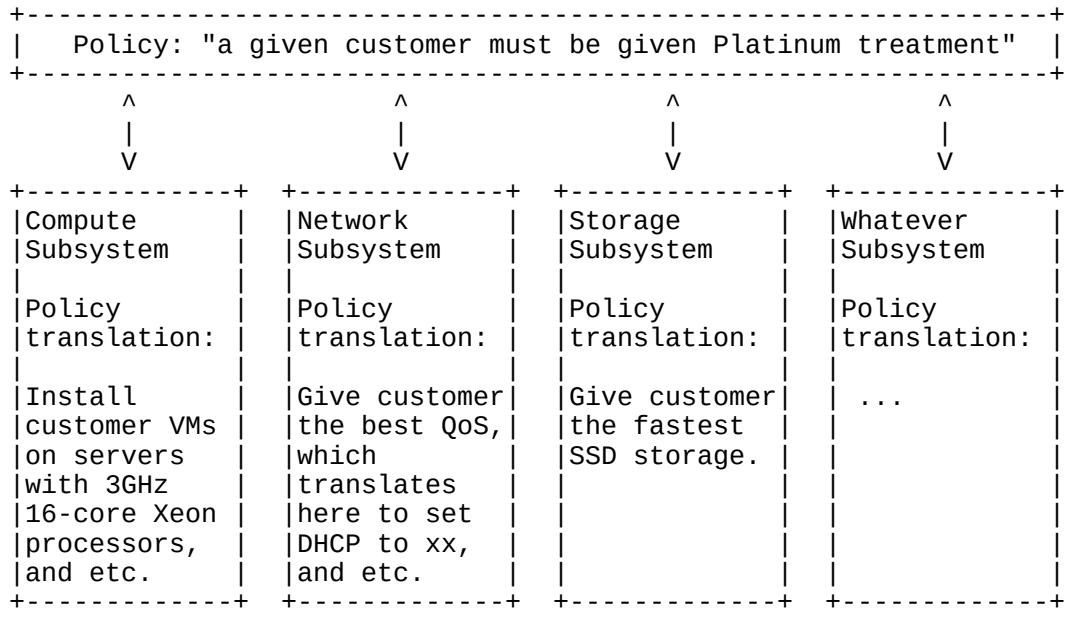


Figure 8: Example of Subsystem Translations of Policy Actions

5.7. Static vs Dynamic vs Autonomic Policies

Content to be merged

5.8. Policy Conflicts and Resolution

Content to be merged

5.9. Soft vs Hard Policy Constraints

Content to be merged

5.10. Operational Policies for Resource management

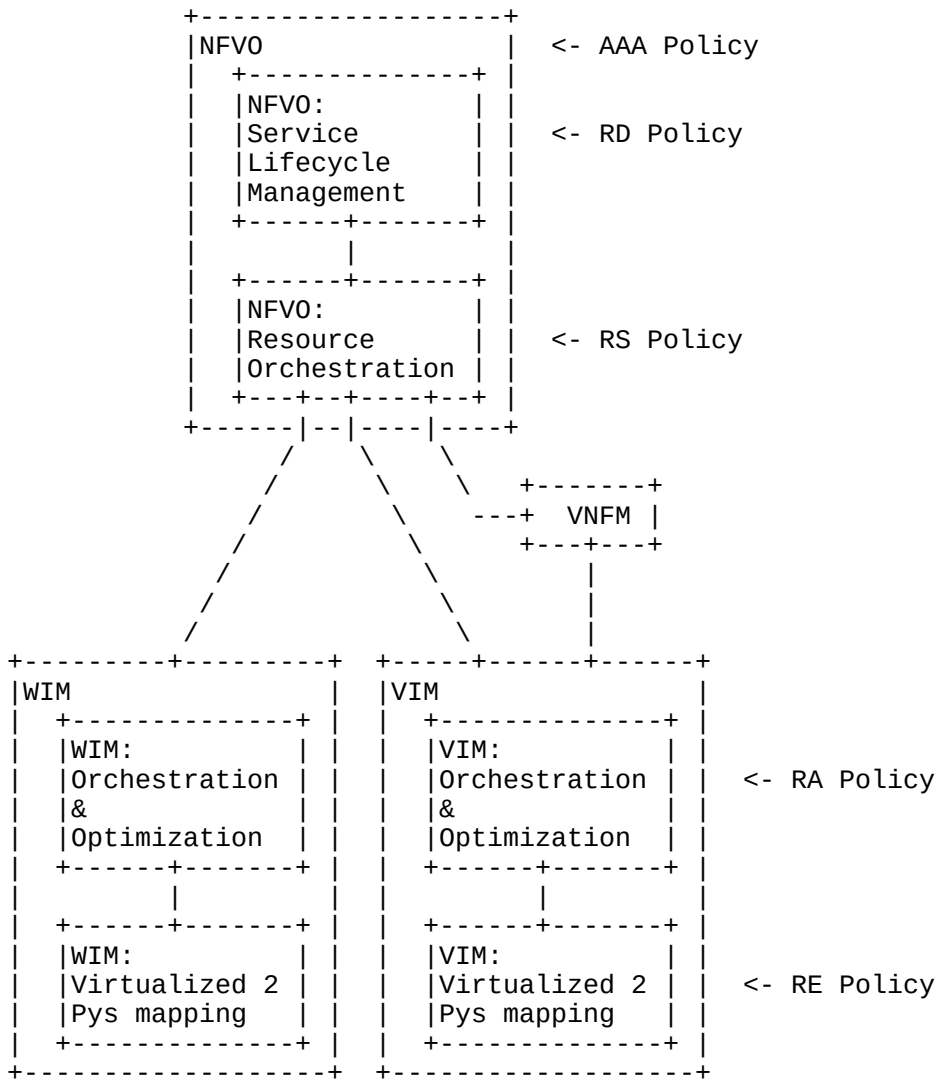


Figure 9: Operational policies for resource management

The use of NFVI resources for multiple network services can be optimized in various objectives as defined in the operational policies (as described in Section 5.2).

The operational policies can be split to different layers of NFVO and VIM/WIM and they include 1) resource scheduling (RS) policy, resource adaptation (RD) policy and authentication, authorization, accounting (AAA) policy at NFVO, and 2) resource allocation (RA) policy and

resource embedding (RE) policy at VIM/WIM. They can be mapped to the MANO architecture as shown in Figure 9.

5.10.1. Operational Policies at NFVO

During NS/VNF lifecycles, states of NFVI/WAN resources or the performance of VNF and VL instances may vary in time (e.g., the performance degradation due to incorrect placement or incorrect forwarding action). Another concern for such dynamic changes is fail-over as a fundamental consideration, i.e., physical resources or virtualized resources in NFVI may fail during network services. These dynamic changes significantly could affect the overall performance for NS. Therefore, such dynamic changes triggered during NS/VNF lifecycles should be coped with for guaranteeing the NS performance and the optimized resource usage. Figure 9 shows that NFVO needs to enforce resource adaptation (RD) policy as an operational policy at NFVO. RD policy supports how NFVO adapts the allocated NFVI/WAN resources (e.g., VM migration, scaling) by dealing with triggered variations. RD policy engine can detect the changes from measurement and diagnosis from VNFM and/or VIM/WIM.

Figure 9 also shows that NFVO needs to enforce resource scheduling (RS) policy. RS policy determines the locations of VNF and VL instances that constitute NS across multiple PoPs and WANs while optimally allocating NFVI and WAN resources to the instances.

In particular, RD and RA policies would consider a business model from OSS/BSS which specifies operational (or business) objectives (e.g., overall energy consumption and NFVI resource utilization) within its domain and with taking account of (on-boarded) network service descriptor (NSD) as an NS policy including the virtualization aspects of application feature, QoS parameters, affinity, anti-affinity rules, and so on.

On the one hand, for the user authorization, authentication, authorization, accounting (AAA) policy may be needed. Authentication policy provides a way of identifying a user while the authorization policy determines whether the user has the authority for virtualized resources (i.e., NFVI/WAN resources) to receive the network service or not. Accounting policy measures the resources the user consumes during the network service. This can include the amount of system time/data, and so on.

5.10.2. Operational Policies at VIM/WIM

As shown in Figure 9, RA policy supports how each subsystem (e.g., compute, storage subsystem) in NFVI is allocated depending on the placement information from NFVO to further optimize the resource

usage. Moreover, the assigned NFVI resources are embedded (or allocated) to physical resources in VIM/WIM depending on states and usage of resources by means of resource embedding (RE) policy as shown in Figure 9. In other words, RE policy determines and coordinates how the allocated virtual resources are mapped to physical resources. For example, RE policy may be updated when some physical resources are failed or a virtualization technique is changed.

6. Policy-Based Resource Management Examples

6.1. Policy-Based Multipoint Ethernet Service

Content to be merged

6.2. Policy-Based NFV Placement

Content to be merged

6.3. Policy-Based VNF-FG Management

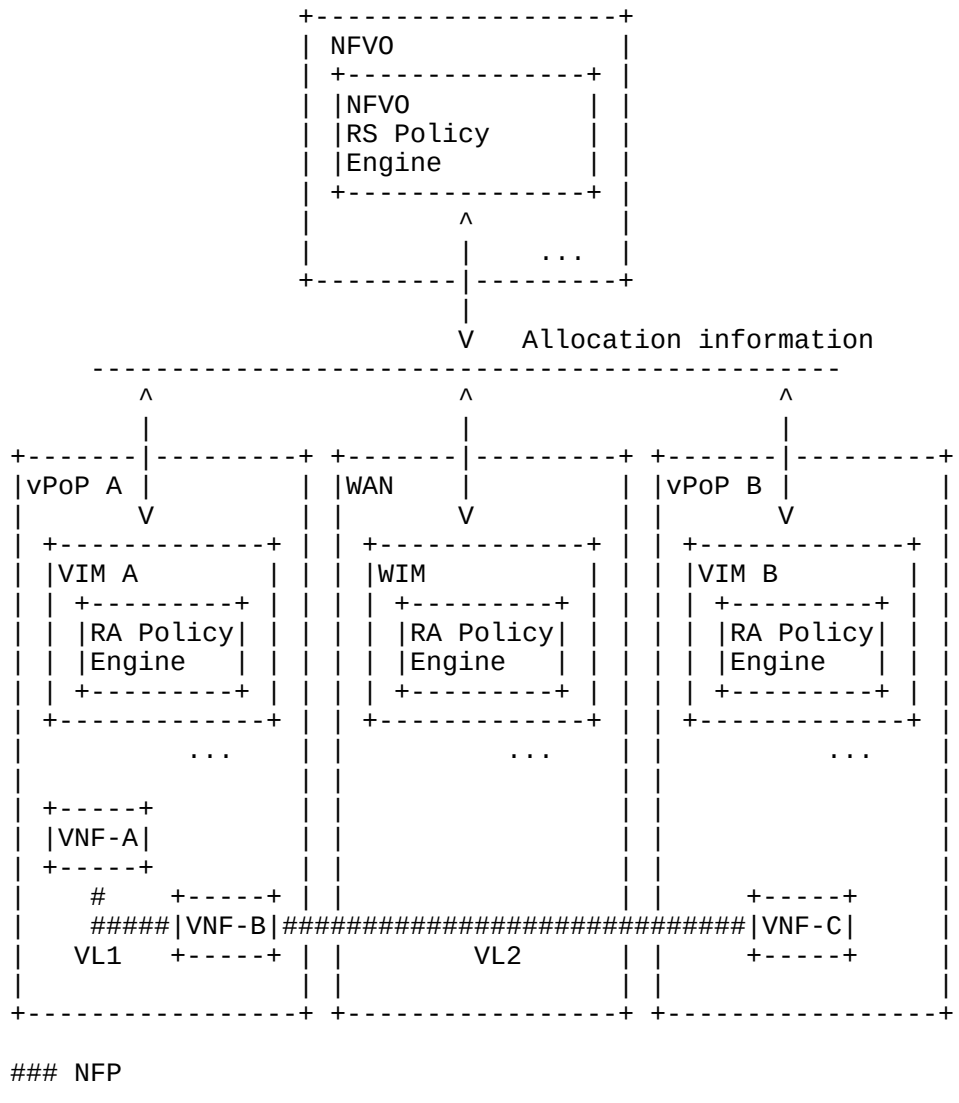


Figure 10: Policy-based VNF-FG Management

Another subsystem example for the policy framework is VNF-FG. When VNF-FGs of end-to-end network services are realized, NFVI resources across multiple NFVI-PoPs and WAN resources that connect among them should be allocated to the VNF-FGs. It depends on the target KPIs of individual VNF and VL instances that constitute VNF-FGs. In particular, in case of VNF-FG, chained performances and capabilities

of VNF and VL instances need to be considered together with on VL instances the inter-connectivity between different NFVI-PoPs. For example, if one of the VNF instances or VL instances along the VNF-FG gets overloaded, the end-to-end network service may also get affected. Therefore, while features of such VNF-FG are carefully considered, proper operational policies for resource management (see Section 5.10) are required.

As shown in Figure 10, consider a scenario where a user requests a VNF-FG composed of "VNF A-VL 1-VNF B-VL 2-VNF C". For the VNF-FG, an RA policy is enforced in which it is designed to avoid over-utilization of PoP A and to reduce latency on VL 1. Therefore, NFVO places VNF A, VNF B, and VL 1 on PoP A by consuming its computing and network resources to achieve low latency. On the other hand, VL 2 and VNF C is allocated to the resources of WAN and PoP B, respectively to avoid over-utilization of PoP A.

On the one hand, dynamic changes such as a VNF failure significantly affect on the overall performance of VNF-FG since VNF-FG is a chain of VNF and VL instances. Thus, such dynamic changes should be coped with by RD policy for guaranteeing the VNF-FG performance and the optimized resource usage. A fault management for VNF-FG based on policy example is shown in Section 6.4.

6.4. Policy-Based Fault Management

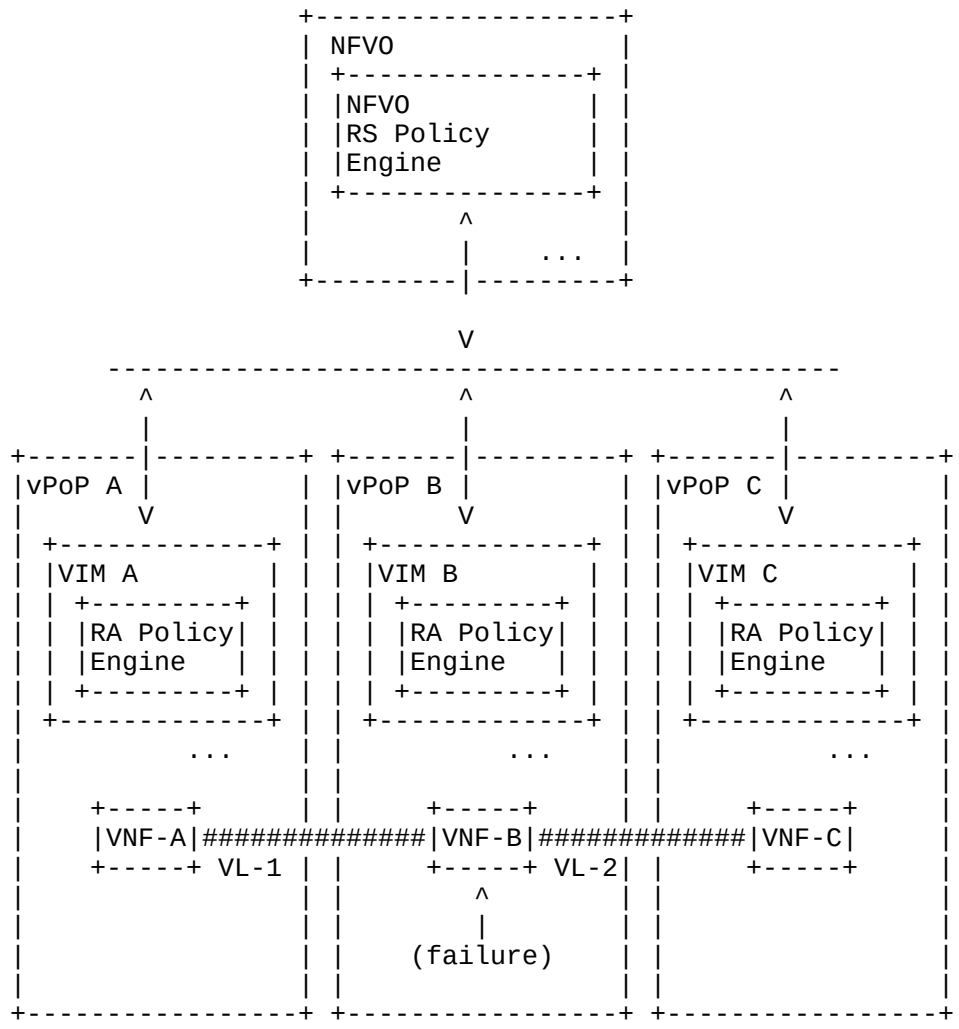


Figure 11: Failure Scenario for VNF-FG

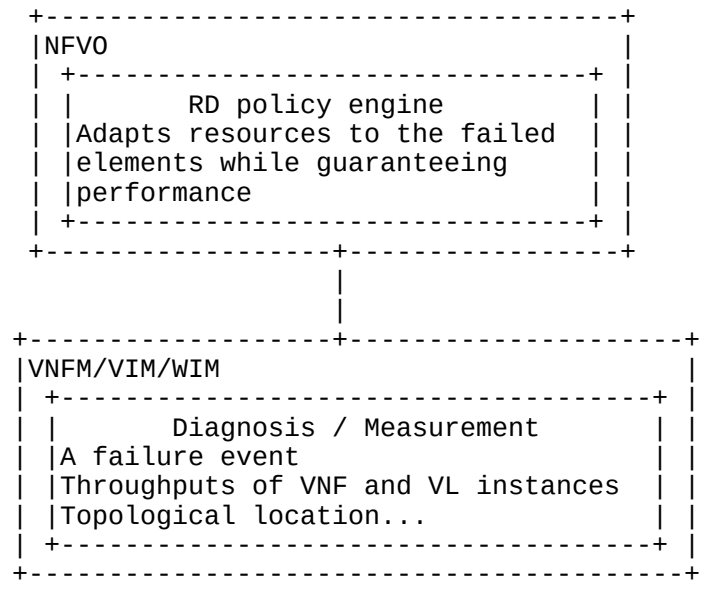


Figure 12: Architecture for policy-based fault management

As shown in Figure 11, consider a scenario that a VM related to VNF-B (i.e., a VNF-B instance) is failed in the given VNF-FG composed VNF-A, VNF-B, VNF-C in order. Note that the NFVI and WAN resources are already allocated to the instances by RS policy. For service continuity, failure of the VNF-B instance needs to be detected based on diagnosis function in VIM/VNFM and the failed one needs to be replaced with a new instance or to be assigned to the existing instance which is available. The diagnosis and measurement function may collect current performance measures and location for instances as well as such a failure event.

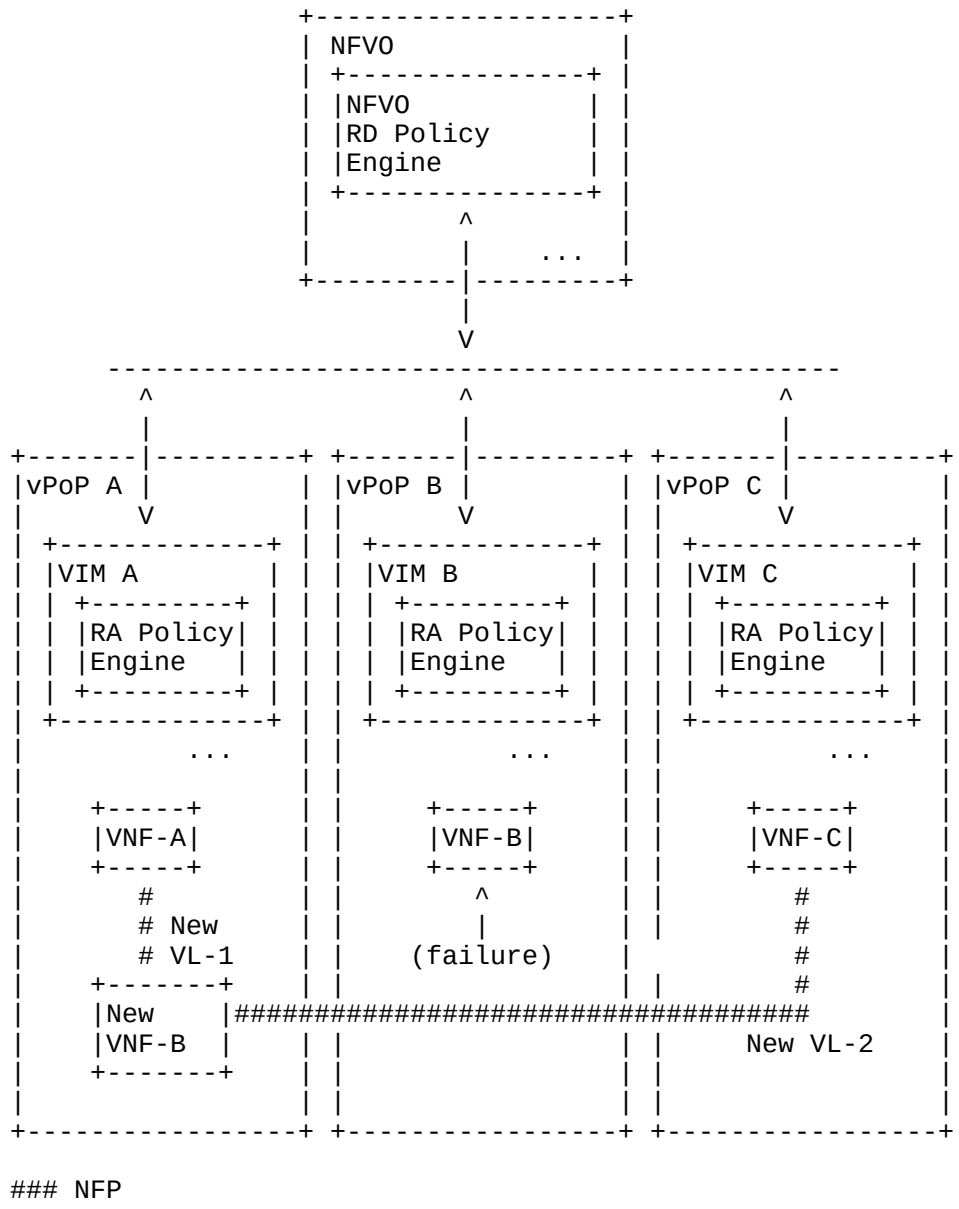


Figure 13: Re-instantiation for VNF-FG

In the first case where a VNF instantiation is needed, a new VNF instantiation is determined by the RD policy engine in NFVO. For

example, NFVO may avoid replacement of VNF B on NFVI-PoP B owing to high possibility of failure. Therefore, NFVO could instantiate VNF B on NFVI-PoP A or NFVI-PoP C with the setup of new connection points (CPs) while guaranteeing performance as shown in Figure 13.

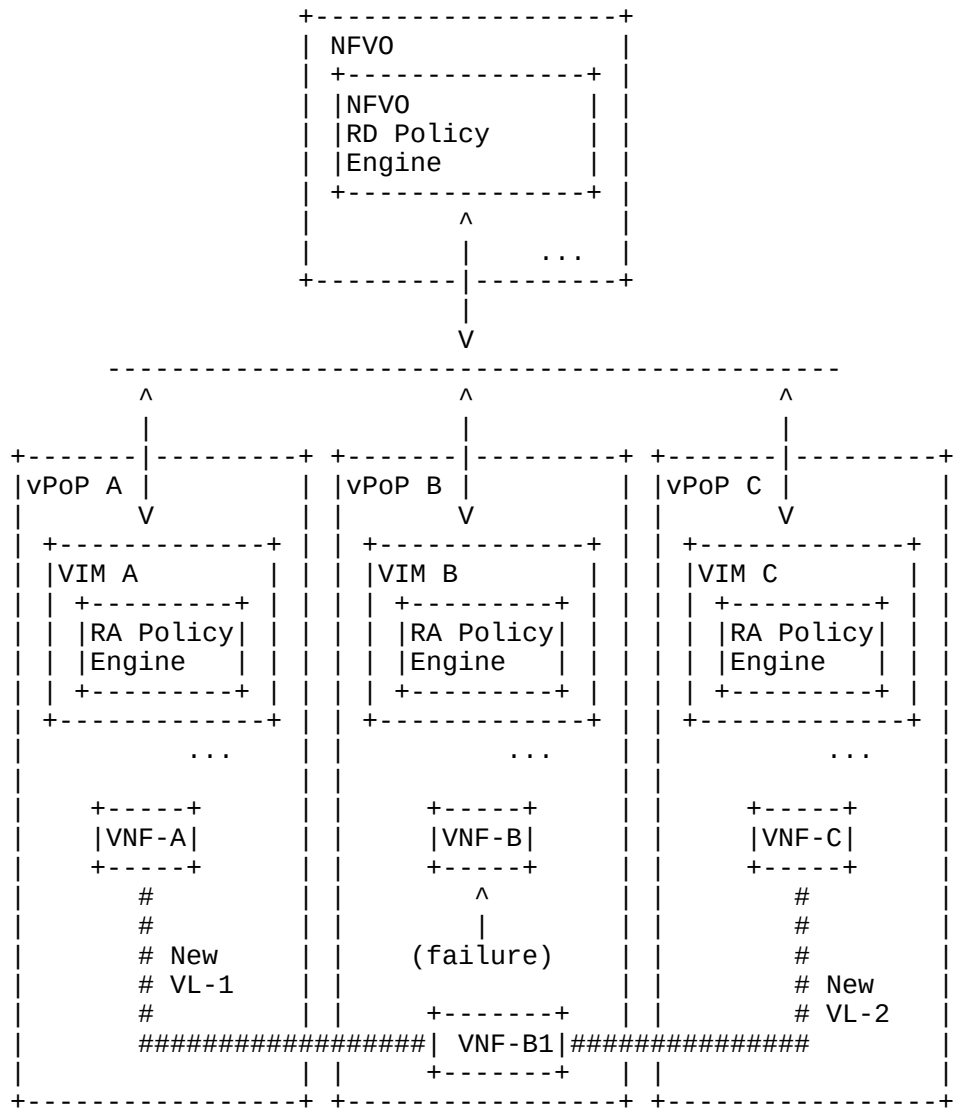


Figure 14: No Re-instantiation for VNF-FG

In the second case where no VNF instantiation is needed since a redundant VNF exists, the available VNF-B instance can used by the VNF-FG. For example, a redundant VNF B instance exists in NFVI-PoP

B. Therefore, NFVO selects the instance and re-constructs two VLs as shown in Figure 14, and the corresponding NS can be continued without re-instantiation.

7. Implementation Examples

tbd

8. Gaps and Open Questions

tbd

9. Conclusions

tbd

9.1. Relation to other IETF/IRTF activities

tbd

10. Acknowledgements

The research leading to some of the results described in this document has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 619609 - the UNIFY project. The views expressed here are those of the authors only. The European Commission is not liable for any use that may be made of the information in this document.

11. Contributors

This document is the result of merging multiple drafts. This section acknowledges those who provided important ideas and text into this document.

- Z. Qiang (Ericsson), M. Kind (DT-AG) from [I-D.unify-nfvrg-recursive-programming]
- R. Krishnan (Dell), D. Lopez (Telefonica) and S. Wright (AT&T) from [I-D.irtf-nfvrg-nfv-policy-arch]
- S. Lee (ETRI), S. Pack (KU), M-K. Shin (ETRI) and E. Paik (KT) from [I-D.irtf-nfvrg-resource-management-service-chain]

12. IANA Considerations

tbd

13. Security Considerations

tbd

14. References

14.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3060] Moore, B., Ellesson, E., Strassner, J., and A. Westerinen, "Policy Core Information Model -- Version 1 Specification", RFC 3060, DOI 10.17487/RFC3060, February 2001, <http://www.rfc-editor.org/info/rfc3060>.

[RFC3198] Westerinen, A., Schnizlein, J., Strassner, J., Scherling, M., Quinn, B., Herzog, S., Huynh, A., Carlson, M., Perry, J., and S. Waldbusser, "Terminology for Policy-Based Management", RFC 3198, DOI 10.17487/RFC3198, November 2001, <http://www.rfc-editor.org/info/rfc3198>.

[RFC3670] Moore, B., Durham, D., Strassner, J., Westerinen, A., and W. Weiss, "Information Model for Describing Network Device QoS Datapath Mechanisms", RFC 3670, DOI 10.17487/RFC3670, January 2004, <http://www.rfc-editor.org/info/rfc3670>.

14.2. Informative References

[CERI-DATALOG]
Ceri, S. and others, "What you always wanted to know about Datalog (and never dared to ask", IEEE Transactions on Knowledge and Data Engineering, (Volume: 1, Issue: 1), August 2002.

[ETSI-NFV-MANO]
ETSI, "Network Function Virtualization (NFV) Management and Orchestration V0.6.3", October 2014.

[ETSI-NFV-PER001]
ETSI, "Network Function Virtualization: Performance and Portability Best Practices v1.1.1", June 2014.

[ETSI-NFV-TERM]
ETSI, "NFV Terminology for Main Concepts in NFV", October 2013.

[ETSI-NFV-WHITE-PAPER]
ETSI NFV White Paper, "Network Functions Virtualisation, An Introduction, Benefits, Enablers, Challenges, & Call for Action", <http://portal.etsi.org/NFV/NFV_White_Paper.pdf>.

[I-D.ietf-bmwg-virtual-net]
Morton, A., "Considerations for Benchmarking Virtual Network Functions and Their Infrastructure", draft-ietf-bmwg-virtual-net-04 (work in progress), August 2016.

[I-D.irtf-nfvrg-nfv-policy-arch]
Figueira, N., Krishnan, R., Lopez, D., Wright, S., and D. Krishnaswamy, "Policy Architecture and Framework for NFV Infrastructures", draft-irtf-nfvrg-nfv-policy-arch-04 (work in progress), September 2016.

[I-D.irtf-nfvrg-resource-management-service-chain]
Lee, S., Pack, S., Shin, M., Paik, E., and R. Browne, "Resource Management in Service Chaining", draft-irtf-nfvrg-resource-management-service-chain-03 (work in progress), March 2016.

[I-D.liu-bmwg-virtual-network-benchmark]
Liu, V., Liu, D., Mandeville, B., Hickman, B., and G. Zhang, "Benchmarking Methodology for Virtualization Network Performance", draft-liu-bmwg-virtual-network-benchmark-00 (work in progress), July 2014.

[I-D.norival-nfvrg-nfv-policy-arch]
Figueira, N., Krishnan, R., Lopez, D., and S. Wright, "Policy Architecture and Framework for NFV Infrastructures", draft-norival-nfvrg-nfv-policy-arch-04 (work in progress), June 2015.

[I-D.unify-nfvrg-recursive-programming]
Szabo, R., Qiang, Z., and M. Kind, "Towards recursive virtualization and programming for network and cloud resources", draft-unify-nfvrg-recursive-programming-02 (work in progress), October 2015.

[ODL-GB-POLICY]
"OpenDaylight Group Based Policy", <https://wiki.opendaylight.org/view/Project_Proposals:Group_Based_Policy_Plugin>.

[ODL-NIC-PROJECT]
"OpenDaylight Network Intent Composition Project", <https://wiki.opendaylight.org/index.php?title=Network_Intent_Composition:Main#Friday_8AM_Pacific_Time>.

[ODL-SDN-CONTROLLER]
"OpenDaylight SDN Controller", <http://www.opendaylight.org/>.

[OPENSTACK]
"OpenStack", <http://www.openstack.org/>.

[OPENSTACK-CONGRESS]
"OpenStack Congress", <https://wiki.openstack.org/wiki/Congress>.

[OPENSTACK-NEAT]
"OpenStack Neat", <http://openstack-neat.org/>.

[OPENSTACK-NEUTRON]
"OpenStack Neutron", <https://wiki.openstack.org/wiki/Neutron>.

[POLICY-FRAMEWORK-WG]
"Policy Framework Working Group (IETF)", <http://www.ietf.org/wg/concluded/policy.html>.

[RESOURCE-MGMT-SERVICE-CHAIN]
Lee, S. and others, "Resource Management in Service Chaining", <https://datatracker.ietf.org/doc/draft-irtf-nfvrg-resource-management-service-chain/>.

[SDN-MULTI-DOMAIN]
Figueira, N. and R. Krishnan, "SDN Multi-Domain Orchestration and Control: Challenges and Innovative Future Directions", IEEE International Conference on Computing (ICNC), February 2015.

[VM-HOSTING-NET-CLUSTER]
Grit, L. and others, "Virtual Machine Hosting for Networked Clusters: Building the Foundations for "Autonomic" Orchestration", Virtualization Technology in Distributed Computing (VTDC), 2006.

Authors' Addresses

Robert Szabo (editor)
Ericsson
Konyves Kaman krt. 11
Budapest, EMEA 1097
Hungary

Phone: +36703135738
Email: robert.szabo@ericsson.com

Seungik Lee (editor)
ETRI
218 Gajeong-ro Yuseung-Gu
Daejeon 305-700
Korea

Phone: +82 42 860 1483
Email: seungiklee@etri.re.kr

Norival Figueira
Brocade

Email: nfigueir@Brocade.com

Internet Research Task Force (IRTF)
Internet-Draft
Intended status: Informational
Expires: September 21, 2016

S. Lee
ETRI
S. Pack
KU
M-K. Shin
ETRI
E. Paik
KT
R. Browne
Intel

March 20, 2016

Resource Management in Service Chaining
draft-irtf-nfvrg-resource-management-service-chain-03

Abstract

This document specifies problem definition and use cases of NFV resource management in service chaining for path optimization, traffic optimization, failover, load balancing, etc. It further describes design considerations and relevant framework for the resource management capability that dynamically creates and updates network forwarding paths (NFPs) considering resource constraints of NFV infrastructure.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 21, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 3
2. Terminology 4
3. Resource management in service chain 5
3.1. Resource scheduling among network services 5
3.2. Performance guarantee within a service chain 5
3.3. Multiple policies and conflicts 6
3.4. Dynamic adaptation of service chains 6
4. Use cases 7
4.1. Fail-over 7
4.2. Load balancing 8
4.3. Path optimization 8
4.4. Traffic optimization 8
4.5. Energy efficiency 9
5. Evaluation Model 9
5.1. System Model 9
5.2. Objective functions 11
5.2.1. Load balancing 11
5.2.2. Throughput optimization 11
5.2.3. Energy efficiency 12
6. Framework 12
7. Applicability to SFC 13
7.1. Related works in IETF SFC WG 13
7.2. Integration in SFC control-plane architecture 13
8. Security Considerations 15
9. IANA Considerations 15
10. Contributors 15
11. References 15
11.1. Normative References 15
11.2. Informative References 15
Acknowledgements 17
Authors' Addresses 17

1. Introduction

Network Functions Virtualisation (NFV) [ETSI-NFV-WHITE] offers a new way to design, deploy and manage network services. The network service can be composed of one or more network functions and NFV relocates the network functions from dedicated hardware appliances to generic servers, so they can run in software. Using these virtualized network functions (VNFs), one or more VNF forwarding graphs (VNF-FGs; a.k.a. service chains) can be associated to the network service, each of which describes a network connectivity topology, by referencing VNFs and Virtual Links that connect them. One or more network forwarding paths (NFPs) can be built on top of such a topology, each defining an ordered sequence of VNFs and Virtual Links to be traversed by traffic flows matching certain criteria.

The network service is instantiated by allocating NFVI resources for VNFs and VLs which constitute the VNF-FGs. Thus, the capacity and performance of the network service depends on the state and attributes of the network resources used for its VNF and VL instances. While this brings a similar problem to the VM placement optimization in a cloud computing environment, it differs as one or more VNF instances are interconnected for a single network service. For example, if one of the VNF instances in the VNF-FG gets failed or overloaded, the whole network service also gets affected. Thus, the VNF instances need to be carefully placed during NS instantiation considering their connectivity within NFPs. They also need to be monitored and dynamically migrated or scaled at run-time to adapt to changes in the resources.

The resource management problem in VNF-FGs matters not only to the performance and capacity of network services but also to the optimized use of NFVI resources. For example, if processing and bandwidth burden converges on the VNF instances placed in a specific NFVI-PoP, it may result in scalability problem of the NFV infrastructure. Thus care is encouraged to be taken in distributing load across local and external VNF instances at run-time.

This document addresses resource management problem in service chaining to optimize the NS performance and NFVI resource usage. It provides the relevant use cases of the resource management such as traffic optimization, failover, load balancing and further describes design considerations and relevant framework for the resource management capability that dynamically creates and updates NFP instances considering NFVI resource states for VNF instances and VL instances.

Note that this document mainly focuses on the resource management capability based on the ETSI NFV framework [ETSI-NFV-ARCH] but also studies contribution points to the work for control plane of SFC architecture [I-D.ietf-sfc-architecture] [I-D.ietf-sfc-control-plane].

2. Terminology

This document uses the following terms and most of them were reproduced from [ETSI-NFV-TERM].

- Network Functions (NF): A functional building block within a network infrastructure, which has well-defined external interfaces and a well-defined functional behavior.
- Network service: A composition of network functions and defined by its functional and behavioural specification.
- NFV Framework: The totality of all entities, reference points, information models and other constructs defined by the specifications published by the ETSI ISG NFV.
- Virtualised Network Function (VNF): An implementation of an NF that can be deployed on a Network Function Virtualisation Infrastructure (NFVI).
- NFV Infrastructure (NFVI): The NFV-Infrastructure is the totality of all hardware and software components which build up the environment in which VNFs are deployed.
- NFVI-PoP: A location or point of presence that hosts NFV infrastructure
- VNF Forwarding Graph (VNF-FG): A NF forwarding graph where at least one node is a VNF.
- Network Forwarding Path (NFP): The sequence of hardware/software switching ports and operations in the NFV network infrastructure as configured by management and orchestration that implements a logical VNF forwarding graph "link" connecting VNF "node" logical interfaces.
- Virtual Link: A set of connection points along with the connectivity relationship between them and any associated target performance metrics (e.g. bandwidth, latency, QoS). The Virtual Link can interconnect two or more entities (e.g., VNF components, VNFs, or PNFs).

3. Resource management in service chain

This section addresses several issues for considerations in NFV resource management of service chain.

3.1. Resource scheduling among network services

In the NFV framework, network services are realized with NS instantiation procedures at which virtualized NFVI resources are assigned to the VNFs and VLs which constitute VNF-FGs of the network service. The NFVI resources are placed and located along the VNF-FG by NFV Orchestrator (NFVO) dynamically according to:

- Resource availability,
- Deployment templates which define resource requirements of NS instances and VNF instances to support KPIs (e.g., capacity and performance) of the network service, and
- Resource policies which define how to govern NFVI resources for NS instances and VNF instances (e.g., affinity/anti-affinity rules, scaling, and fault management) to support an efficient use of NFVI resources as well as KPIs of the network service.

In order to satisfy the deployment templates and resource policies, VNF-FGs of the network services need to be built by considering the state of NFVI resources for VNF instances (e.g., availability, throughput, load, disk usage) and VL instances (e.g., bandwidth, delay, delay variation, packet loss).

However, since the NFVI resources are shared by different network services and their deployment constraints are very different from each other, it is required to carefully schedule the NFVI resources for multiple network services to optimize their KPIs.

3.2. Performance guarantee within a service chain

In NFV, a network service is composed of one or more virtualized network functions which are connected via virtual links along NFPs specified for a traffic flow for the network service. Thus, the performance of a network service is determined by the performance and capability of a coupling of VNF instances and VL instances. For example, if one of the VNF instances or VL instances of an NFP gets failed or overloaded, the whole network service also gets affected. Thus, the VNF instances need to be carefully placed during NS instantiation considering their connectivity within NFPs.

This performance coupling can be handled by considering deployment rules for affinity/anti-affinity, geography, or topological locations of VNFs; and QoS of virtual links.

Another important factor for virtual links is the inter-connectivity between different NFVI-PoPs, which is an enabler of resource sharing among different NFVI-PoPs. When the VNF instances of a network service are allocated at different NFVI-PoPs, the NFVI-PoP interconnect may be a bottleneck point which needs to be monitored to support KPIs of the service chain.

3.3. Multiple policies and conflicts

The NFVI resources for a network service should be allocated and managed according to a NS policy given in the network service descriptor (NSD), which describes how to govern NFVI resources for VNF instances and VL instances to support KPIs of the network service. The examples of NS policy are affinity/anti-affinity, scaling, fault and performance, geography, regulatory rules, NS topology, etc. Since network services may have different NS policies for their own deployment and performance, this may cause resource management difficult within the shared NFVI resources.

For network-wide (or NS-wide) resource management, NFVI policy (or network policy) can be also provided. It may describe the resource management policy for optimized use of infrastructure resources rather than the performance of a single network service. The examples of NFVI policy are NFVI resource access control, reservation and/or allocation policies, placement optimization based on affinity and/or anti-affinity rules, geography and/or regulatory rules, resource usage, etc.

Multiple administrative domains or subsystems may have different NFVI policies so that it may bring some conflicts when enforcing them in a global infrastructure. There could be a similar problem among NS policies and NFVI policies.

Note that the similar topics are being studied in [I-D.irtf-nfvrg-nfv-policy-arch]

3.4. Dynamic adaptation of service chains

The performance and capability of NFVI resources may vary in time due to different uses and management policies of the resources. If some changes in the resources make the service quality unacceptable, the VNF instances can be scaled according to the given auto-scaling policies. But it’s only for local quality of the VNF.

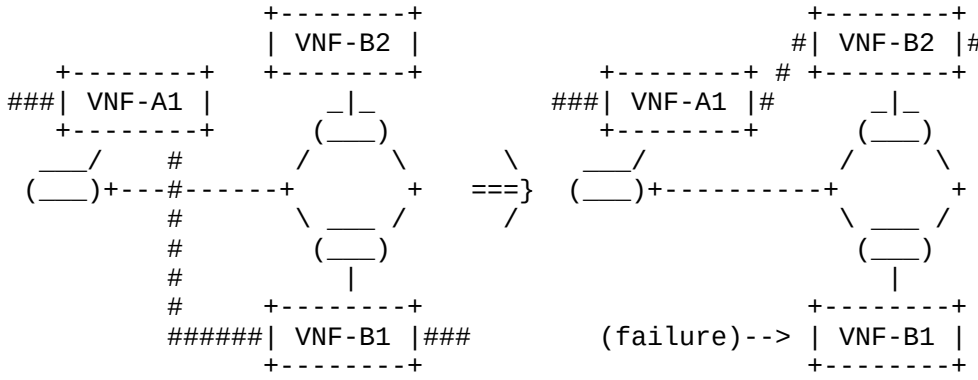
In order to provide optimized KPIs to network services, the NFP instances need to dynamically adapt to the changes of the resource state at run-time. The performance of the whole NFP instance should be measured by monitoring the resource state of VNF instances and VL instances. Based on the monitoring results, some VNF instances may be determined and relocated at different virtualized resources with better performance and capabilities.

4. Use cases

In this section, several (but not exhausted) use cases for resource management in service chaining are provided: fail-over, load balancing, path optimization, traffic optimization, and energy efficiency.

4.1. Fail-over

For service continuity, failure of a VNF instance needs to be detected and the failed one needs to be replaced with the other one which is available to use as per redundancy policy. Figure 1 presents an example of the fail-over use case. A network service is defined as a chain of VNF-A and VNF-B; and the service chain is instantiated with VNF-A1 and VNF-B1 which are instances of VNF-A and VNF-B, respectively. In the meantime, failure of VNF-B1 is detected so that VNF-B2 replaces the failed one for fail-over of the NFP.



NFP

Figure 1: A fail-over use case

The above is in the case where there is a 1+1 or 1:N redundancy scheme. In event that VNF instance overloads before NFVO has time to

scale out, or when resources do not permit a scale-out then we can route the service chain deterministically to a remote VNF instance. This adaptation may be revertive or non-revertive dependent on service provider policy and resource availability.

4.2. Load balancing

A single VNF instance may be a bottleneck point of a service chain due to its overload. It may affect the performance of the whole service chain consequently so that an NFP instance needs to be built to avoid bottleneck points or maintained to distribute workloads of overloaded VNF instances.

With NFVI-PoP Interconnect, service chains can be balanced between NFVI-PoPs in a way that best utilises NFV infrastructure and ensures service integrity. The wide area conditions can be monitored in real-time to provide KPIs, such as BW, delay, delay variation and packet loss per QoS class to the service chaining application which may enable use of external VNF instances when there is an overload or failure condition in the local NFVI-PoP. In this way the service chaining application can make a service chain reroute decision (in the event of failure and/or overload) that is network and platform-aware. The service chaining application understands the state of external VNFs and WAN conditions per QoS class between the local NFVI-PoP and remote NFVI-PoP in real-time.

4.3. Path optimization

Traffic for a network service traverses all of the VNF instances and the connecting VL instances given by a NFP instance to reach a target end point. Thus, quality of the network service depends on the resource constraints (e.g., processing power, bandwidth, topological locations, latency) of VNF instances, VL instances including NFVI-PoP interconnects. In order to optimize the path of the network service, the resource constraints of VNF instances and VLs need to be considered at constructing NFPs. Since the resource state may vary in time during the service, NFP instances also need to adapt to the changes of resource constraints of the VNF instances and VL instances by monitoring and replacing them at run-time.

4.4. Traffic optimization

A network operator may provide multiple network services with different VNF-FGs and different flows of traffic traverse between source and destination end-points along the VNF-FGs. For efficiency of resource usage, the NFP instances need to be built by default to localize the traffic flows and to avoid processing and network bottlenecks. It is only in the case of local failure or overload

(whereby the NFVO is unable or has not completed a scale-out of on-site resources) that NFP instances would be constructed between NFVI-PoPs. In this case, multiple VNF instances of different NFVI-PoPs need to be considered together at constructing a new NFP instance or adapting one.

4.5. Energy efficiency

Energy efficiency in the network is getting important to reduce impact on the environment so that energy consumption of VNF instances using NFVI resources (e.g., compute, storage, I/O) needs to be considered at NFP instantiation or adaptation. For example, a NFP can be instantiated as to make traffic flows aggregated into a limited number of VNF instances as much as its performance is preserved in a certain level. Policy may vary between centralized or distributed NFV applications, and could include policies for even energy distribution between sites, time-of-day etc.

5. Evaluation Model

To derive specific algorithms for use cases discussed in Section 4, an evaluation model for a service chain (or a NFP) needs to be developed, which can address two problems for a given network topology and input parameters (e.g., VL/VNF capacity, incoming traffic flows, etc.) : 1) how much traffic flows pass on each VL instance and 2) how much processing capacity is needed for the installed VNF instance. This section first describes the system model and then presents main objectives for the evaluation model.

5.1. System Model

The system model considers the following network topology. The network topology under consideration is composed of start/end points and multiple NFVI-PoPs where multiple VNF instances locate. On the other hand, VL instances inter-connect VNF instances in NFVI-PoPs.

Start and end points are incoming and outgoing points of traffic flow for a given network service, respectively. Specifically, the amount of incoming traffic flows for a network service (i.e., a VNF-FG) at the start point is given as an input parameter in the model.

Under the network topology, the network traffic is processed by one or more VNF instances and delivered via VL instances. Thus, the VNF processing capacity can be defined as the maximum amount of traffic flows that a VNF instance can process according to the resource allocation policies defined in its deployment template. The VL capacity can be also defined as the maximum amount of traffic flows

that can pass on a VL instance according to the resource allocation policies defined in the deployment template.

In NFV, traffic flows for a VNF-FG should be processed according to the VNF order described in the given VNF-FG. Accordingly, traffic flows at the start point should not be processed by any VNF. Meanwhile, traffic flows at the end point should be processed by all VNFs specified in the given VNF-FG.

In a given VNF-FG, VNFs should be individually placed on multiple NFVI-PoPs. Therefore, a decision variable, VNF placement indicator function (VPIF), is defined as:

- VNF placement indicator function (VPIF): indicator function (i.e., 0 or 1) to represent the location (i.e., a NFVI-PoP) where the VNF instance is placed.

Intuitively, the amount of traffic flows that pass a VL instance should not exceed the VL capacity to avoid any overload at the VL instance. Likewise, the amount of incoming traffic flows to a VNF instance should not exceed the VNF processing capacity. (These constraints will be covered in the following paragraphs) Therefore, traffic flows for a network service (i.e., a VNF-FG) should be distributed to multiple NFPs depending on resource and capacity constraints for VNF and VL instances. Moreover, multiple network services can be supported by distributing traffic flows for each network service. Therefore, another decision variable, traffic flow ratio (TFR), is defined as:

- Traffic flow ratio (TFR): the ratio of the traffic flows distributed to each NFP. Therefore, the amount of traffic flows that passes on each NFP is the product of TFR and the amount of incoming traffic flows for a network service. Note that TFR and the amount of incoming traffic flows can be computed by measuring the amount of traffic flows that passes on each VL.

The constraints regarding the amount of network traffic and capacity of VNF and VL instances can be specified as follows.

- Network traffic conservation constraints: In the VNF-FG system model, the amount of network traffic should be conserved within a VNF-FG. That is, 1) the amount of incoming network traffic to a VNF instance should be equal (more or less in case of packet manipulation) to the amount of outgoing network traffic from the VNF instance; and 2) the amount of incoming network traffic to a VNF instance should not exceed the flow rate of the corresponding NFP which can be determined by TFR.

- Network traffic processing order constraints: As defined in the VNF-FG, the network traffic can be processed by a VNF instance only after being processed by the preceding VNF instances along the NFP. Similarly, the incoming network traffic to a NFP should be firstly processed by the VNF instance which is located at the ingress point of the NFP; and the outgoing network traffic from a NFP should be the result of processing by every VNF instance in the order defined by the NFP.

- Link and processing capacity constraints: The amount of incoming network traffic to a VL instance should not exceed the given link capacity of the VL to avoid any congestion at the link. Likewise, the amount of incoming network traffic to a VNF instance should not exceed the processing capacity of the VNF.

This system model can be exploited to obtain the optimal solutions of network resource (i.e., VNF and VL instances) placement for network resource usage, network service throughput, and so on. This optimization problem can be solved, for example with linear programming (LP), by defining different objective functions.

5.2. Objective functions

In the evaluation model, three objectives are considered including, but not limited to, 1) load balancing, 2) flow throughput maximization, and 3) energy efficiency.

5.2.1. Load balancing

For load balancing for a network service, the remaining capacity for VNF instances and VL instances should be balanced to avoid any bottlenecks. To this end, the minimum remaining processing capacity for VNF instances and the minimum remaining link capacity for VL instances should be maximized.

5.2.2. Throughput optimization

On the other hand, the flow throughput considers both throughputs for VNF processing and for VL instance. Then, the throughput of an NFP can be calculated as the product of TFR and the sum of capacities, and the total throughput is the sum of computed throughputs for all NFPs. By maximizing the total flow throughput, it is possible to reduce the network service time.

5.2.3. Energy efficiency

Since each VNF instance consumes an amount of energy for processing its function and transmitting/receiving traffic flows across VL instances, the energy consumption for each VNF instance should be minimized for energy efficiency of network services. Detailed model is under construction.

6. Framework

To support the aforementioned use cases, it is required to support resource management capability which provides service chain (or NFP) construction and adaptation by considering resource state or constraints of VNF instances and VL instances which connect them. The resource management operations for service chain construction and adaptation can be divided into several sub-actions:

- Locate VNF instances
- Evaluate the performance of VNF instances and VL instances
- Relocate (or scale) VNF instances to update a NFP instance
- Monitor state or resource constraints of a VNF instance, VL instances including NFVI-PoP interconnects

As listed above, VNF instances are relocated according to monitoring or evaluation results of performance metrics of the VNF instances and VL instances. Studies about evaluation methodologies and performance metrics for VNF instances and NFVI resources can be found at [ETSI-NFV-PER001] [I-D.liu-bmwg-virtual-network-benchmark] [I-D.ietf-bmwg-virtual-net]. The performance metrics of VNF instances and VL instances specific to service chain construction and adaptation can be defined as follows:

- availability (or failure) of a VNF instance and a VL instance
- a topological location of a VNF instance
- CPU and memory utilization rate of a VNF instance
- a throughput of a VNF instance
- energy consumption of a VNF instance
- bandwidth of a VL instance
- packet loss of a VL instance

- latency of a VL instance
- delay variation of a VL instance

The resource management functionality for dynamic service chain adaptation takes role of NFV orchestration with support of VNF manager (VNFM) and Virtualised Infrastructure Manager (VIM) in the NFV framework [ETSI-NFV-ARCH]. Detailed functional building block and interfaces are still under study.

7. Applicability to SFC

7.1. Related works in IETF SFC WG

IETF SFC WG provides a new service deployment model that delivers the traffic along the predefined logical paths of service functions (SFs), called service function chains (SFCs) with no regard of network topologies or transport mechanisms. Basic concept of the service function chaining is similar to VNF-FG where a network service is composed of SFs and deployed by making traffic flows traversed instances of the SFs in a pre-defined order.

There are several works in progress in IETF SFC WG for resource management of service chaining. [I-D.ietf-sfc-architecture] defines SFC control plane that selects specific SFs for a requested SFC, either statically or dynamically but details are currently outside the scope of the document. There are other works [I-D.ietf-sfc-control-plane] [I-D.lee-sfc-dynamic-instantiation] [I-D.krishnan-sfc-oam-req-framework] [I-D.ietf-sfc-oam-framework] which define the control plane functionality for service function chain construction and adaptation but details are still under study. While [I-D.dunbar-sfc-fun-instances-restoration] and [I-D.meng-sfc-chain-redundancy] provide detailed mechanisms of service chain adaptation, they focus only on resilience or fail-over of service function chains.

7.2. Integration in SFC control-plane architecture

In SFC WG, [I-D.ietf-sfc-control-plane] describes requirements for conveying information between Service Function Chaining (SFC) control elements (including management components) and SFC functional elements. It also identifies a set of control interfaces to interact with SFC-aware elements to establish, maintain or recover service function chains.

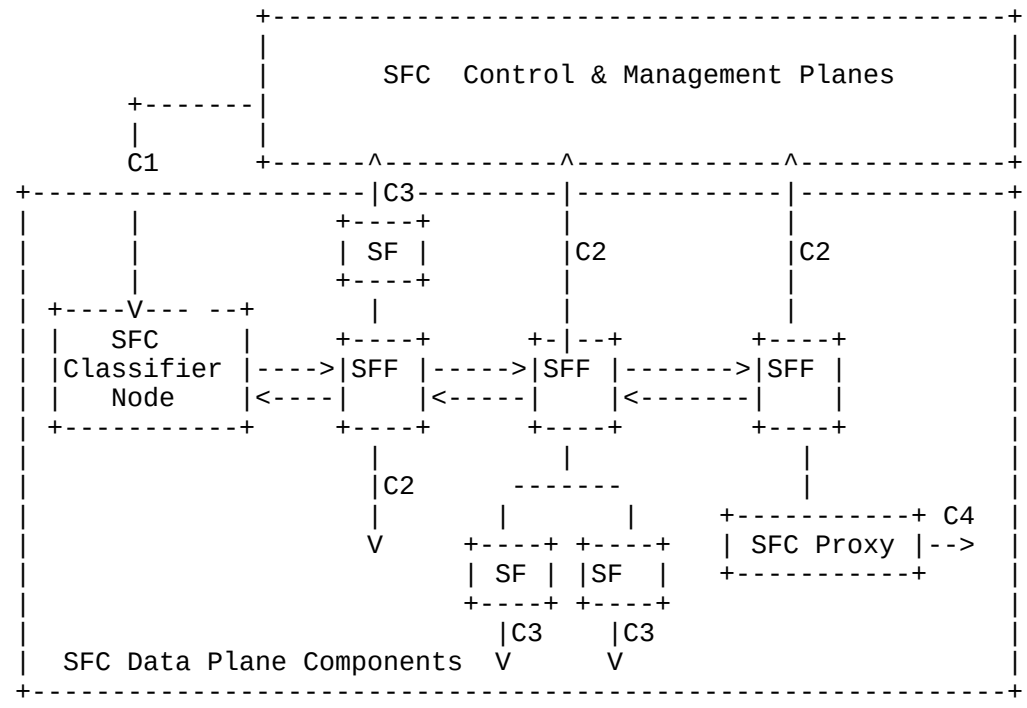


Figure 2: SFC control plane overview

The service chain adaptation addressed in this document may be integrated into the SFC Control & Management Planes and may use the C2 and C4 interfaces for monitoring or collecting the resource constraints of VNF instances, NFVI-PoP interconnects and VL instances.

To prevent constant integration between the application and probing functions we would propose a 3-tier architecture per NFVI-PoP.

- Top level application control at the SFC Control & Management Planes
- An abstraction layer between the application layer and the probing layer. This would decouple NFVI and link monitoring methods from the application layer
- A probing layer that monitors VNF, physical and virtual link resources

Note that SFC does not assume that Service Functions are virtualized. Thus, the parameters of resource constraints may differ, and it needs further study for integration.

8. Security Considerations

TBD.

9. IANA Considerations

TBD.

10. Contributors

In addition to the authors, the following individuals contributed to the content.

Insun Jang
Korea University
zerantoss@korea.ac.kr

11. References

11.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

11.2. Informative References

[ETSI-NFV-ARCH]
ETSI, "ETSI NFV Architectural Framework v1.1.1", October 2013.

[ETSI-NFV-MANO]
ETSI, "Network Function Virtualization (NFV) Management and Orchestration V0.6.3", October 2014.

[ETSI-NFV-PER001]
ETSI, "Network Function Virtualization: Performance and Portability Best Practices v1.1.1", June 2014.

[ETSI-NFV-TERM]
ETSI, "NFV Terminology for Main Concepts in NFV", October 2013.

[ETSI-NFV-WHITE]
ETSI, "NFV Whitepaper 2", October 2013.

[I-D.dunbar-sfc-fun-instances-restoration]
Dunbar, L. and A. Malis, "Framework for Service Function Instances Restoration", draft-dunbar-sfc-fun-instances-restoration-00 (work in progress), April 2014.

[I-D.ietf-bmwg-virtual-net]
Morton, A., "Considerations for Benchmarking Virtual Network Functions and Their Infrastructure", draft-ietf-bmwg-virtual-net-01 (work in progress), September 2015.

[I-D.ietf-sfc-architecture]
Halpern, J. and C. Pignataro, "Service Function Chaining (SFC) Architecture", draft-ietf-sfc-architecture-11 (work in progress), July 2015.

[I-D.ietf-sfc-control-plane]
Li, H., Wu, Q., Huang, O., Boucadair, M., Jacquenet, C., Haeffner, W., Lee, S., Parker, R., Dunbar, L., Malis, A., Halpern, J., Reddy, T., and P. Patil, "Service Function Chaining (SFC) Control Plane Components & Requirements", draft-ietf-sfc-control-plane-03 (work in progress), January 2016.

[I-D.ietf-sfc-oam-framework]
Aldrin, S., Krishnan, R., Akiya, N., Pignataro, C., and A. Ghanwani, "Service Function Chaining Operation, Administration and Maintenance Framework", draft-ietf-sfc-oam-framework-01 (work in progress), February 2016.

[I-D.irtf-nfvrg-nfv-policy-arch]
Figueira, N., Krishnan, R., Lopez, D., Wright, S., and D. Krishnaswamy, "Policy Architecture and Framework for NFV Infrastructures", draft-irtf-nfvrg-nfv-policy-arch-03 (work in progress), March 2016.

[I-D.krishnan-sfc-oam-req-framework]
Krishnan, R., Ghanwani, A., Gutierrez, P., Lopez, D., Halpern, J., Kini, S., and A. Reid, "SFC OAM Requirements and Framework", draft-krishnan-sfc-oam-req-framework-00 (work in progress), July 2014.

[I-D.lee-sfc-dynamic-instantiation]
Lee, S., Pack, S., Shin, M., and E. Paik, "SFC dynamic instantiation", draft-lee-sfc-dynamic-instantiation-01 (work in progress), October 2014.

[I-D.liu-bmwg-virtual-network-benchmark]
Liu, V., Liu, D., Mandeville, B., Hickman, B., and G. Zhang, "Benchmarking Methodology for Virtualization Network Performance", draft-liu-bmwg-virtual-network-benchmark-00 (work in progress), July 2014.

[I-D.meng-sfc-chain-redundancy]
Meng, W. and C. Wang, "Redundancy Mechanism for Service Function Chains", draft-meng-sfc-chain-redundancy-02 (work in progress), October 2015.

[Jang-2016]
Jang, I., Choo, S., Kim, M., Pack, S., and M. Shin, "Optimal Network Resource Utilization in Service Function Chaining", IEEE Conference on Network Softwarization (NetSoft) (To be published), June 2016.

Acknowledgements

The authors would like to thank Sukjin Choo and Myeongsu Kim for the review and comments.

Authors' Addresses

Seungik Lee
ETRI
218 Gajeong-ro Yuseung-Gu
Daejeon 305-700
Korea

Phone: +82 42 860 1483
Email: seungiklee@etri.re.kr

Sangheon Pack
Korea University
145 Anam-ro, Seongbuk-gu
Seoul 136-701
Korea

Phone: +82 2 3290 4825
Email: shpack@korea.ac.kr

Myung-Ki Shin
ETRI
218 Gajeong-ro Yuseung-Gu
Daejeon 305-700
Korea

Phone: +82 42 860 4847
Email: mkshin@etri.re.kr

EunKyoung Paik
KT
17 Woomyeon-dong, Seocho-gu
Seoul 137-792
Korea

Phone: +82 2 526 5233
Email: eun.paik@kt.com

Rory Browne
Intel
Dromore House, East Park
Shannon, Co. Clare
Ireland

Phone: +353 61 477400
Email: rory.browne@intel.com

NFV Research Group
Internet-Draft
Intended status: Informational
Expires: April 29, 2018

M-K. Shin, Ed.
ETRI
K. Nam
Friesty
S. Pack
KU
S. Lee
ETRI
R. Krishnan
Dell

October 29, 2017

Verification of NFV Services : Problem Statement and Challenges
draft-irtf-nfvrg-service-verification-05

Abstract

NFV relocates network functions from dedicated hardware appliances to generic servers, so they can run in software. However, incomplete or inconsistent configuration of virtualized network functions (VNFs) and forwarding graph (FG, aka service chain) could cause break-down of the supporting infrastructure. In this sense, verification is critical for network operators to check their requirements and network properties are correctly enforced in the supporting infrastructures. Recognizing these problems, we discuss key properties to be checked on NFV services. Also, we present challenging issues related to verification in NFV environments.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 27, 2017.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 3
1.1. Terminology 3
2. Problem statement 3
2.1. Dependencies of network service components in NFV framework .3
2.2. Invariant and error check in VNF FGs 4
2.3. Load Balancing and optimization among VNF Instances 4
2.4. Policy and state consistency on NFV services 4
2.5. Performance 5
2.6. Security 5
3. Examples - NS policy conflict with NFVI policy 6
4. Requirements of verification framework 7
5. Challenging issues 8
5.1. Consistency check in distributed state 8
5.2. Intent-based service composition 8
5.3. Finding infinite loops in VNF FGs 8
5.4. Complexity of live traffic verification 9
5.5. Languages and their semantics 9
5.6. Stateful VNFs with multiple physical views 9
6. Gap analysis - open source projects 10
6.1. OPNFV 10
6.2. ODL 12
6.3. Summary 13
7. Security Considerations 13
8. Acknowledgements 14
9. References 14
Author's Address 16

1. Introduction

NFV is a network architecture concept that proposes using IT virtualization related technologies, to virtualize entire classes of network service functions into building blocks that may be connected, or chained, together to create end-to-end network services. NFV service is defined as a composition of network functions and described by its functional and behavioral specification, where network functions (i.e., firewall, DPI, SSL, load balancer, NAT, AAA, etc.) are well-defined, hence both their functional behavior as well as their external interfaces are described in each specifications.

In NFV, a VNF is a software package that implements such network functions. A VNF can be decomposed into smaller functional modules or APIs for scalability, reusability, and/or faster response [ETSI-NFV-Arch],[ETSI-NFV-MANO]. These modular updates or composition for a network function may lead to many other verification or security issues. In addition, a set of ordered network functions which build FGs may be connected, or chained, together to create an end-to-end network service. Multiple VNFs can be composed together to reduce the complexity of management and VNF FGs. While autonomic networking techniques could be used to automate the configuration process including FG updates, it is important to take into account that incomplete and/or inconsistent configuration may lead to verification issues. Moreover, automation of NFV process with integration of SDN may lead the network services to be more error-prone. In this sense, we need to identify and verify key properties to be correct before VNFs and FGs are physically placed and realized in the supporting infrastructure.

1.1. Terminology

This document draws freely on the terminology defined in [ETSI-NFV-Arch].

2. Problem statement

The verification services should be able to check the following properties:

2.1. Dependencies of network service components in NFV framework

In NFV framework, there exist several network service components including NFVI, VNFs, MANO, etc. as well as network controller and switches to realize end-to-end network services. Unfortunately, these components have intricate dependencies that make operation incorrect. In this case, there is inconsistency between states stored and

managed in VNF FGs and network tables (e.g., flow tables), due to communication delays and/or configuration errors. For example, if a VNF is replicated into the other same one for the purpose of load balance and a new FG is established through the copied one, but all the state/DBs replication is not finished yet due to delays, this can lead to unexpected behaviors or errors of the network service. Therefore, these dependencies make it difficult to correctly compose NFV-enabled end-to-end network services.

2.2. Invariant and error check in VNF FGs

In VNF FGs, an infinite loop construction should be avoided and verified. Let us consider the example. Two VNF A and VNF B are located in the same service node X whereas another VNF C resides in other service node Y [SIGCOMM-Gember]. Also, the flow direction is from X to Y, and the given forwarding rule is A->C->B. In such a case, service node Y can receive two ambiguous flows from VNF A: 1) one flow processed by VNF A and 2) another flow processed by VNF A, B, and C. For the former case, the flow should be processed by VNF C whereas the latter flow should be further routed to next service nodes. If these two flows cannot be distinguished, service node Y can forward the flow to service node X even for the latter case and a loop can be formed. To avoid the infinite loop formation, the forwarding path over VNF FG should be checked in advance with the consideration of physical placement of VNF among service nodes. Also, reactive verification may be necessary, since infinite loop formation may not be preventable in cases where configuration change is happening with live traffic.

In addition, isolation between VNFs (e.g. confliction of properties or interference between VNFs) and consistent ordering of VNF FGs should be always checked and maintained.

2.3. Load balancing among VNF instances

In VNF FG, different number of VNF instances can be activated on several service nodes to carry out the given task. In such a situation, load balancing among the VNF instances is one of the most important considerations. In particular, the status in resource usage of each service node can be different and thus an appropriate amount of jobs should be distributed to the VNF instances. To guarantee well-balanced load among VNF instances, the correctness of hash functions for load balancing needs to be verified. Moreover, when VNF instances locate in physically different service nodes, simple verification of load balancing in terms of resource usage is not sufficient because different service nodes experience diverse network conditions (e.g., different levels of network congestion)[ONS-Gember]. Therefore, it is needed to monitor global network condition

as well as local resource condition to achieve the network-wide load balancing in VNF FGs. Also, whether the monitoring function for network/compute/storage resources is correctly working should be checked.

2.4. Policy and state consistency on NFV services

In VNF FG, policy to specific users can be dynamically changed. For example, a DPI VNF can be applied only in the daytime in order to prohibit from watching adult contents while no DPI VNFs applied during the nighttime. When the policy is changed, the changed policy should be reconfigured in VNF service nodes as soon as possible. If the reconfiguration procedure is delayed, inconsistent policies may exist in service nodes. Consequently, policy inconsistency or confliction needs to be checked. Also in some situations, states for VNF instances may be conflicted or inconsistent. Especially when a new VNF instance is instantiated for scale-up and multiple VNF instances are running, these multiple VNF instances may have inconsistent states owing to inappropriate instantiation procedure [SIGCOMM-Gember]. In particular, since the internal states of VNF instances (e.g., the instantaneous state of CPU, register, and memory in virtual machine) are not easily-visible, a new way to check the VNF internal states should be devised.

2.5. Performance

In VNF FG, VNF instances can be located in different service nodes and these service nodes have different load status and network conditions. Consequently, the overall throughput of VNF FG is severely affected by the service nodes running VNF instances. For example, if a VNF instance locates in a heavily loaded service node, the service time at the service node will be increased. In addition, when a VNF FG includes a bottleneck link experiencing congestion, the end-to-end performance (e.g., latency and throughput) in the VNF FG can be degraded. Furthermore, policies on the performance such as minimum bandwidth or latency can be given to VNFs or their FGs. Therefore, the identification of bottleneck link and node is the first step for performance verification or guarantee of the VNF FG [ONS-Gember]. After detecting the bottleneck link/node, the VNF requiring scale up or down can be identified and the relocation of VNF instance among service nodes can be determined.

2.6. Security

How to verify security holes in VNF FG is another important consideration. In terms of security services, authentication, data integrity, confidentiality, and replay protection should be provided. On the other hand, several VNFs (e.g., NAT) can modify or update

packet headers and payload. In these environments, it is difficult to protect the integrity of flows traversing such VNFs. Another security concern in the VNF FG is distributed denial of service (DDoS) to a specific service node. If an attacker floods packets to a target service node, the target service node cannot perform its functions correctly. Therefore, such security attacks in the VNF FG should be detected and handled in an efficient manner. In the case of DDoS, adding a DDoS appliance as the first element in the service chain would help alleviate the problem. Moreover, unknown or unauthorized VNFs can run and thus how to identify those problems is another security challenge.

3. Examples - NS policy conflict with NFVI policy

Another target of NFV verification is conflict of Network Service (NS) policies against global network policy, called NFVI policy.

NFV allocates and manages NFVI resources for a network service according to an NS policy given in the network service descriptor (NSD), which describes how to govern NFVI resources for VNF instances and VL instances to support KPIs of the network service. Example factors of the NS policy are resource constraints (or deployment flavor), affinity/anti-affinity, scaling, fault and performance management, NS topology, etc.

For a network-wide (or NS-wide) management of NFVI, NFVI policy (or global network policy) can be provided to describe how to govern the NFVI resources for optimized use of the infrastructure resources (e.g., energy efficiency and load balancing) rather than optimized performance of a single network service. Example factors of the NFVI policy are NFVI resource access control, reservation and/or allocation policies, placement optimization based on affinity and/or anti-affinity rules, geography and/or regulatory rules, resource usage, etc.

While both of the policies define the requirements for resource allocation, scheduling, and management, the NS policy is about a single network service; and the NFVI policy is about the shared NFVI resources, which may affect all of the given network services globally. Thus, some of NS and NFVI policies may be inconsistent with each other when they have contradictive resource constraints on the shared NFVI resources. Examples of the policy conflicts are as follows:

<Example conflict case #1>

- NS policy of NS_A (composed of VNF_A and VNF_B)

- Resource constraints: 3 CPU core for VNF_A and 2 CPU core for VNF_B
- Affinity rule between VNF_A and VNF_B

o NFVI policy
- No more than 4 CPU cores per physical host

o Conflict case
- The NS policy cannot be met within the NFVI policy

<Example conflict case #2>

o NS policy of NS_B (composed of VNF_A and VNF_B)
- Affinity rule between VNF_A and VNF_B

o NFVI policy
- Place VM whose outbound traffic is larger than 100Mbps at POP_A
- Place VM whose outbound traffic is smaller than 100Mbps at POP_B

o Conflict case
- If VNF_A and VNF_B generate traffic in 150Mbps and 50Mbps, respectively,
- VNF_A and VNF_B need to be placed at POP_A and POP_B, respectively according to the NFVI policy
- But it will violate the affinity rule given in the NS policy

<Example conflict case #3>

o NS policy of NS_C (composed of VNF_A and VNF_B)
- Resource constraints: VNF_A and VNF_B exist in the same POP
- Auto-scaling policy: if VNF_A has more than 300K CPS, scale-out

o NFVI policy
- No more than 10 VMs per physical host in POP_A

o Conflict case
- If CPS of VNF_A in POP_A gets more than 300K CPS,
- and if there is no such physical host in the POP_A whose VMs are smaller than 10,
- VNF_A need to be scaled-out to other POP than POP_A according to the NFVI policy
- But it will violate the NS policy

4. Requirements of verification framework

The verification framework addressed in this document follows [ETSI-NFV-Testing]. [ETSI-NFV-Testing] covers the following aspects of pre-deployment testing: 1) assessing the performance of the NFVI and its

ability to fulfil the performance and reliability requirements of the VNFs executing on the NFVI, 2) data and control plane testing of VNFs and their interactions with the NFV Infrastructure and the NFV MANO, and 3) validating the performance, reliability and scaling capabilities of network services.

A verification framework for NFV-based services also needs to satisfy the following requirements:.

- o R1 : It should be able to check global and local properties and invariants. Global properties and invariants relate to the entire VNFs, and local properties and invariants relates to the specific domain or resources that some of the VNFs are using. For example, Loop-freeness and isolation between VNFs can be regarded as global. The policies that are related only to the specific network controllers or devices are local.

- o R2 : It should be able to access to information on the entire network states and resource usage whenever verification tasks are started. It can directly manage the states of network and NFV-based services through databases or any solution that specializes in dealing with the network topology and configurations, or can utilize the functions provided by NFV M&O and VNFI solutions to get or set the states at any time.

- o R3 : It should be independent from specific solutions and frameworks, and provide standard APIs.

- o R4 : It should able to process standard protocols such as NetConf, YANG, OpenFlow, and northbound and southbound interfaces that are related network configurations, and used by OSS.

5. Challenging issues

There are emerging challenges that the verification services face with.

5.1. Consistency check in distributed state

Basically, NFV states as well as SDN controllers are distributed. Writing code that works correctly in a distributed setting is very hard. Therefore, distributed state management and consistency check has challenging issues. Some open source projects such as ONOS offers a core set of primitives to manage this complexity. RAFT algorithm

[RAFT] is used for distribution and replication. Similarly, Open daylight project has a clustering concept to management distributed state. There is no "one-size-fits-all" solution for control plane data consistency.

5.2. Intent-based service composition

Recently, Intent-based policing feature/approach/mechanism has been added in open source projects [ODL],[ONOS]. The Intent allows for a descriptive way to get what is desired from the infrastructure, unlike the current NFV description and SDN interfaces which are based on describing how to provide different services. This Intent will accommodate orchestration services and network and business oriented SDN/NFV applications, including OpenStack Neutron, Service Function Chaining, and Group Based Policy. A Intent compiler translates and compiles it into low level instructions (e.g., SDN controller/OpenStack primitives) for network service components. In this sense, error checking and debugging are critical for reliable Intent-based service composition.

5.3. Finding infinite loops

General solutions for the infinite loop can lead to intractable problem (e.g. the halting problem). To make the verification practical and minimize the complexity, some restrictions are required. Finding cycle can be processed in polynomial time but the restriction could be too much for some cases that service functions or network flows requires finite loops.

5.4. Complexity of live traffic verification

It is a known fact that the complexity of verification tasks for the real and big problem is high. A few invariants can be checked in real-time but it would be impossible if the size of VNFs increases or properties to be checked are complex.

5.5. Languages and their semantics

For the verification, all the information on VNFs including configurations, dynamic states and their temporal orderings need to be precisely expressed in platform independent languages based on formal semantics. The languages and their semantic models should be optimized to the verification frameworks as well as the NFV infrastructures.

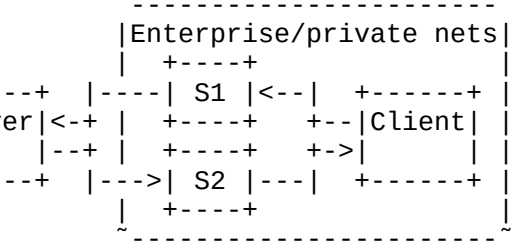
5.6. Stateful VNFs with multiple physical views

The correctness of VNFs whose behaviors depend on the previous states

(packets, actions, etc) and whose physical entities are multiple should be checked differently than the stateless ones. Such VNFs include firewall, load balancer, NAT, flow rules with counter or soft timeout.

- Case 1:
  If a firewall service is implemented over two physical OpenFlow switches, there could be two paths that the client-server packets go through. If the packets between client and server go through the same switch, the firewall functions correctly. However if packets from client to server go through S1 but packets from server to client come back through S2, those flows could be blocked and lead to false-negative result.

  To mitigate the situation, states of all instances for one logical VNF must be considered to verify the correctness.



- Case 2:
  If there are VNFs whose behavior depend on the previous VNF, those dependency must be considered as well.

  For example, if firewall and load balancer gets packets go through NAT service, they need to know the header mapping information that the NAT have set to correctly process their functions. If the FG consists of IPS followed by DPI and those functions are connected different switches, the switch connecting DPI must know if the incomming packets should be forwarded to DPI or not. Port knocking is also well-known example of stateful function.

  To mitigate the situation, the states of all VNFs having behavioral dependency must be considered when they are verified.

6. Gap analysis - open source projects

Recently, the Open Platform for NFV (OPNFV) community is collaborating on a carrier-grade, integrated, open source platform to

accelerate the introduction of new NFV products and services [OPNFV]. Open Daylight (ODL) is also being tightly coupled with this OPNFV platform to integrate SDN controller into NFV framework [ODL].

This clause analyzes the existing open source projects including OPNFV and ODL related to verification of NFV services.

6.1. OPNFV

6.1.1. Doctor

The Doctor project provides a NFVI fault management and maintenance framework on top of the virtualized infrastructure. The key feature is to notify unavailability of virtualized resources and to recover unavailable VNFs.

While the Doctor project focuses only on faults in NFVI including compute, network, and storage resources, the document discusses broader fault management issues such as break-down of the supporting infrastructure due to incomplete or inconsistent configuration of NFV services.

6.1.2. Moon

The Moon project implements a security management system for the cloud computing infrastructure. The project also enforces the security managers through various mechanisms, e.g., authorization for access control, firewall for networking, isolation for storage, and logging for tractability.

Note that the main interest of the Moon project is the DDoS attack to a service node and the IDS management for VNFs. A wider range of security issues in the NFV service verification need to be discussed.

6.1.3 Bottlenecks

The Bottlenecks project aims to find system bottlenecks by testing and verifying OPNFV infrastructure in a staging environment before committing it to a production environment. Instead of debugging the deployment in production environment, an automatic method for executing benchmarks to validate the deployment during staging is adopted. For example, the system measures the performance of each VNF by generating workload on VNFs. The Bottlenecks project does not consider incomplete or inconsistent configurations on NFV services that might cause the system bottlenecks. Furthermore, the Bottlenecks project aims to find system bottlenecks before committing it to a production environment. Meanwhile, the draft also considers how to

find bottlenecks in real time.

6.1.4 VSPerf

The VSPerf projects provides an automated testing framework and comprehensive test suite based on industry standards for measuring data-plane performance. The architecture of VSPerf is agnostic to switch and traffic generator and test scenarios can be customized.

The VSPerf can be used for developing switching technologies as well as for evaluation and optimization of the data-path performance.

6.2. ODL

6.2.1. Network Intent Composition

The Network Intent Composition project enables the controller to manage and direct network services and network resources based on intent for network behaviors and network policies. Intents are described to the controller through a new northbound interface, which provides generalized and abstracted policy semantics. Also, the Network Intent Composition project aims to provide advanced composition logic for identifying and resolving intent conflicts across the network applications.

When the reconfiguration upon the policy (i.e, intent) is delayed, policy inconsistency in service nodes may occur after the policy is applied to service nodes. While the Network Intent Composition project resolves such intent conflicts only before they are translated into service nodes, this document covers intent conflicts and inconsistency issues in a broader sense.

6.2.2. Controller Shield

The Controller Shield project proposes to create a repository called unified-security plugin (USecPlugin). The unified-security plugin is a general purpose plugin to provide the controller security information to northbound applications. The security information could be for various purposes such as collating source of different attacks reported in southbound plugins and suspected controller intrusions. Information collected at this plugin can also be used to configure firewalls and create IP blacklists for the network.

In terms of security services, the document covers authentication, data integrity, confidentiality, and replay protection. However, the Controller Shield project only covers authentication, data integrity, and replay protection services where the confidentiality service is not considered.

6.2.3. Defense4All

The Defense4All project proposes a SDN application for detecting and mitigating DDoS attacks. The application communicates with ODL controller via the northbound interface and performs the two main tasks; 1) Monitoring behavior of protected traffic and 2) Diverting attacked traffic to selected attack mitigation systems (AMSs).

While the Defense4All project only focuses on defense system at the controller, this document includes broader defense issues at the service node as well as the controller.

6.3. Summary

The verification functions should spread over the platforms to accomplish the requirements mentioned in clause 3. The correctness of NFV- based services and their network configurations can be checked in the NFV MANO layer which has the entire states of the VNFs. Each NFVI needs to provide verification layer which composed of policy manager, network database and interfaces (e.g. REST APIs). Local properties and invariants can be verified inside the specific NFVI, and the global properties and invariants can be checked by merging local verification results from the related NFVIs.

The verification service provides verification functions to NFV MANO, NFVI, and any other low-level modules such as SDN controllers. For the platform independency, it provides standard APIs to process the verification tasks. It also uses standard APIs provided by OSS such as OpenStack (Neutron) and Open Daylight. The compiler and interpreter translate standard description languages and protocols into the internal model which optimized to the verification tasks. It can process user-defined properties to be checked as well. The properties to be checked whether they are user-defined or pre-defined invariants are managed by property library. The verifier maintains a set of verification algorithms to check the properties. The network database inside the verification service manages the global network states directly or indirectly.

A PoC can be implemented using OpenStack (Neutron) and Open Daylight. The modules related to verification framework can reside in between network virtualization framework (e.g. OpenStack Neutron) and SDN controller (e.g. Open Daylight). Neutron and Open Daylight uses standard APIs provided by verification service to accomplish verification tasks. The initial use case for the PoC could be, in particular, any of security, performance, etc as mentioned in clause 2.

7. Security Considerations

As already described in clause 2.6, how to verify security holes in VNF FG is very important consideration. In terms of security services, authentication, data integrity, confidentiality, and replay protection should be provided. On the other hand, potential security concern should be also carefully checked since several VNFs (e.g., NAT) can modify or update packet headers and payload.

8. Acknowledgements

The authors would like to thank formal methods lab members in Korea University for their verification theory support. Also thank Jose Saldana for his useful comments.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

9.2. Informative References

[ETSI-NFV-Arch] ETSI, "Network Function Virtualisation (NFV); Architectural Framework," 2014.

[ETSI-NFV-MANO] ETSI, "Network Function Virtualiztion (NFV) Management and Orchestration," 2014.

[ETSI-NFV-Testing] ETSI, "Network Function Virtualiztion (NFV) Pre-deployment Testing; Report on Validation of NFV Environments and Services," 2016.

[SIGCOMM-Qazi] Z. Qazi, C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "SIMPLE-fying Middlebox Policy Enforcement Using SDN," in Proc. ACM SIGCOMM 2013, August 2013.

[ONS-Gember] A. Gember, R. Grandl, A. Anand, T. Benson, and A. Akella, "Stratos: Virtual Middleboxes as First-Class Entities," ONS 2013 and TR.

[SIGCOMM-Gember] A. Gember, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "OpenNF: Enabling

Innovation in Network Function Control," in Proc. ACM SIGCOMM 2014, August 2014.

[HOTSDN-Ghorbani] S. Ghorbani, B. Godfrey, "Towards Correct Network Virtualization", HOTSDN 2014.

[RAFT] https://raftconsensus.github.io/.

[ODL] "OpenDaylight SDN Controller, "http://www.opendaylight.org/

[ONOS] "Open Network Operating System, "http://onosproject.org/

[OPNFV] "Open Platform for NFV, "https://www.opnfv.org/

Authors' Addresses

Myung-Ki Shin
ETRI
161 Gajeong-dong Yuseng-gu
Daejeon, 305-700
Korea

Phone: +82 42 860 4847
Email: mkshin@etri.re.kr

Ki-Hyuk Nam
Friesty

Email: nam@friesty.com

Sangheon Pack
Korea University

Email: shpack@korea.ac.kr

Seungik Lee
ETRI
161 Gajeong-dong Yuseng-gu
Daejeon, 305-700
Korea

Phone: +82 42 860 1483
Email: seungiklee@etri.re.kr

Ramki Krishnan
Dell

Email: Ramki_Krishnan@dell.com

NFVRG
Internet-Draft
Intended status: Informational
Expires: September 22, 2016

R. Szabo, Ed.
Z. Qiang
Ericsson
M. Kind
Deutsche Telekom AG
March 21, 2016

Recursive virtualization and programming for network and cloud resources
draft-irtf-nfvrg-unify-recursive-programming-00

Abstract

The introduction of Network Function Virtualization (NFV) in carrier-grade networks promises improved operations in terms of flexibility, efficiency, and manageability. NFV is an approach to combine network and compute virtualizations together. However, network and compute resource domains expose different virtualizations and programmable interfaces. In [I-D.unify-nfvrg-challenges] we argued for a joint compute and network virtualization by looking into different compute abstractions.

In this document we analyze different approaches to orchestrate a service graph with transparent network functions relying on a public telecommunication network and ending in a commodity data center. We show that a recursive compute and network joint virtualization and programming has clear advantages compared to other approaches with separated control between compute and network resources. In addition, the joint virtualization will have cost and performance advantages by removing additional virtualization overhead. The discussion of the problems and the proposed solution is generic for any data center use case; however, we use NFV as an example.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 22, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 2
2. Terms and Definitions 3
3. Use Cases 4
3.1. Black Box DC 4
3.1.1. Black Box DC with L3 tunnels 5
3.1.2. Black Box DC with external steering 6
3.2. White Box DC 8
3.3. Conclusions 9
4. Recursive approach 10
4.1. Virtualization 11
4.1.1. The virtualizer's data model 13
5. Relation to ETSI NFV 24
5.1. Policy based resource management 27
6. Examples 29
6.1. Infrastructure reports 29
6.2. Simple requests 35
7. Experimentations 37
8. IANA Considerations 38
9. Security Considerations 38
10. Acknowledgement 38
11. Informative References 38
Authors' Addresses 39

1. Introduction

To a large degree there is agreement in the research community that rigid network control limits the flexibility of service creation. In [I-D.unify-nfvrg-challenges]

- we analyzed different compute domain abstractions to argue that joint compute and network virtualization and programming is needed for efficient combination of these resource domains;

- we described challenges associated with the combined handling of compute and network resources for a unified production environment.

Our goal here is to analyze different approaches to instantiate a service graph with transparent network functions into a commodity Data Center (DC). More specifically, we analyze

- two black box DC set-ups, where the intra-DC network control is limited to some generic compute only control programming interface;

- a white box DC set-up, where the intra-DC network control is exposed directly to for a DC external control to coordinate forwarding configurations;

- a recursive approach, which illustrates potential benefits of a joint compute and network virtualization and control.

The discussion of the problems and the proposed solution is generic for any data center use case; however, we use NFV as an example.

2. Terms and Definitions

We use the terms compute and "compute and storage" interchangeably throughout the document. Moreover, we use the following definitions, as established in [ETSI-NFV-Arch]:

NFV: Network Function Virtualization - The principle of separating network functions from the hardware they run on by using virtual hardware abstraction.

NFVI: NFV Infrastructure - Any combination of virtualized compute, storage and network resources.

VNF: Virtualized Network Function - a software-based network function.

MANO: Management and Orchestration - In the ETSI NFV framework [ETSI-NFV-MANO], this is the global entity responsible for management and orchestration of NFV lifecycle.

Further, we make use of the following terms:

NF: a network function, either software-based (VNF) or appliance-based.

SW: a (routing/switching) network element with a programmable control plane interface.

DC: a data center is an interconnection of Compute Nodes (see below) with a data center controller, which offers programmatic resource control interface to its clients.

CN: a server, which is controlled by a DC control plane and provides execution environment for virtual machine (VM) images such as VNFs.

3. Use Cases

Service Function Chaining (SFC) looks into the problem how to deliver end-to-end services through the chain of network functions (NFs). Many of such NFs are envisioned to be transparent to the client, i.e., they intercept the client connection for adding value to the services without the knowledge of the client. However, deploying network function chains in DCs with Virtualized Network Functions (VNFs) are far from trivial [I-D.ietf-sfc-dc-use-cases]. For example, different exposures of the internals of the DC will imply different dynamisms in operations, different orchestration complexities and may yield for different business cases with regards to infrastructure sharing.

We investigate different scenarios with a simple NF forwarding graph of three VNFs (o->VNF1->VNF2->VNF3->o), where all VNFs are deployed within the same DC. We assume that the DC is a multi-tier leaf and spine (CLOS) and that all VNFs of the forwarding graph are bump-in-the-wire NFs, i.e., the client cannot explicitly access them.

3.1. Black Box DC

In Black Bock DC set-ups, we assume that the compute domain is an autonomous domain with legacy (e.g., OpenStack) orchestration APIs. Due to the lack of direct forwarding control within the DC, no native L2 forwarding can be used to insert VNFs running in the DC into the forwarding graph. Instead, explicit tunnels (e.g., VxLAN) must be used, which need termination support within the deployed VNFs. Therefore, VNFs must be aware of the previous and the next hops of the forwarding graph to receive and forward packets accordingly.

3.1.1. Black Box DC with L3 tunnels

Figure 1 illustrates a set-up where an external VxLAN termination point in the SDN domain is used to forward packets to the first NF (VNF1) of the chain within the DC. VNF1, in turn, is configured to forward packets to the next SF (VNF2) in the chain and so forth with VNF2 and VNF3.

In this set-up VNFs must be capable of handling L3 tunnels (e.g., VxLAN) and must act as forwarders themselves. Additionally, an operational L3 underlay must be present so that VNFs can address each other.

Furthermore, VNFs holding chain forwarding information could be untrusted user plane functions from 3rd party developers. Enforcement of proper forwarding is problematic.

Additionally, compute only orchestration might result in sub-optimal allocation of the VNFs with regards to the forwarding overlay, for example, see back-forth use of a core switch in Figure 1.

In [I-D.unify-nfvrg-challenges] we also pointed out that within a single Compute Node (CN) similar VNF placement and overlay optimization problem may reappear in the context of network interface cards and CPU cores.

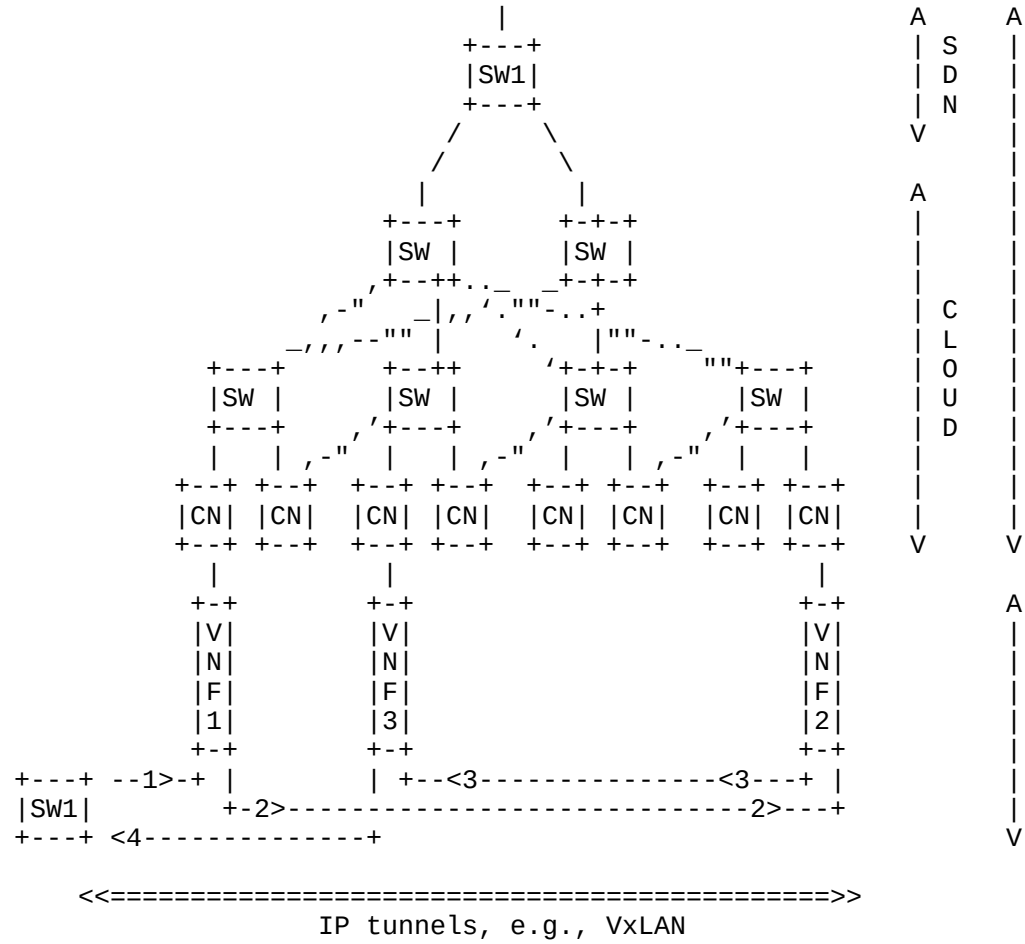


Figure 1: Black Box Data Center with VNF Overlay

3.1.2. Black Box DC with external steering

Figure 2 illustrates a set-up where an external VxLAN termination point in the SDN domain is used to forward packets among all the SFs (VNF1-VNF3) of the chain within the DC. VNFs in the DC need to be configured to receive and send packets between only the SDN endpoint, hence are not aware of the next hop VNF address. Shall any VNFs need to be relocated, e.g., due to scale in/out as described in [I-D.zu-nfvrg-elasticity-vnf], the forwarding overlay can be transparently re-configured at the SDN domain.

Note however, that traffic between the DC internal SFs (VNF1, VNF2, VNF3) need to exit and re-enter the DC through the external SDN switch. This, certainly, is sub-optimal an results in ping-pong traffic similar to the local and remote DC case discussed in [I-D.zu-nfvrg-elasticity-vnf].

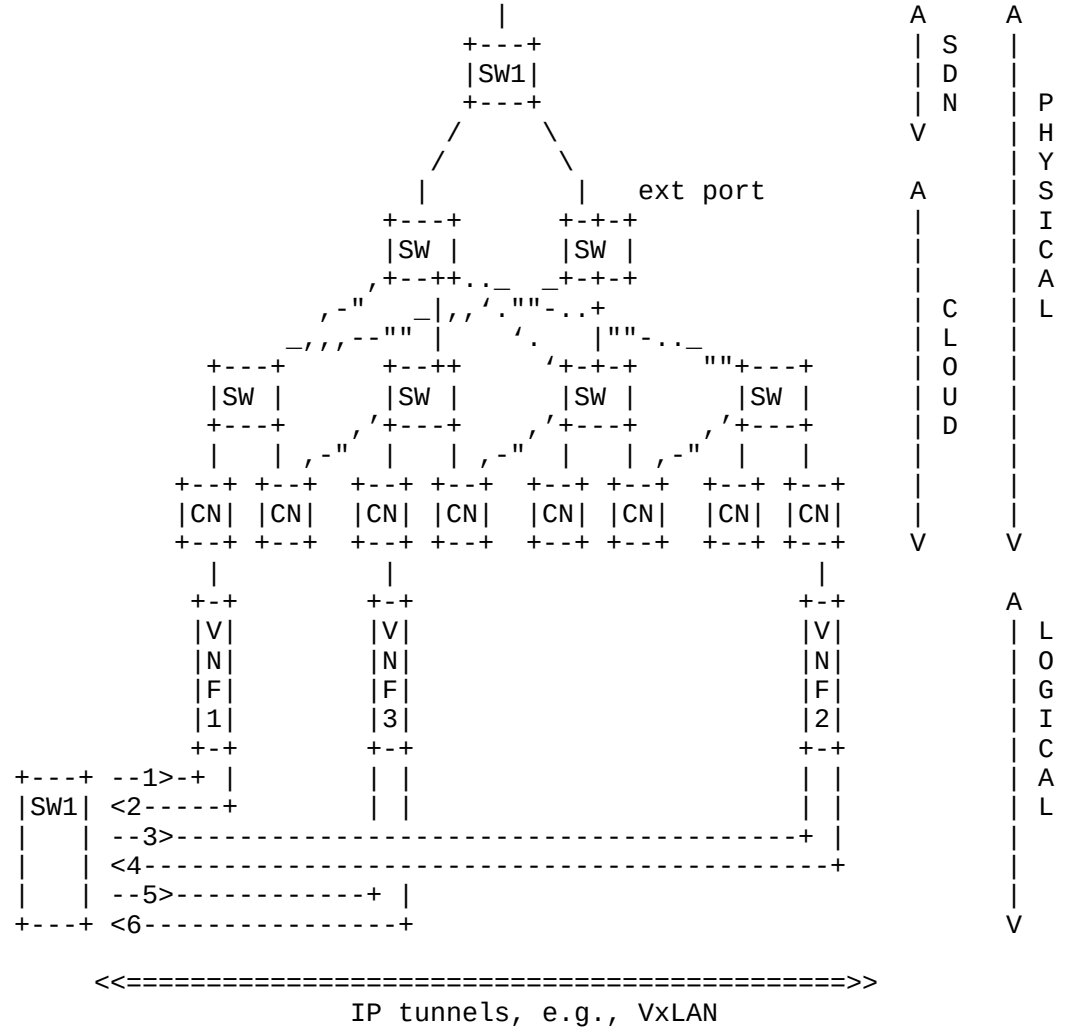


Figure 2: Black Box Data Center with ext Overlay

3.2. White Box DC

Figure 3 illustrates a set-up where the internal network of the DC is exposed in full details through an SDN Controller for steering control. We assume that native L2 forwarding can be applied all through the DC until the VNFs' port, hence IP tunneling and tunnel termination at the VNFs are not needed. Therefore, VNFs need not be forwarding graph aware but transparently receive and forward packets. However, the implications are that the network control of the DC must be handed over to an external forwarding controller (see that the SDN domain and the DC domain overlaps in Figure 3). This most probably prohibits clear operational separation or separate ownerships of the two domains.

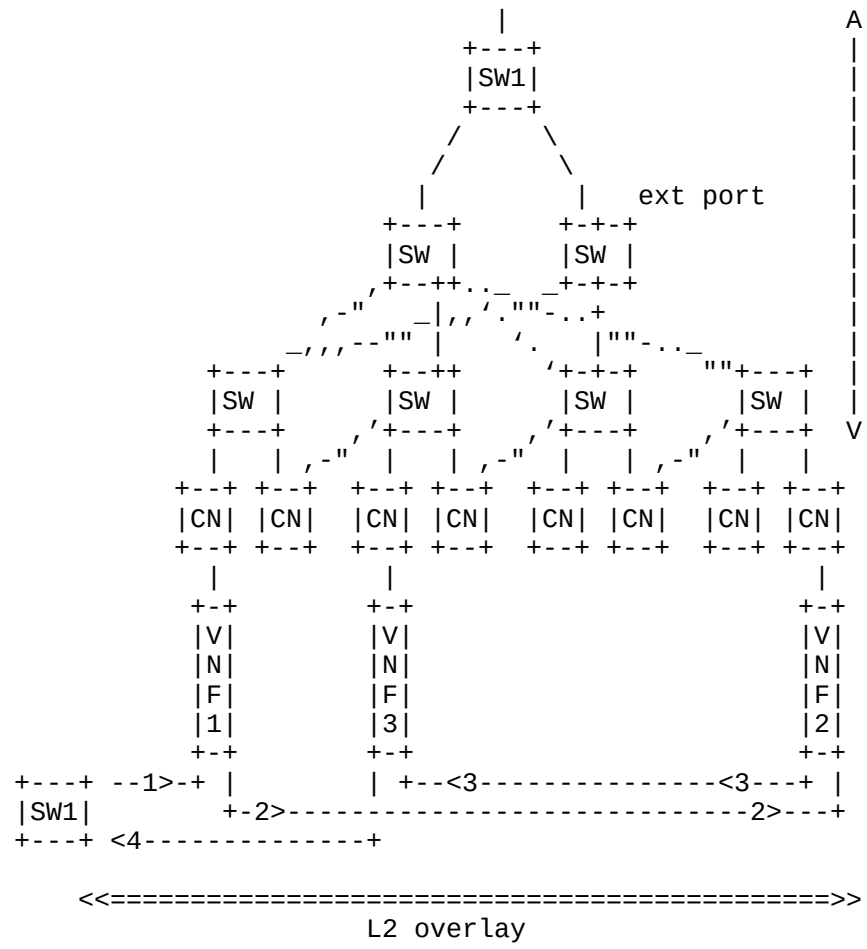


Figure 3: White Box Data Center with L2 Overlay

3.3. Conclusions

We have shown that the different solutions imply different operation and management actions. From network operations point of view, it is not desirable to run and manage similar functions several times (L3 blackbox DC case) - especially if the networking overlay can be easily managed upfront by using a programmatic interface, like with the external steering in black and whitebox DC scenarios.

4. Recursive approach

We argued in [I-D.unify-nfvrg-challenges] and [I-D.caszpe-nfvrg-orchestration-challenges] for a joint software and network programming interface. Consider that such joint software and network abstraction (virtualization) exists around the DC with a corresponding resource programmatic interface. A software and network programming interface could include VNF requests and the definition of the corresponding network overlay. However, such programming interface is similar to the top level services definition, for example, by the means of a VNF Forwarding Graph.

Figure 4 illustrates a joint domain virtualization and programming setup. In Figure 4 "[x]" denotes ports of the virtualized data plane while "x" denotes port created dynamically as part of the VNF deployment request. Over the joint software and network virtualization VNF placement and the corresponding traffic steering could be defined in an atomic, which is orchestrated, split and handled to the next levels (see Figure 5) in the hierarchy for further orchestration. Such setup allows clear operational separation, arbitrary domain virtualization (e.g., topology details could be omitted) and constraint based optimization of domain wide resources.

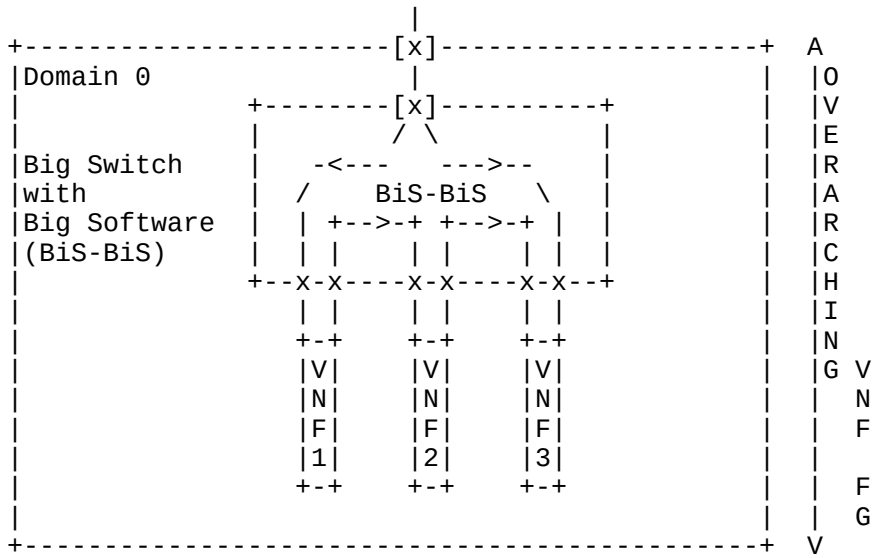


Figure 4: Recursive Domain Virtualization and Joint VNF FG programming: Overarching View

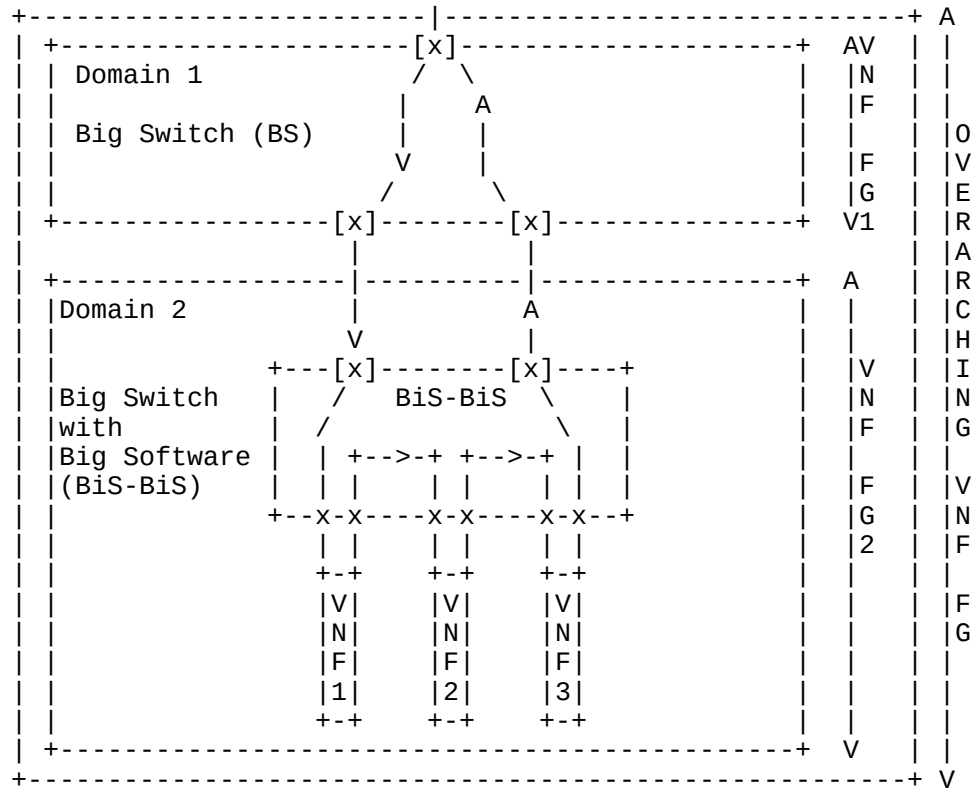


Figure 5: Recursive Domain Virtualization and Joint VNF FG programming: Domain Views

4.1. Virtualization

Let us first define the joint software and network abstraction (virtualization) as a Big Switch with Big Software (BiS-BiS). A BiS-BiS is a node abstraction, which incorporates both software and networking resources with an associated joint software and network control API (see Figure 6).

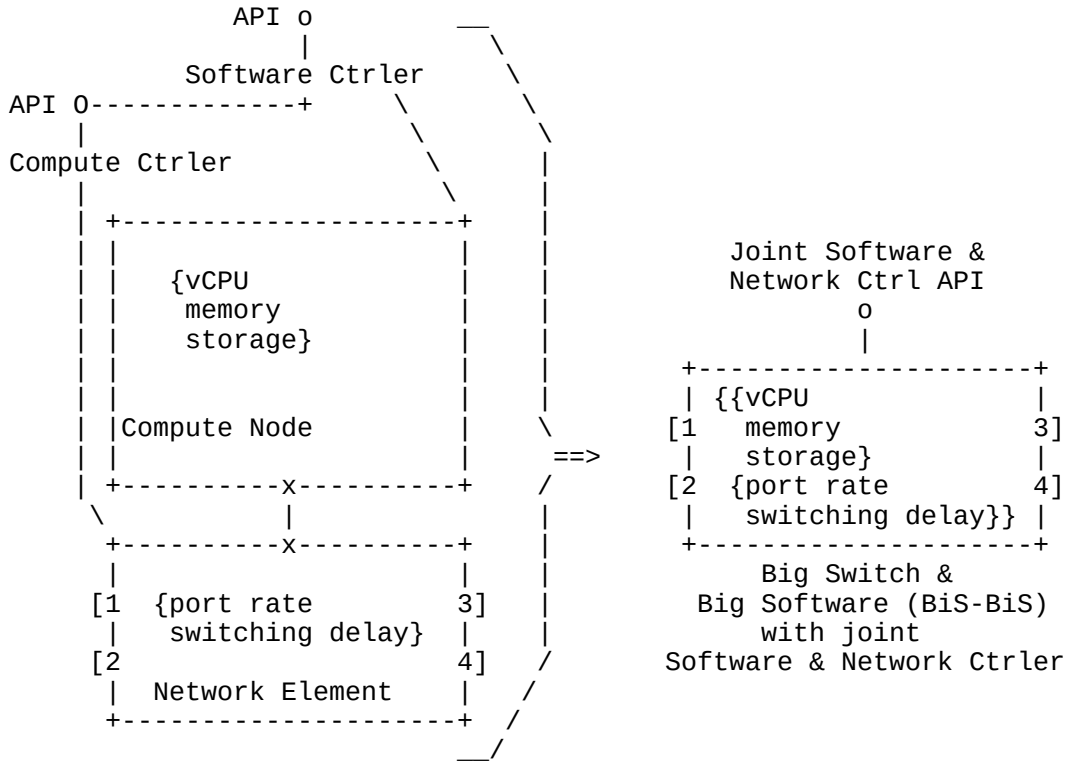


Figure 6: Big Switch with Big Software definition

The configuration over a BiS-BiS allows the atomic definition of NF placements and the corresponding forwarding overlay as a Network Function - Forwarding Graph (NF-FG). The embedment of NFs into a BiS-BiS allows the inclusion of NF ports into the forwarding overlay definition (see ports a, b, ...,f in Figure 7). Ports 1,2, ..., 4 are seen as infrastructure ports while NF ports are created and destroyed with NF placements.

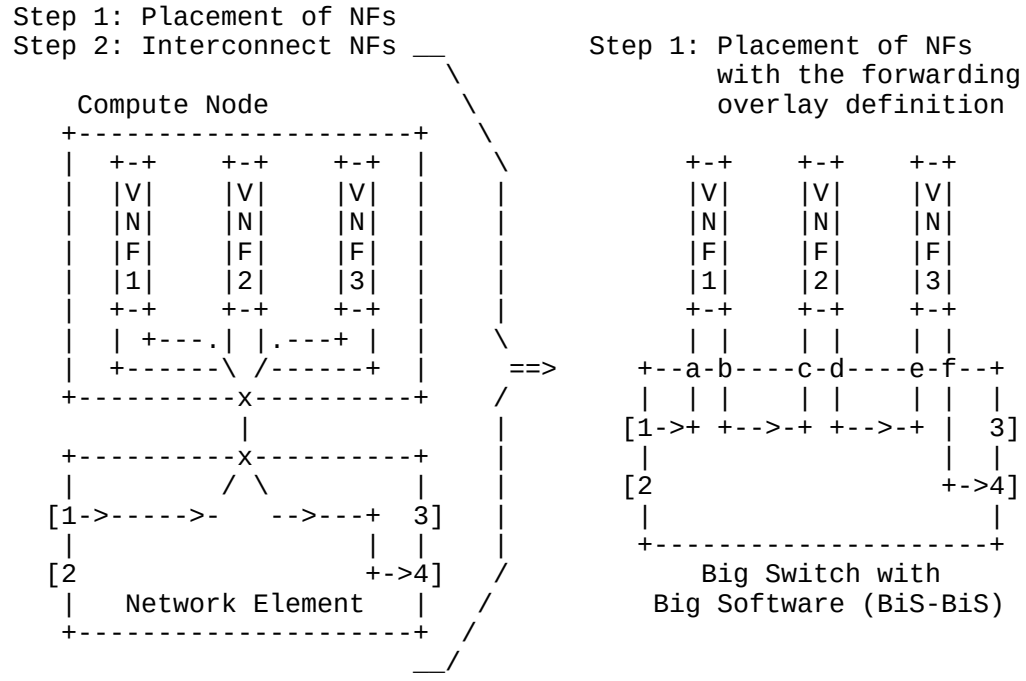


Figure 7: Big Switch with Big Software definition with a Network Function - Forwarding Graph (NF-FG)

4.1.1. The virtualizer's data model

4.1.1.1. Tree view

```
module: virtualizer
   +--rw virtualizer
      +--rw id          string
      +--rw name?       string
      +--rw nodes
      |  +--rw node* [id]
      |     +--rw id              string
      |     +--rw name?           string
      |     +--rw type            string
      |     +--rw ports
      |     |  +--rw port* [id]
      |     |     +--rw id            string
      |     |     +--rw name?         string
      |     |     +--rw port_type?    string
      |     |     +--rw capability?   string
      |     |     +--rw sap?          string
      |     |     +--rw sap_data
```

```
|     |     |  +--rw technology?   string
|     |     |  +--rw resources
|     |     |     +--rw delay?       string
|     |     |     +--rw bandwidth?   string
|     |     |     +--rw cost?        string
|     |     +--rw control
|     |     |  +--rw controller?     string
|     |     |  +--rw orchestrator?   string
|     |     +--rw addresses
|     |     |  +--rw l2?   string
|     |     |  +--rw l3* [id]
|     |     |  |  +--rw id           string
|     |     |  |  +--rw name?        string
|     |     |  |  +--rw configure?   string
|     |     |  |  +--rw client?      string
|     |     |  |  +--rw requested?   string
|     |     |  |  +--rw provided?    string
|     |     |  +--rw l4?   string
|     |     +--rw metadata* [key]
|     |        +--rw key      string
|     |        +--rw value?   string
|     +--rw links
|     |  +--rw link* [id]
|     |     +--rw id           string
|     |     +--rw name?        string
|     |     +--rw src?         ->
|     |     +--rw dst?         ->
|     |     +--rw resources
|     |        +--rw delay?       string
|     |        +--rw bandwidth?   string
|     |        +--rw cost?        string
|     +--rw resources
|     |  +--rw cpu        string
|     |  +--rw mem        string
|     |  +--rw storage    string
|     |  +--rw cost?      string
|     +--rw metadata* [key]
|     |  +--rw key      string
|     |  +--rw value?   string
|     +--rw NF_instances
|     |  +--rw node* [id]
|     |     +--rw id           string
|     |     +--rw name?        string
|     |     +--rw type?        string
|     |     +--rw ports
|     |     |  +--rw port* [id]
|     |     |     +--rw id            string
|     |     |     +--rw name?         string
```

```
|     |     |     +--rw port_type?    string
|     |     |     +--rw capability?   string
|     |     |     +--rw sap?          string
|     |     |     +--rw sap_data
|     |     |     |  +--rw technology?   string
|     |     |     |  +--rw resources
|     |     |     |     +--rw delay?       string
|     |     |     |     +--rw bandwidth?   string
|     |     |     |     +--rw cost?        string
|     |     |     +--rw control
|     |     |     |  +--rw controller?     string
|     |     |     |  +--rw orchestrator?   string
|     |     |     +--rw addresses
|     |     |     |  +--rw l2?   string
|     |     |     |  +--rw l3* [id]
|     |     |     |  |  +--rw id           string
|     |     |     |  |  +--rw name?        string
|     |     |     |  |  +--rw configure?   string
|     |     |     |  |  +--rw client?      string
|     |     |     |  |  +--rw requested?   string
|     |     |     |  |  +--rw provided?    string
|     |     |     |  +--rw l4?   string
|     |     |     +--rw metadata* [key]
|     |     |        +--rw key      string
|     |     |        +--rw value?   string
|     |     +--rw links
|     |     |  +--rw link* [id]
|     |     |     +--rw id           string
|     |     |     +--rw name?        string
|     |     |     +--rw src?         ->
|     |     |     +--rw dst?         ->
|     |     |     +--rw resources
|     |     |        +--rw delay?       string
|     |     |        +--rw bandwidth?   string
|     |     |        +--rw cost?        string
|     |     +--rw resources
|     |     |  +--rw cpu        string
|     |     |  +--rw mem        string
|     |     |  +--rw storage    string
|     |     |  +--rw cost?      string
|     |     +--rw metadata* [key]
|     |        +--rw key      string
|     |        +--rw value?   string
|     +--rw capabilities
|     |  +--rw supported_NFs
|     |     +--rw node* [id]
|     |        +--rw id           string
|     |        +--rw name?        string
```

```
|      |        +--rw type?        string
|      |        +--rw ports
|      |        |  +--rw port* [id]
|      |        |     +--rw id            string
|      |        |     +--rw name?         string
|      |        |     +--rw port_type?    string
|      |        |     +--rw capability?   string
|      |        |     +--rw sap?          string
|      |        |     +--rw sap_data
|      |        |     |  +--rw technology?   string
|      |        |     |  +--rw resources
|      |        |     |     +--rw delay?       string
|      |        |     |     +--rw bandwidth?   string
|      |        |     |     +--rw cost?        string
|      |        |     +--rw control
|      |        |     |  +--rw controller?     string
|      |        |     |  +--rw orchestrator?   string
|      |        |     +--rw addresses
|      |        |     |  +--rw l2?   string
|      |        |     |  +--rw l3* [id]
|      |        |     |  |  +--rw id           string
|      |        |     |  |  +--rw name?        string
|      |        |     |  |  +--rw configure?   string
|      |        |     |  |  +--rw client?      string
|      |        |     |  |  +--rw requested?   string
|      |        |     |  |  +--rw provided?    string
|      |        |     |  +--rw l4?   string
|      |        |     +--rw metadata* [key]
|      |        |        +--rw key      string
|      |        |        +--rw value?   string
|      |        +--rw links
|      |        |  +--rw link* [id]
|      |        |     +--rw id           string
|      |        |     +--rw name?        string
|      |        |     +--rw src?         ->
|      |        |     +--rw dst?         ->
|      |        |     +--rw resources
|      |        |        +--rw delay?       string
|      |        |        +--rw bandwidth?   string
|      |        |        +--rw cost?        string
|      |        +--rw resources
|      |        |  +--rw cpu        string
|      |        |  +--rw mem        string
|      |        |  +--rw storage    string
|      |        |  +--rw cost?      string
|      |        +--rw metadata* [key]
|      |           +--rw key      string
|      |           +--rw value?   string
```

```
|     +--rw flowtable
|        +--rw flowentry* [id]
|           +--rw id           string
|           +--rw name?        string
|           +--rw priority?    string
|           +--rw port         ->
|           +--rw match        string
|           +--rw action       string
|           +--rw out?         ->
|           +--rw resources
|              +--rw delay?       string
|              +--rw bandwidth?   string
|              +--rw cost?        string
+--rw links
|  +--rw link* [id]
|     +--rw id           string
|     +--rw name?        string
|     +--rw src?         ->
|     +--rw dst?         ->
|     +--rw resources
|        +--rw delay?       string
|        +--rw bandwidth?   string
|        +--rw cost?        string
+--rw metadata* [key]
|  +--rw key      string
|  +--rw value?   string
+--rw version?    string
```

Figure 8: Virtualizer’s YANG data model: tree view

4.1.1.2. YANG Module

```
<CODE BEGINS> file "virtualizer.yang"
module virtualizer {
  namespace "urn:unify:virtualizer";
  prefix "virtualizer";
  organization "ETH";
  contact "Robert Szabo <robert.szabo@ericsson.com>";

  revision "2016-02-24" {
    description "V5.0: Common port configuration were added to the yang model fr
om the metadata fields";
  }

  revision "2016-02-19" {
    description "Added port/control (for Cf-Or interface); port/resources; link-
resources/cost and sofware-resource/cost for administrative metric; clarificatio
ns for port/capability";
  }

  revision "2016-01-28" {
```

```
    description "Metadata added to infra_node and virtualizer level; Virtualizer
’s revised data model based on virtualizer3; changes: link key is set to id";
  }

  //======================== REUSABLE GROUPS ================================

  grouping id-name {
    leaf id { type string; }
    leaf name { type string;}
  }

  grouping id-name-type {
    uses id-name;
    leaf type {
      type string;
      // for infrastructue view: mandatory true; --> refined in infrastrucutre v
iew
      mandatory false;
    }
  }

  grouping metadata {
    list metadata {
      min-elements 0;
      key key;
      leaf key{
        type string;
        mandatory true;
      }
      leaf value{
        type string;
        mandatory false;
      }
    }
  }

  grouping link-resource {
    leaf delay {
      type string;
      mandatory false;
    }
    leaf bandwidth {
      type string;
      mandatory false;
    }
    leaf cost {
      description "Administrative metric.";
      type string;
      mandatory false;
    }
```

```
  }

  grouping l3-address {
    uses id-name;
    leaf configure {
      description "True: this is a configuration request; False: this is fyi";
      type string;
    }
    leaf client {
      description "Configuration service support at the client: {'dhcp-client',
'pre-configured'}; if not present it is left to the infrastructure to deal with
it.";
      type string;
    }
    leaf requested {
      description "To request port configuration, options: {'public', 'ip/mask'}
, where public means the request of public IP address and private ip/mask a give
n address/mask configuration";
      type string;
    }
    leaf provided {
      description "The provided L3 configuration in response to the requested fi
eld.";
      type string;
    }
  }
  // ------------ PORTS -------
  grouping port {
    uses id-name;
    leaf port_type {
      description "{port-abstract, port-sap} port-sap is to represent UNIFY doma
in boundary; port-abstract is to represent UNIFY native port. Technology specifi
c attributes of a SAP is in the metadata.";
      type string;
    }
    leaf capability {
      description "To describe match and action capabilities associated with the
 port, e.g., match=port,tag,ip,tcp,udp,mpls,of1.0, where port: based forwarding;
 tag: unify abstract tagging; ip: ip address matching etc.";
      type string;
    }
    leaf sap {
      type string;
    }
    container sap_data {
      leaf technology {
        description "e.g., ('IEEE802.1q': '0x00c', 'MPLS': 70, 'IEEE802.1q')";
        type string;
      }
      container resources{
        description "Only used for domain boundary ports (port-sap type), where
this is used to derive interconnection link characteristics.";
        uses link-resource;
      }
    }
    container control {
      description "Used to connect this port to a UNIFY orchestrator's Cf-Or ref
erence point. Support controller - orchestrator or orchestrator - controller con
nection establishment.";
      leaf controller{
```

```
        description "URI of the local controller service at this NF, e.g., http:
//*:8080/cf-or/";
        type string;
      }
      leaf orchestrator{
        description "URI of the scoped orchestration service offered to this NF
specifically, e.g., http://192.168.1.100:8080/cf-or/";
        type string;
      }
    }
    container addresses {
      leaf l2 {
        description "Requested or provided";
        type string;
      }
      list l3 {
        key "id";
        uses l3-address;
      }
      leaf l4 {
        description "e.g., request: {tcp/22, tcp/8080}; response {tcp/22: (192.1
68.1.100, 1001)";
        type string;
      }
    }
    uses metadata;
  }

  // ------------ FLOW CONTROLS -------

  grouping flowentry {
    description "The flowentry syntax will follow ovs-ofctrl string format. The
UNIFY general tagging mechanism will be use like 'mpls'-> 'tag', i.e., push_tag:
tag; pop_tag:tag...";
    uses id-name;
    leaf priority {
      type string;
    }
    leaf port {
      type leafref {
        path "";
      }
      mandatory true;
    }
    leaf match {
      description "The match syntax will follow ovs-ofctrl string format with 'm
pls'->'tag', e.g.,: in_port=port, dl_tag=A, where port is the leafref above";
      type string;
      mandatory true;
    }
    leaf action {
      description "The action syntax will follow ovs-ofctrl string format with '
mpls'->'tag', e.g.,: push_tag:A, set_tag_label:A, output:out, where out is the l
eafref below";
      type string;
      mandatory true;
```

```
    }
    leaf out {
      type leafref {
        path "";
      }
    }
    container resources{
      uses link-resource;
    }

  }

  grouping flowtable {
    container flowtable {
      list flowentry {
        key "id";
        uses flowentry;
      }
    }
  }

  // ------------ LINKS  -------


  grouping link {
    uses id-name;
    leaf src {
      type leafref {
        path "";
      }
    }
    leaf dst {
      type leafref {
        path "";
      }
    }
    container resources{
      uses link-resource;
    }
  }

  grouping links {
    container links {
      list link {
        key "id";
        uses link;
      }
    }
```

```
  }

  // ---------- NODE ------------------

  grouping software-resource {
    leaf cpu {
      type string;
      mandatory true;
    }
    leaf mem {
      type string;
      mandatory true;
    }
    leaf storage {
      type string;
      mandatory true;
    }
    leaf cost {
      description "Administrative metric.";
      type string;
      mandatory false;
    }
  }

  grouping node {
    description "Any node: infrastructure or NFs";
    uses id-name-type;
    container ports {
      list port{
        key "id";
        uses port;
      }
    }
    uses links;
    container resources{
      uses software-resource;
    }
    uses metadata;
  }

  grouping nodes {
    list node{
      key "id";
      uses node;
    }
  }

  grouping infra-node { // they can contain other nodes (as NFs)
```

```
    uses node {
      refine type {
        mandatory true;
      }
    }
    container NF_instances {
      uses nodes;
    }
    container capabilities {
      container supported_NFs { // if supported NFs are enumerated
        uses nodes;
      }
    }
    uses flowtable;
  }



  //======================== NF-FG: Virtualizer and the Mapped request =========
========================

  container virtualizer {
    description "Container for a single virtualizer";
    uses id-name {
      refine id {
        mandatory true;
      }
    }
    container nodes{
      list node{ // infra nodes
        key "id";
        uses infra-node;
      }
    }
    uses links; // infra links
    uses metadata;
    leaf version {
      description "yang and virtualizer library version";
      type string;
    }
  }
}
<CODE ENDS>
```

Figure 9: Virtualizer’s YANG data model

5. Relation to ETSI NFV

According to the ETSI MANO framework [ETSI-NFV-MANO], an NFVO is split into two functions:

- The orchestration of NFVI resources across multiple VIMs, fulfilling the Resource Orchestration functions. The NFVO uses the Resource Orchestration functionality to provide services that support accessing NFVI resources in an abstracted manner independently of any VIMs, as well as governance of VNF instances sharing resources of the NFVI infrastructure

- The lifecycle management of Network Services, fulfilling the network Service Orchestration functions.

Similarly, a VIM is split into two functions:

- Orchestrating the allocation/upgrade/release/reclamation of NFVI resources (including the optimization of such resources usage), and

- managing the association of the virtualised resources to the physical compute, storage, networking resources.

The functional split is shown in Figure 14.

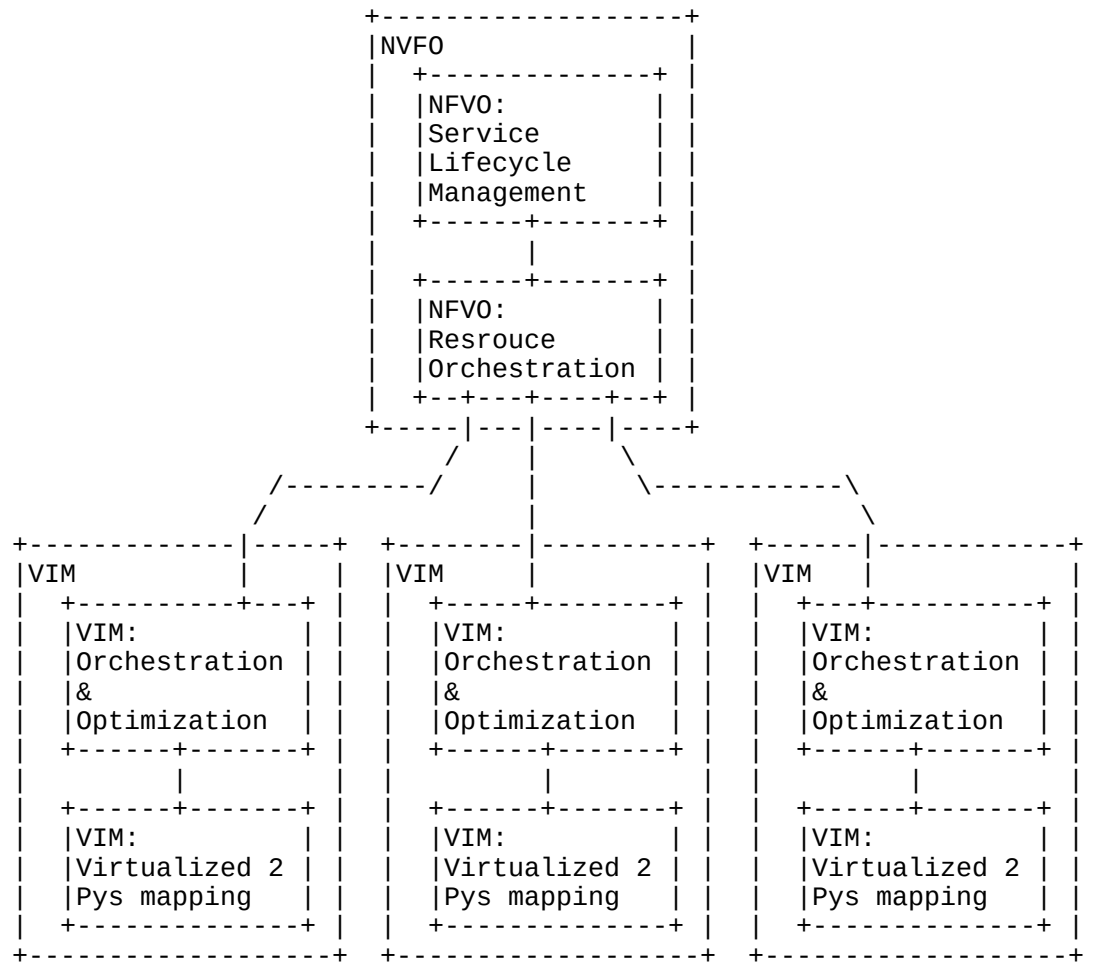


Figure 10: Functional decomposition of the NFVO and the VIM according to the ETSI MANO

If the Joint Software and Network Control API (Joint API) could be used between all the functional components working on the same abstraction, i.e., from the north of the VIM Virtualized to physical mapping component to the south of the NFVO: Service Lifecycle Management as shown in Figure 11, then a more flexible virtualization programming architecture could be created as shown in Figure 12.

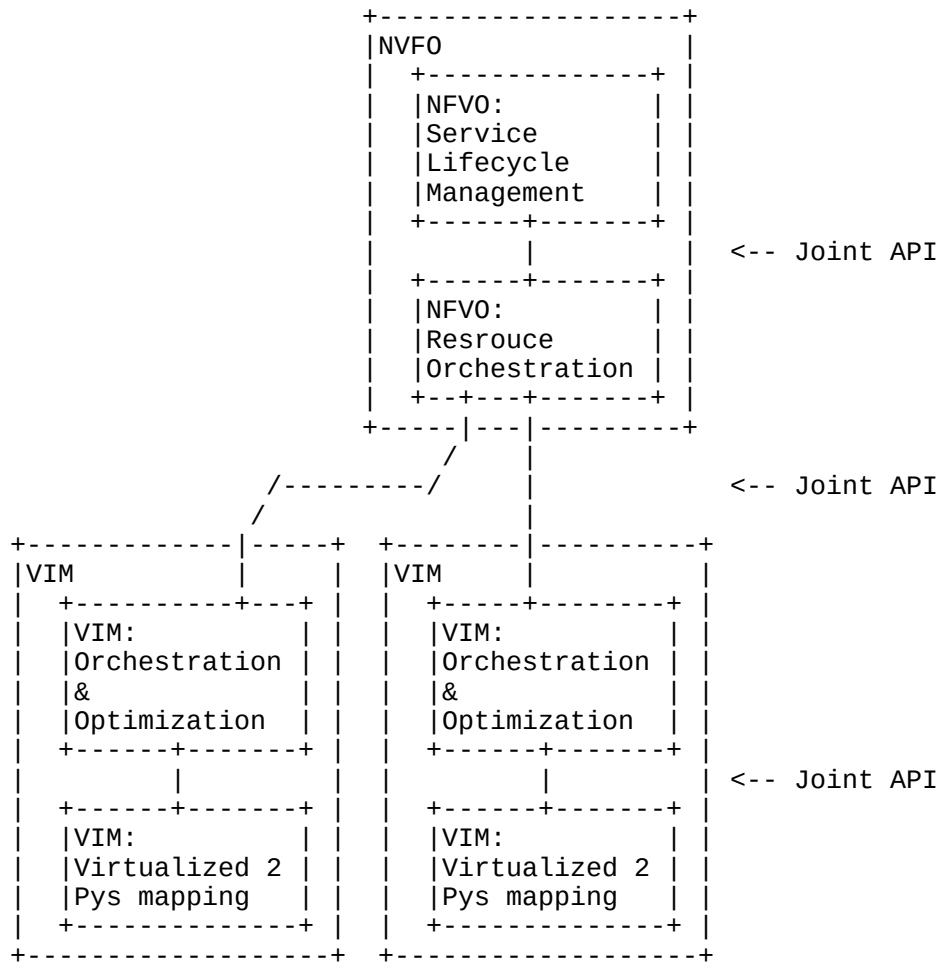


Figure 11: Functional decomposition of the NFVO and the VIM with the Joint Software and Network control API

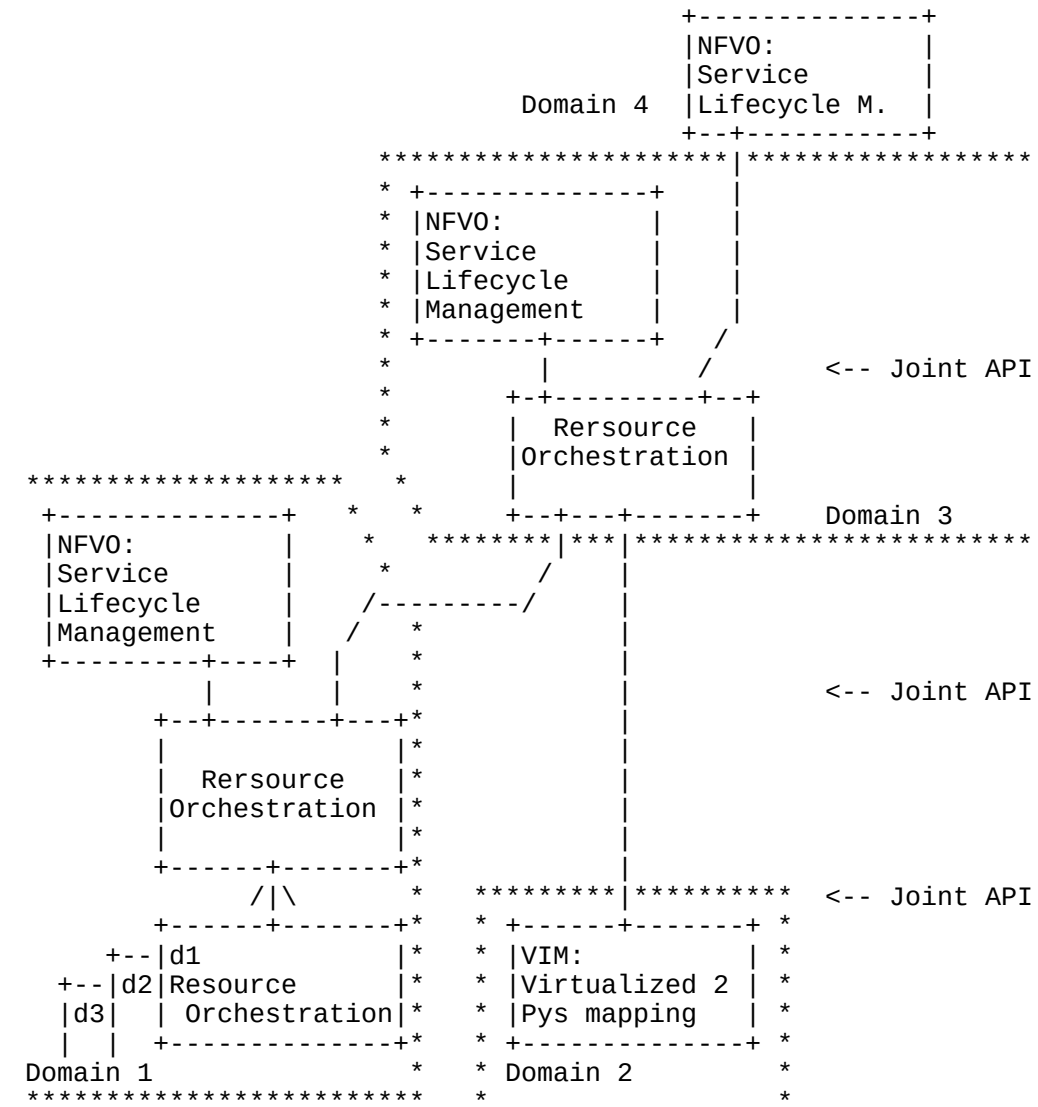


Figure 12: Joint Software and Network Control API: Recurring Flexible Architecture

5.1. Policy based resource management

In Figure 13 we show various policies mapped to the MANO architecture:

- Tenant Policies: Tenant policies exist whenever a domain offers a virtualization service to more than one consumer. User tenants may exists at the northbound of the NFVO. Additionally, if a VIM expose resource services to more than one NFVO, then each NFVO may appear as a tenant (virtualization consumer) at the northbound of the VIM.

- Wherever virtualization services are produced or consumed corresponding export and import policies may exist. Export policies govern the details of resources, capabilities, costs, etc. exposed to consumers. In turn, consumers (tenants) apply import policies to filter, tweak, annotate resources and services received from their southbound domains. An entity may at the same time consume and produce virtualization services hence apply both import and export policies.

- Operational policies support the business logic realized by the domain’s ownership. They are often associated with Operations or Business Support Systems (OSS or BSS) and frequently determine operational objectives like energy optimization, utilization targets, offered services, charing models, etc. Operational policies may be split according to different control plane layers, for example, i) lifecycle and ii) resource management layers within the NFVO.

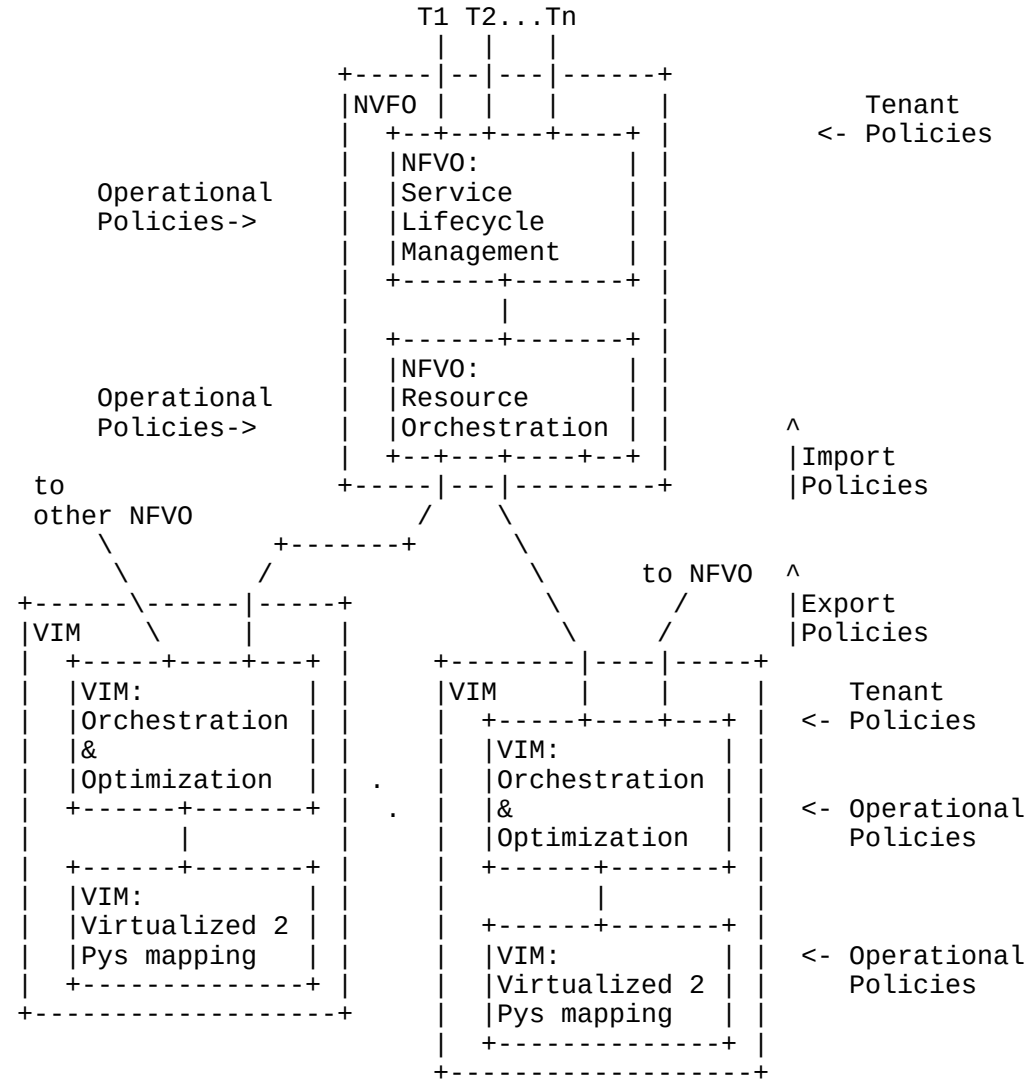


Figure 13: Policies within the MANO framework

6. Examples

6.1. Infrastructure reports

Figure 14 and Figure 15 show a single node infrastructure report. The example shows a BiS-BiS with two ports, out of which Port 0 is also a Service Access Point 0 (SAP0).

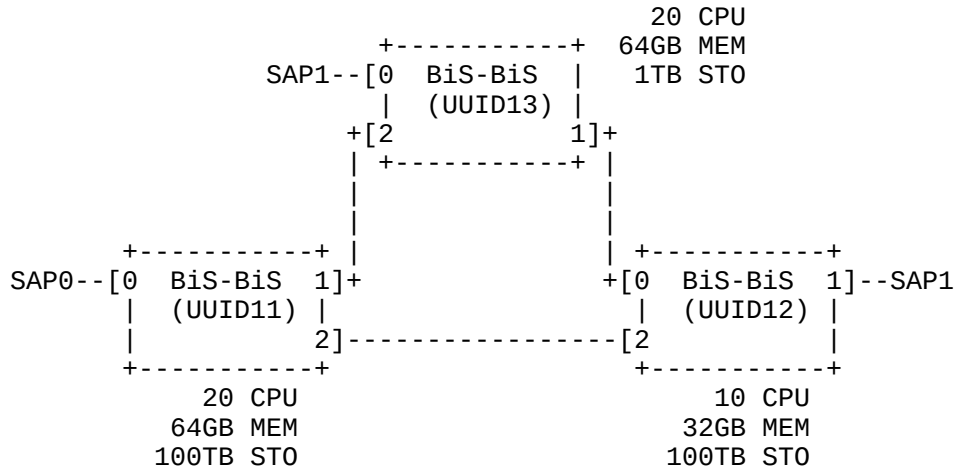


Figure 14: Single node infrastructure report example: Virtualization view

```
<virtualizer xmlns="http://fp7-unify.eu/framework/virtualizer">
    <id>UUID001</id>
    <name>Single node simple infrastructure report</name>
    <nodes>
        <node>
            <id>UUID11</id>
            <name>single Bis-Bis node</name>
            <type>BisBis</type>
            <ports>
                <port>
                    <id>0</id>
                    <name>SAP0 port</name>
                    <port_type>port-sap</port_type>
                    <vxlan>...</vxlan>
                </port>
                <port>
                    <id>1</id>
                    <name>North port</name>
                    <port_type>port-abstract</port_type>
                    <capability>...</capability>
                </port>
                <port>
                    <id>2</id>
                    <name>East port</name>
                    <port_type>port-abstract</port_type>
                    <capability>...</capability>
                </port>
            </ports>
            <resources>
                <cpu>20</cpu>
                <mem>64 GB</mem>
                <storage>100 TB</storage>
            </resources>
        </node>
    </nodes>
</virtualizer>
```

Figure 15: Single node infrastructure report example: xml view

Figure 16 and Figure 17 show a 3-node infrastructure report with 3 BiS-BiS nodes. Infrastructure links are inserted into the virtualization view between the ports of the BiS-BiS nodes.

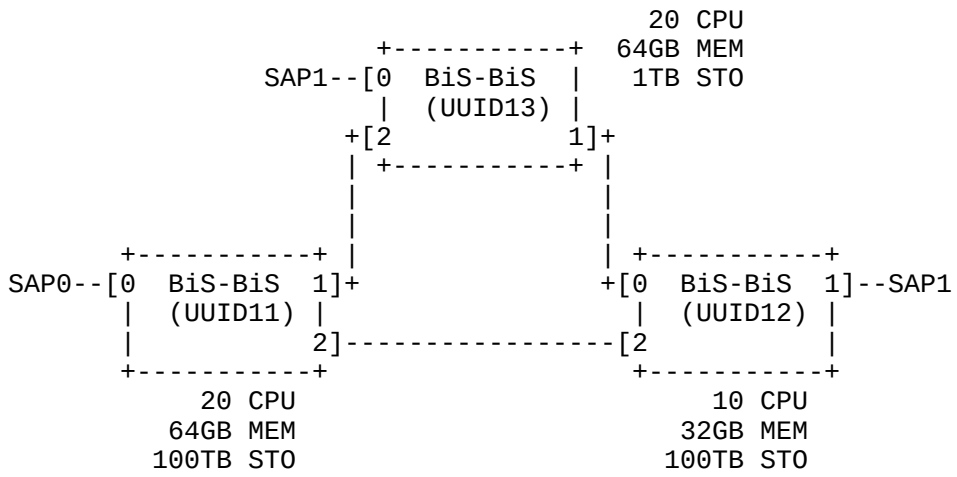


Figure 16: 3-node infrastructure report example: Virtualization view

```
 <virtualizer xmlns="http://fp7-unify.eu/framework/virtualizer">
     <id>UUID002</id>
     <name>3-node simple infrastructure report</name>
     <nodes>
         <node>
             <id>UUID11</id>
             <name>West Bis-Bis node</name>
             <type>BisBis</type>
             <ports>
                 <port>
                     <id>0</id>
                     <name>SAP0 port</name>
                     <port_type>port-sap</port_type>
                     <vxlan>...</vxlan>
                 </port>
                 <port>
                     <id>1</id>
                     <name>North port</name>
                     <port_type>port-abstract</port_type>
                     <capability>...</capability>
                 </port>
                 <port>
                     <id>2</id>
                     <name>East port</name>
                     <port_type>port-abstract</port_type>
                     <capability>...</capability>
                 </port>
             </ports>
             <resources>
```

```
                    <cpu>20</cpu>
                    <mem>64 GB</mem>
                    <storage>100 TB</storage>
                </resources>
            </node>
            <node>
                <id>UUID12</id>
                <name>East Bis-Bis node</name>
                <type>BisBis</type>
                <ports>
                    <port>
                        <id>1</id>
                        <name>SAP1 port</name>
                        <port_type>port-sap</port_type>
                        <vxlan>...</vxlan>
                    </port>
                    <port>
                        <id>0</id>
                        <name>North port</name>
                        <port_type>port-abstract</port_type>
                        <capability>...</capability>
                    </port>
                    <port>
                        <id>2</id>
                        <name>West port</name>
                        <port_type>port-abstract</port_type>
                        <capability>...</capability>
                    </port>
                </ports>
                <resources>
                    <cpu>10</cpu>
                    <mem>32 GB</mem>
                    <storage>100 TB</storage>
                </resources>
            </node>
            <node>
                <id>UUID13</id>
                <name>North Bis-Bis node</name>
                <type>BisBis</type>
                <ports>
                    <port>
                        <id>0</id>
                        <name>SAP2 port</name>
                        <port_type>port-sap</port_type>
                        <vxlan>...</vxlan>
                    </port>
                    <port>
                        <id>1</id>
```

```
                        <name>East port</name>
                        <port_type>port-abstract</port_type>
                        <capability>...</capability>
                    </port>
                    <port>
                        <id>2</id>
                        <name>West port</name>
                        <port_type>port-abstract</port_type>
                        <capability>...</capability>
                    </port>
                </ports>
                <resources>
                    <cpu>20</cpu>
                    <mem>64 GB</mem>
                    <storage>1 TB</storage>
                </resources>
            </node>
        </nodes>
        <links>
            <link>
                <id>0</id>
                <name>Horizontal link</name>
                <src>../../nodes/node[id=UUID11]/ports/port[id=2]</src>
                <dst>../../nodes/node[id=UUID12]/ports/port[id=2]</dst>
                <resources>
                    <delay>2 ms</delay>
                    <bandwidth>10 Gb</bandwidth>
                </resources>
            </link>
            <link>
                <id>1</id>
                <name>West link</name>
                <src>../../nodes/node[id=UUID11]/ports/port[id=1]</src>
                <dst>../../nodes/node[id=UUID13]/ports/port[id=2]</dst>
                <resources>
                    <delay>5 ms</delay>
                    <bandwidth>10 Gb</bandwidth>
                </resources>
            </link>
            <link>
                <id>2</id>
                <name>East link</name>
                <src>../../nodes/node[id=UUID12]/ports/port[id=0]</src>
                <dst>../../nodes/node[id=UUID13]/ports/port[id=1]</dst>
                <resources>
                    <delay>2 ms</delay>
                    <bandwidth>5 Gb</bandwidth>
                </resources>
```

```
            </link>
        </links>
    </virtualizer>
```

Figure 17: 3-node infrastructure report example: xml view

6.2. Simple requests

Figure 18 and Figure 19 show the allocation request for 3 NFs (NF1: Parental control B.4, NF2: Http Cache 1.2 and NF3: Stateful firewall C) as instrumented over a BiS-BiS node. It can be seen that the configuration request contains both the NF placement and the forwarding overlay definition as a joint request.

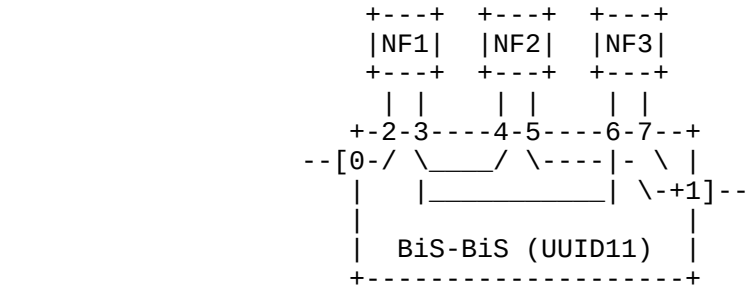


Figure 18: Simple request of 3 NFs on a single BiS-BiS: Virtualization view

```
<virtualizer xmlns="http://fp7-unify.eu/framework/virtualizer">
    <id>UUID001</id>
    <name>Single node simple request</name>
    <nodes>
        <node>
            <id>UUID11</id>
            <NF_instances>
                <node>
                    <id>NF1</id>
                    <name>first NF</name>
                    <type>Parental control B.4</type>
                    <ports>
                        <port>
                            <id>2</id>
                            <name>in</name>
                            <port_type>port-abstract</port_type>
                            <capability>...</capability>
                        </port>
                        <port>
                            <id>3</id>
```

```
                            <name>out</name>
                            <port_type>port-abstract</port_type>
                            <capability>...</capability>
                        </port>
                    </ports>
                </node>
                <node>
                    <id>NF2</id>
                    <name>cache</name>
                    <type>Http Cache 1.2</type>
                    <ports>
                        <port>
                            <id>4</id>
                            <name>in</name>
                            <port_type>port-abstract</port_type>
                            <capability>...</capability>
                        </port>
                        <port>
                            <id>5</id>
                            <name>out</name>
                            <port_type>port-abstract</port_type>
                            <capability>...</capability>
                        </port>
                    </ports>
                </node>
                <node>
                    <id>NF3</id>
                    <name>firewall</name>
                    <type>Stateful firewall C</type>
                    <ports>
                        <port>
                            <id>6</id>
                            <name>in</name>
                            <port_type>port-abstract</port_type>
                            <capability>...</capability>
                        </port>
                        <port>
                            <id>7</id>
                            <name>out</name>
                            <port_type>port-abstract</port_type>
                            <capability>...</capability>
                        </port>
                    </ports>
                </node>
            </NF_instances>
            <flowtable>
                <flowentry>
                    <port>../../ports/port[id=0]</port>
```

```
                        <match>*</match>
                        <action>output:../../NF_instances/node[id=NF1]
                          /ports/port[id=2]</action>
                    </flowentry>
                    <flowentry>
                        <port>../../NF_instances/node[id=NF1]
                          /ports/port[id=3]</port>
                        <match>fr-a</match>
                        <action>output:../../NF_instances/node[id=NF2]
                          /ports/port[id=4]</action>
    rpcre                </flowentry>
                    <flowentry>
                        <port>../../NF_instances/node[id=NF1]
                          /ports/port[id=3]</port>
                        <match>fr-b</match>
                        <action>output:../../NF_instances/node[id=NF3]
                          /ports/port[id=6]</action>
                    </flowentry>
                    <flowentry>
                        <port>../../NF_instances/node[id=NF2]
                          /ports/port[id=5]</port>
                        <match>*</match>
                        <action>output:../../ports/port[id=1]</action>
                    </flowentry>
                    <flowentry>
                        <port>../../NF_instances/node[id=NF3]
                          /ports/port[id=7]</port>
                        <match>*</match>
                        <action>output:../../ports/port[id=1]</action>
                    </flowentry>
                </flowtable>
            </node>
        </nodes>
    </virtualizer>
```

Figure 19: Simple request of 3 NFs on a single BiS-BiS: xml view

7. Experimentations

We have implemented the proposed recursive control plane architecture with joint software and network virtualization and control. We used a Python based open source implementation [virtualizer-library] of the virtualizer data structure for the orchestration API. We used the Extensible Service ChAin Prototyping Environment (ESCAPE) [ESCAPE] as the general orchestration platform with various technology specific domain adapters like OpenStack, Docker and Ryu SDN controller. A detailed service function chaining report is available at [I-D.unify-sfc-control-plane-exp].

8. IANA Considerations

This memo includes no request to IANA.

9. Security Considerations

TBD

10. Acknowledgement

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 619609 - the UNIFY project. The views expressed here are those of the authors only. The European Commission is not liable for any use that may be made of the information in this document.

We would like to thank in particular David Jocha and Janos Elek from Ericsson for the useful discussions.

11. Informative References

[ESCAPE] BME, "Extensible Service ChAin Prototyping Environment (open source)", Mar. 2016, <http://sb.tmit.bme.hu/mediawiki/index.php/ESCAPE>.

[ETSI-NFV-Arch] ETSI, "Architectural Framework v1.1.1", Oct 2013, <http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf>.

[ETSI-NFV-MANO] ETSI, "Network Function Virtualization (NFV) Management and Orchestration V0.6.1 (draft)", Jul. 2014, <http://docbox.etsi.org/ISG/NFV/Open/Latest_Drafts/NFV-MAN001v061-%20management%20and%20orchestration.pdf>.

[I-D.caszpe-nfvrg-orchestration-challenges] Carrozzo, G., Szabo, R., and K. Pentikousis, "Network Function Virtualization: Resource Orchestration Challenges", draft-caszpe-nfvrg-orchestration-challenges-00 (work in progress), November 2015.

[I-D.ietf-sfc-dc-use-cases] Surendra, S., Tufail, M., Majee, S., Captari, C., and S. Homma, "Service Function Chaining Use Cases In Data Centers", draft-ietf-sfc-dc-use-cases-04 (work in progress), January 2016.

[I-D.unify-nfvrg-challenges]
Szabo, R., Csaszar, A., Pentikousis, K., Kind, M., Daino, D., Qiang, Z., and H. Woesner, "Unifying Carrier and Cloud Networks: Problem Statement and Challenges", draft-unify-nfvrg-challenges-03 (work in progress), January 2016.

[I-D.unify-sfc-control-plane-exp]
Szabo, R. and B. Sonkoly, "SFC Control Plane Experiment: UNIFYed Approach", March 2016, <draft-unify-sfc-control-plane-exp>.

[I-D.zu-nfvrg-elasticity-vnf]
Qiang, Z. and R. Szabo, "Elasticity VNF", draft-zu-nfvrg-elasticity-vnf-01 (work in progress), March 2015.

[virtualizer-library]
Ericsson, "Python based virtualizer library for Netconf protocol (open source)", Mar. 2016, <https://github.com/Ericsson/unify-virtualizer>.

Authors' Addresses

Robert Szabo (editor)
Ericsson Research, Hungary
Irinyi Jozsef u. 4-20
Budapest 1117
Hungary

Email: robert.szabo@ericsson.com
URI: http://www.ericsson.com/

Zu Qiang
Ericsson
8400, boul. Decarie
Ville Mont-Royal, QC 8400
Canada

Email: zu.qiang@ericsson.com
URI: http://www.ericsson.com/

Mario Kind
Deutsche Telekom AG
Winterfeldtstr. 21
10781 Berlin
Germany

Email: mario.kind@telekom.de

Network Working Group
Internet Draft
Intended status: Standards Track
Expires: September 2016

V.liu
China Mobile

March 21, 2016

Analytic Framework for NFV Orchestrator
draft-liu-nfvrg-analytic-framework-00.txt

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 21, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document.

Abstract

This draft introduces an analytic model and framework with data collection and policy management in NFV Orchestrator.

Table of Contents

1. Introduction 2
2. Data collection and policy inventory..... 3
2.1. Data collection framework..... 3
2.2. Policy inventory..... 4
3. Analytic model and framework..... 5
3.1. The real-time analytic mode 6
3.2. The non-real-time analytic model..... 8
4. Analytic framework in NFV Orchestrator..... 9
5. Security Considerations..... 10
6. IANA Considerations 10
7. Conclusions 10
8. References 10
8.1. Normative References..... 11
8.2. Informative References..... 11
9. Acknowledgments 11

1. Introduction

As NFV being researched, we have observed more and more research started focusing on NFV Orchestrator. Most research in Orchestrator is to address issues around orchestrator architecture, task fulfillment, status monitoring as well as network analytic and policy management.

In this draft, we would like to focus on discussing the network analytic and policy management (FAPM) module. After that, we will introduce some Orchestrator architecture, feature and the relationship with FAPM module. We propose this draft because we believe FAPM will play an important role in future network orchestration. We have researched and implemented the features of orchestration task fulfillment and VIM resource monitoring. In the GUI, the network operator can define Network Service, manage VNF lifecycle (e.g. create, terminate, scale in/out), monitor its status from VIM ceilometer. However, the current network policy is simple and pre-set in orchestrator. If the FAPM module can be integrated into this orchestration loop (monitor->FAPM->fulfillment), we may

create a more 'intelligent' network orchestration and policy management.

In section 2 of this draft, we would like to introduce the capability of data monitor and collection attributes by Orchestrator. Besides, the policy inventory is also important to introduce in this section.

In section 3, we proposed the analytic model and framework for FAPM module. The real-time analytic model and the non-real-time analytic model are described respectively. We present one data mining techniques for the real-time analytic model, and provide the expectation for the non-real-time analytic model.

In section 4, we describe the Orchestrator architecture along with introduction of how FAPM module fits in this architecture.

2. Data collection and policy inventory

2.1. Data collection framework

The orchestrator is more like a central brain to provide a global manage and control of the network resource as well as the logical network service (e.g. NS model, SFC). By our research, we have onboard of several network elements (VNF) from different venders, such as vEPC, vIMS, vCPE, vBRAS and Nanocell Gateway. Above those VNFs, Data Collection Framework in Orchestrator is response of collecting status attribute. We have discovered that this work is not easy to accomplish due to the unfinished definition of monitoring API. Therefore, we propose a Data Collection Framework as below in Figure 1.

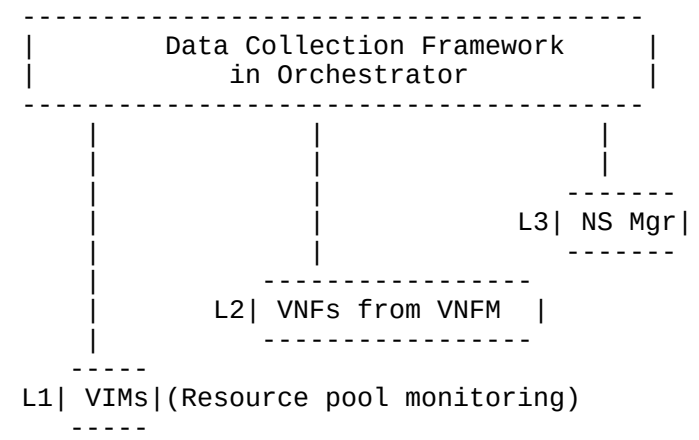


Figure 1: The framework of data collection

The monitoring data collection consists of three layers. The first layer is connecting between VIM and Orchestrator to collect resource pool status, like CPU usage, Memory usage, File-system usage and Nic(network) usage. The second layer is connecting between VNFM and Orchestrator to collect VNFs status. In addition, the third layer is connecting between NS manager and Orchestrator to collect service status.

2.2. Policy inventory

Conceptually, the policy is the key to trigger the network management action such as scale in/out. The analytic model will calculate the decision to hit the particular policies from policy inventory. For the policy in Orchestrator, previously there are simple pre-set policies, such as an attribute of a threshold value for VNF scaled up. When the status reaches the attribute threshold value, the system will trigger the policy action. However, the real situation of network is unstable or fluctuant. For example, we assume that the attribute value to trigger the scale-out action is by the CPU usage. The real situation is there are lots of peaks of CPU usage’s curve that over threshold value. How did the system decide to trigger the action of scaling out? With this question, we propose a closed circle to work out this problem, shown in Figure 2.

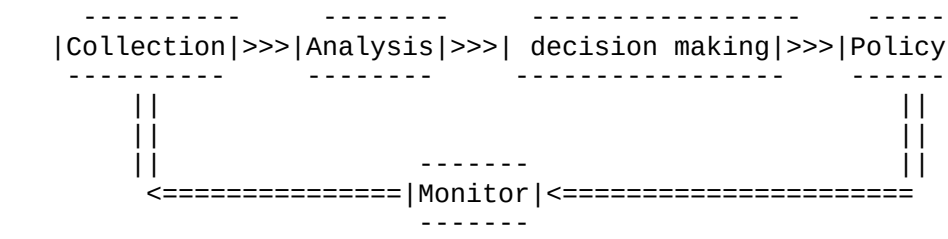


Figure 2: The closed circle of FAPM

In Figure 2, the collection model depicts the data collection mentioned previously. The block of analysis depicts the analytic model described in the following section. It provides data analysis to predict the results or make decisions thus trigger the hit in policy inventory. The monitor block provides the feedback of the circle.

3. Analytic model and framework

As the development of Orchestrator, a more intelligent trend wishes to be researched and exploited. One of the FAPM's function, analytic model, is designed for achieve the goal.

As tons of the status attribute data are collected, we cannot directly obtain the relationship among them. However, data analysis can be implemented to process and model the data to trigger the conclusion and decision-making to the orchestrator. Data mining is a particular data analysis technique that focuses on modeling and knowledge discovery for predictive [wiki]. It is a promising yet challenge direction to enforce into the Orchestrator.

The analytic module in the Orchestrator can be divided into two parts. One is the real-time analysis which can provide an on demand network policy delivery for features such as dynamic scale in/out or auto-healing, etc. The other one is the non-real-time analysis that process batch data (e.g. log, etc.) and provide for static analysis and global resource planning. For real-time analytic model, we propose K-nearest neighbor (KNN) algorithm, because KNN can be realized in this analysis to assistant other components making real-time response. In non-real-time analytic model, unfortunately, the collection module has not accomplished, we just bring out an assumption of using decision trees learning, or neural networks.

3.1. The real-time analytic model

The real-time analytic model is a significantly dynamic part that is processing the real-time data and predicting results or trends for employing by other components, like optimized resource, work distribution, etc. It requires a real-time method to attain the target of quickly response and opportunely updating model. Accordingly, the real-time KNN algorithm is proposed to realize the goal.

Firstly, KNN algorithm is a type of instance-based learning, or lazy learning used for classification. The input consists of the k closest training examples in the feature space, and the output depends on a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors. It assumes all the instances correspond to points in the n-dimensional space.

In the training phase, the algorithm only stores the training data and their corresponding labels. More precisely, an instance x can be described as

[a1(x), a2(x), ? an(x)]

where the ai(x) denotes the ith attribute (e.g. CPU utilization or memory usage or file-system usage or nic usage or etc.) of instance x, and its corresponded label is marked as l. All kinds of labels will have their own corresponding relations to the following actions or events, like the number 1 could present to increase or decrease the number of virtual machines, etc.

In the classification phase, one defined number of k should be provided, and a testing instance y (or an unlabeled instance) is classified by assigning the label which is most frequent among the k training samples nearest to that query point [wiki]. Generally, the standard Euclidean distance is utilized to define the nearest neighbor of an instance as

d(x,y)=sqrt(sum(ai(x)-ai(y))^2)

where d(x,y) denotes the distance between x and y. After obtaining the distance matrix D between the testing instance y and all training instances, the nearest k neighbors could be utilized to vote the label of the testing instance y.

Figure 3 illustrates an example of implementing the KNN algorithm, where the instances are points in a two-dimensional space and the labels are marked by different numbers. For example, the two-dimensional space could include two arbitrary attributes of collected data, like CPU usage and memory usage. The marked numbers could present the action of scaling up, scaling down or termination. The testing instance y is shown as well. If the k is assumed as 5, y will be classified as class 1 by voting. Furthermore, this algorithm also can be implemented to analyze for either low-dimensional or high-dimensional data. It can be easily to extend the example of two-dimensional space to a higher dimension by increasing the number of attributes.

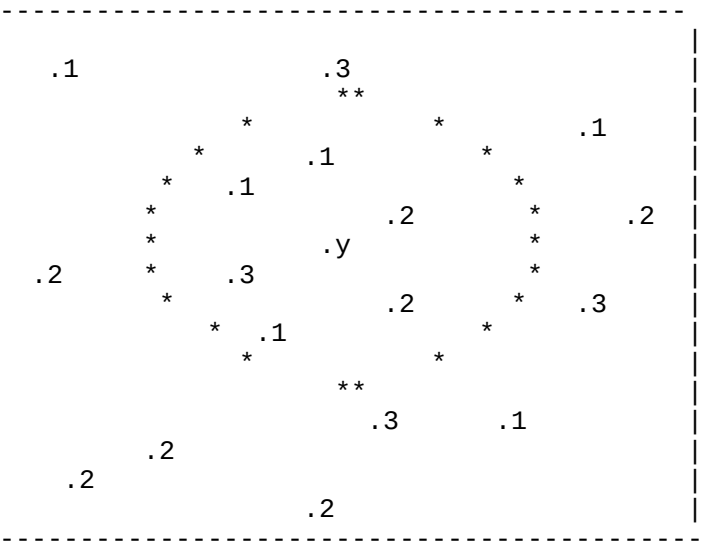


Figure 3: An example of KNN algorithm

Based on the KNN algorithm, our purpose is to achieve the real-time analytic model. Figure 4 describes the real-time model implemented in the KNN algorithm. Firstly, the same process is implemented to classify the testing data. In addition, there will be a detection model to check the label of it. If the label does not generate some anomalies, the testing data can be added into the training database to achieve the real-time analysis and update the training dataset.

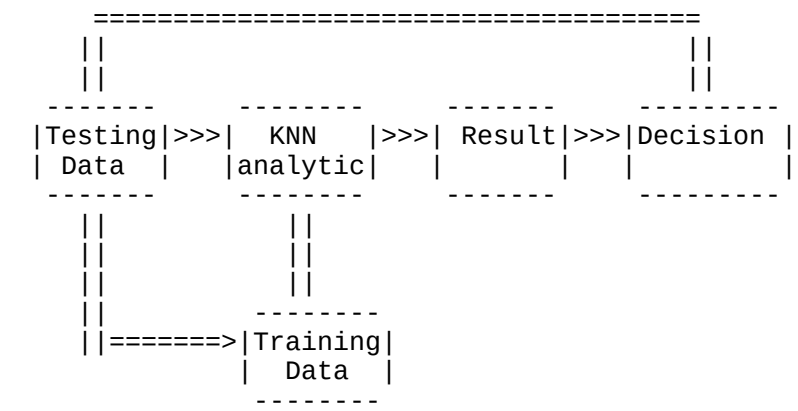


Figure 4: The model of real-time KNN algorithm

KNN algorithm is one of the most fundamental and simple classification methods, which can be relatively easy to understand. In addition, it is a non-parametric method that can omit the process of setting and optimizing parameters. On the contrary, KNN algorithm is computationally expensive, and requires lots of memory. Fortunately, the orchestration usually implemented by cloud computing which can overcome the weakness. Moreover, there are other solutions to improve this algorithm, like dimensionality reduction, optimized training dataset, etc.

Eventually, the real-time analytic model is proposed. It is achieved by the real-time KNN algorithm to predict the outcomes or directions. Furthermore, the real-time analytic model can fulfill the dynamic control, which can assistant the Orchestrator to achieve a more intelligent way.

3.2. The non-real-time analytic model

For the non-real-time analytic model, it still plays an indispensable role to help the system to predict the future situation. It needs a pre-trained model to obtain the prediction of the following situation. In the case of that, it can help other components to achieve static analysis, etc. Therefore, the method of data mining (e.g. decision tree learning, neural networks, etc.) is a promising direction to deal with this.

4. Analytic framework in NFV Orchestrator

In this section, we would like to introduce the architecture of Orchestrator and how analytic and policy management (FAPM) modules fit in this Architecture. Since we initiate the Open-O project from mid-2015, we have finished most work in task fulfillment and monitoring. The architecture with analytic and policy is shown below in Figure 5.

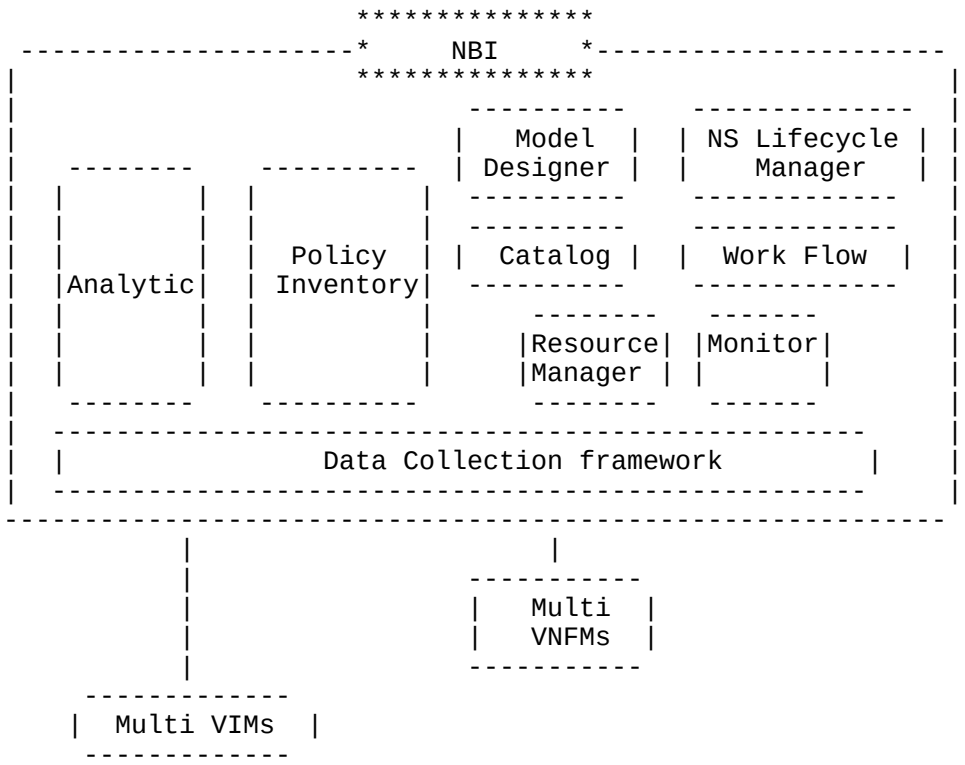


Figure 5: The architecture with analytic and policy

From down to top, the Orchestrator supports Multi VIMs, such as OpenStack, as well as multi VNFMs, such as tacker, Huawei VNFM and other 3rd party VNMFs. With help of those VNFMs, we have onboard VNFs such as vEPC, vIMS, vBras, vCPE and Nanocell Gateway.

On the top of VIM and VNFM, the Orchestrator acts like a central brain of the network. There can be divided as three parts.

The first part is the FAPM module which include Data Collection Platform, Analytic and Policy Inventory. The Data Collection Platform is response for collection status form VIM, VNFM and NS which are discussed in section 2 and provides internal API for Analytic, Policy Inventory and Monitor module to use. The Analytic module is in charge of processing the network analysis and provide decision-making from the Policy Inventory. The Policy Inventory currently reposits policies which are input from NBI manually.

The second part of Open-O is Resource Orchestrator (OR) which includes Resource manager and Monitor. The resource manager is in charge of manager the connection with VNFM to globally manager the VNF and VIM resource. And the Monitor is called from the Data Collection Platform to get the system status.

The third part of Open-O is Network Service Orchestrator(NSO) which includes Model Designer, Catalog, NS lifecycle Manager and workflow engine. Model Designer can provide network operator and user a GUI that allows them to design the network topology and create data model of VNF, NS, VNFG, VL. The Catalog can store the VNF, NS, VNFG, VL data model. NS lifecycle manager is for NS level scale in/out.

Workflow engine assist the lifecycle management process by interacted with internal and external modules.

5. Security Considerations

6. IANA Considerations

7. Conclusions

In this draft, we have introduced data collection framework. And bring out an analytic model and framework with Orchestrator status data. At last, we introduced the analytic and policy architecture with Orchestrator.

8. References

8.1. Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[2] Crocker, D. and Overell, P.(Editors), "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, Internet Mail Consortium and Demon Internet Ltd., November 1997.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2234] Crocker, D. and Overell, P.(Editors), "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, Internet Mail Consortium and Demon Internet Ltd., November 1997.

8.2. Informative References

[3] Faber, T., Touch, J. and W. Yue, "The TIME-WAIT state in TCP and Its Effect on Busy Servers", Proc. Infocom 1999 pp. 1573-1583.

[Fab1999] Faber, T., Touch, J. and W. Yue, "The TIME-WAIT state in TCP and Its Effect on Busy Servers", Proc. Infocom 1999 pp. 1573-1583.

9. Acknowledgments

This document was prepared using 2-Word-v2.0.template.dot.

This draft is also contributed by Yuan Liu from China Mobile.

Authors' Addresses

Vic Liu
China Mobile
32 Xuanwumen West AVE, Xicheng, Beijing
Email: liuzhiheng@chinamobile.com

NFVRG
Internet Draft
Intended status: Informational
Expires: January 2017

C. Meirosu
Ericsson
A. Manzalini
Telecom Italia
R. Steinert
SICS
G. Marchetto
Politecnico di Torino
K. Pentikousis
EICT
S. Wright
AT&T
P. Lynch
Ixia
W. John
Ericsson

July 8, 2016

DevOps for Software-Defined Telecom Infrastructures
draft-unify-nfvrg-devops-05.txt

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This Internet-Draft will expire on January 8, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Abstract

Carrier-grade network management was optimized for environments built with monolithic physical nodes and involves significant deployment, integration and maintenance efforts from network service providers. The introduction of virtualization technologies, from the physical layer all the way up to the application layer, however, invalidates several well-established assumptions in this domain. This draft opens the discussion in NFVRG about challenges related to transforming the telecom network infrastructure into an agile, model-driven environment for communication services. We take inspiration from data center DevOps on the simplification and automation of management processes for a telecom service provider software-defined infrastructure (SDI). A number of challenges associated with operationalizing DevOps principles at scale in software-defined telecom networks are identified in relation to three areas related to key programmable management processes.

Table of Contents

1. Introduction.....3
2. Software-Defined Telecom Infrastructure: Roles and DevOps principles.....5
   2.1. Service Developer Role.....6
   2.2. VNF Developer role.....6
   2.3. System Integrator role.....6
   2.4. Operator role.....7
   2.5. Customer role.....7
   2.6. DevOps Principles.....7
3. Continuous Integration.....9
4. Continuous Delivery.....10

5. Consistency, Availability and Partitioning Challenges.....10
6. Stability and Real-Time Change Challenges.....11
7. Observability Challenges.....13
8. Verification Challenges.....15
9. Testing Challenges.....17
10. Programmable management.....18
11. Security Considerations.....20
12. IANA Considerations.....20
13. References.....20
13.1. Informative References.....20
14. Contributors to earlier versions.....23
15. Acknowledgments.....23
16. Authors' Addresses.....24

1. Introduction

Carrier-grade network management was developed as an incremental solution once a particular network technology matured and came to be deployed in parallel with legacy technologies. This approach requires significant integration efforts when new network services are launched. Both centralized and distributed algorithms have been developed in order to solve very specific problems related to configuration, performance and fault management. However, such algorithms consider a network that is by and large functionally static. Thus, management processes related to introducing new or maintaining functionality are complex and costly due to significant efforts required for verification and integration.

Network virtualization, by means of Software-Defined Networking (SDN) and Network Function Virtualization (NFV), creates an environment where network functions are no longer static or strictly embedded in physical boxes deployed at fixed points. The virtualized network is dynamic and open to fast-paced innovation enabling efficient network management and reduction of operating cost for network operators. A significant part of network capabilities are expected to become available through interfaces that resemble the APIs widespread within datacenters instead of the traditional telecom means of management such as the Simple Network Management Protocol, Command Line Interfaces or CORBA. Such an API-based approach, combined with the programmability offered by SDN interfaces [RFC7426], open opportunities for handling infrastructure, resources, and Virtual Network Functions (VNFs) as code, employing techniques from software engineering.

The efficiency and integration of existing management techniques in virtualized and dynamic network environments are limited, however. Monitoring tools, e.g. based on simple counters, physical network

taps and active probing, do not scale well and provide only a small part of the observability features required in such a dynamic environment. Although huge amounts of monitoring data can be collected from the nodes, the typical granularity is rather static and coarse and management bandwidths may be limited. Debugging and troubleshooting techniques developed for software-defined environments are a research topic that has gathered interest in the research community in the last years. Still, it is yet to be explored how to integrate them into an operational network management system. Moreover, research tools developed in academia (such as NetSight [H2014], OFRewind [W2011], FlowChecker [S2010], etc.) were limited to solving very particular, well-defined problems, and oftentimes are not built for automation and integration into carrier-grade network operations workflows. As the virtualized network functions, infrastructure software and infrastructure hardware become more dynamic [NFVSWA], the monitoring, management and testing approaches also need to change.

The topics at hand have already attracted several standardization organizations to look into the issues arising in this new environment. For example, IETF working groups have activities in the area of OAM and Verification for Service Function Chaining [I-D.aldrin-sfc-oam-framework] [I-D.lee-sfc-verification] for Service Function Chaining. At IRTF, [RFC7149] asks a set of relevant questions regarding operations of SDNs. The ETSI NFV ISG defines the MANO interfaces [NFVMANO], and TMForum investigates gaps between these interfaces and existing specifications in [TR228]. The need for programmatic APIs in the orchestration of compute, network and storage resources is discussed in [I-D.unify-nfvrg-challenges].

From a research perspective, problems related to operations of software-defined networks are in part outlined in [SDNsurvey] and research referring to both cloud and software-defined networks are discussed in [D4.1].

The purpose of this first version of this document is to act as a discussion opener in NFVRG by describing a set of principles that are relevant for applying DevOps ideas to managing software-defined telecom network infrastructures. We identify a set of challenges related to developing tools, interfaces and protocols that would support these principles and how can we leverage standard APIs for simplifying management tasks.

2. Software-Defined Telecom Infrastructure: Roles and DevOps principles

There is no single list of core principles of DevOps, but it is generally recognized as encompassing:

- Iterative development / Incremental feature content
- Continuous deployment
- Automated processes
- Holistic/Systemic views of development and deployment/ operation.

With Deployment/ Operations becoming increasingly linked with software development, and business needs driving more rapid deployments, agile methodologies are assumed as a basis for DevOps. Agile methods used in many software focused companies are focused on releasing small interactions of code to implement VNFs with high velocity and high quality into a production environment. Similarly, Service providers are interested to release incremental improvements in the network services that they create from virtualized network functions. The cycle time for DevOps as applied in many open source projects is on the order of one quarter year or 13 weeks.

The code needs to undergo a significant amount of automated testing and verification with pre-defined templates in a realistic setting. From the point of view of software defined telecom infrastructure management, the of the network and service configuration is expected to continuously evolve as result of network policy decomposition and refinement, service evolution, the updates, failovers or re-configuration of virtual functions, additions/upgrades of new infrastructure resources (e.g. whiteboxes, fibers). When troubleshooting the cause of unexpected behavior, fine-grained visibility onto all resources supporting the virtual functions (either compute, or network-related) is paramount to facilitating fast resolution times. While compute resources are typically very well covered by debugging and profiling toolsets based on many years of advances in software engineering, programmable network resources are a still a novelty and tools exploiting their potential are scarce.

2.1. Service Developer Role

We identify two dimensions of the "developer" role in software-defined infrastructure (SDI). The network service to be developed is captured in a network service descriptor (e.g. [IFA014]). One dimension relates to determining which high-level functions should be part of a particular service, deciding what logical interconnections are needed between these blocks and defining a set of high-level constraints or goals related to parameters that define, for instance, a Service Function Chain. This could be determined by the product owner for a particular family of services offered by a telecom provider. Or, it might be a key account representative that adapts an existing service template to the requirements of a particular customer by adding or removing a small number of functional entities. We refer to this person as the Service Developer and for simplicity (access control, training on technical background, etc.) we consider the role to be internal to the telecom provider.

2.2. VNF Developer role

Another dimension of the "developer" role is a person that writes the software code for a new virtual network function (VNF). The VNF then needs to be delivered as a package (e.g.[IFA011]) that includes various metadata for ingestion/integration into some service. Note that a VNF may span multiple virtual machines to support design objectives (e.g. for reliability or scalability). Depending on the actual VNF being developed, this person might be internal or external (e.g. a traditional equipment vendor) to the telecom provider. We refer to them as VNF Developers.

2.3. System Integrator role

The System Integrator role is to some extent similar to the Service Developer: people in this role need to identify the components of the system to be delivered. However, for the Service Developer, the service components are pre-integrated meaning that they have the right interfaces to interact with each other. In contrast, the Systems Integrator needs to develop the software that makes the system components interact with each other. As such, the Systems Integrator role combines aspects of the Developer roles and adds yet another dimension to it. Compared to the other Developer roles, the System Integrator might face additional challenges due to the fact that they might not have access to the source code of some of the components. This limits for example how fast they could address issues with components to be integrated, as well as uneven workload depending on the release granularity of the different components that need to be integrated. Some system integration activities may take

place on an industry basis in collaborative communities (e.g. OPNFV.org).

2.4. Network service Operator role

The role of a Network Service Operator is to ensure that the deployment processes were successful and a set of performance indicators associated to a particular network service are met. The network service is supported on infrastructure specific set of infrastructure resources that may be owned and operated by that Network Service Operator, or provided under contract from some other infrastructure service provider. .

2.5. Customer role

A Customer contracts a telecom operator to provide one or more services. In SDI, the Customer may communicate with the provider in real time through an online portal. From the customer perspective, such portal interfaces become part of the service definition just like the data transfer aspects of the service. Compared to the Service Developer, the Customer is external to the operator and may define changes to their own service instance only in accordance to policies defined by the Service Developer. In addition to the usual per-service utilization statistics, in SDI the portal may enable the customer to trigger certain performance management or troubleshooting tools for the service. This, for example, enables the Customer to determine whether the root cause of certain error or degradation condition that they observe is located in the telecom operator domain or not and may facilitate the interaction with the customer support teams.

2.6. DevOps Principles

In line with the generic DevOps concept outlined in [DevOpsP], we consider that these four principles as important for adapting DevOps ideas to SDI:

* Automated processes: Deploy with repeatable, reliable processes: Service and VNF Developers should be supported by automated build, orchestrate and deploy processes that are identical in the development, test and production environments. Such processes need to be made reliable and trusted in the sense that they should reduce the chance of human error and provide visibility at each stage of the process, as well as have the possibility to enable manual interactions in certain key stages.

* Holistis/systemic view: Develop and test against production-like systems: both Service Developers and VNF Developers need to have the opportunity to verify and debug their respective SDI code in systems that have characteristics which are very close to the production environment where the code is expected to be ultimately deployed. Customizations of Service Function Chains or VNFs could thus be released frequently to a production environment in compliance with policies set by the Operators. Adequate isolation and protection of the services active in the infrastructure from services being tested or debugged should be provided by the production environment.

* Continuous: Monitor and validate operational quality: Service Developers, VNF Developers and Operators must be equipped with tools, automated as much as possible, that enable to continuously monitor the operational quality of the services deployed on SDI. Monitoring tools should be complemented by tools that allow verifying and validating the operational quality of the service in line with established procedures which might be standardized (for example, Y.1564 Ethernet Activation [Y1564]) or defined through best practices specific to a particular telecom operator.

* Iterative/Incremental: Amplify development cycle feedback loops: An integral part of the DevOps ethos is building a cross-cultural environment that bridges the cultural gap between the desire for continuous change by the Developers and the demand by the Operators for stability and reliability of the infrastructure. Feedback from customers is collected and transmitted throughout the organization. From a technical perspective, such cultural aspects could be addressed through common sets of tools and APIs that are aimed at providing a shared vocabulary for both Developers and Operators, as well as simplifying the reproduction of problematic situations in the development, test and operations environments.

Network operators that would like to move to agile methods to deploy and manage their networks and services face a different environment compared to typical software companies where simplified trust relationships between personnel are the norm. In software companies, it is not uncommon that the same person may be rotating between different roles. In contrast, in a telecom service provider, there are strong organizational boundaries between suppliers (whether in Developer roles for network functions, or in Operator roles for outsourced services) and the carrier’s own personnel that might also take both Developer and Operator roles. Extending DevOps principles across strong organizational boundaries e.g. through co-creation or collaborative development in open source communities) may be a commercial challenge rather than a technical issue.

3. Continuous Integration

Software integration is the process of bringing together the software component subsystems into one software system, and ensuring that the subsystems function together as a system. Software integration can apply regardless of the size of the software components. The objective of Continuous Integration is to prevent integration problems close to the expected release of a software development project into a production (operations) environment. Continuous Integration is therefore closely coupled with the notion of DevOps as a mechanism to ease the transition from development to operations.

Continuous integration may result in multiple builds per day. It is also typically used in conjunction with test driven development approaches that integrate unit testing into the build process. The unit testing is typically automated through build servers. Such servers may implement a variety of additional static and dynamic tests as well as other quality control and documentation extraction functions. The reduced cycle times of continuous enable improved software quality by applying small efforts frequently.

Continuous Integration applies to developers of VNF as they integrate the components that they need to deliver their VNF. The VNFs may contain components developed by different teams within the VNF Provider, or may integrate code developed externally - e.g. in commercial code libraries or in open source communities.

Service developers also apply continuous integration in the development of network services. Network services are comprised of various aspects including VNFs and connectivity within and between them as well as with various associated resource authorizations. The components of the networks service are all dynamic, and largely represented by software that must be integrated regularly to maintain consistency.

Some of the software components that Service Developers integrate may be sourced from VNF Providers or from open source communities. Service Developers and Network Service Operators are increasingly motivated to engage with open Source communities [OSandS]. Open source interfaces supported by open source communities may be more useful than traditional paper interface specifications. Even where Service Providers are deeply engaged in the open source community (e.g. OPNFV) many service providers may prefer to obtain the code through some software provider as a business practice. Such software providers have the same interests in software integration as other

VNF providers. An open source integration community (e.g. OPNFV) may resolve common integration issues across the industry reducing the need for integration issue resolution specific to particular integrators.

4. Continuous Delivery

The practice of Continuous Delivery extends Continuous Integration by ensuring that the software (either a VNF code or code for SDI) checked in on the mainline is always in a user deployable state and enables rapid deployment by those users. For critical systems such as telecommunications networks, Continuous Delivery may require the advantage of including a manual trigger before the actual deployment in the live system, compared to the Continuous Deployment methodology which is also part of DevOps processes in software companies.

Automated Continuous deployment systems in may exceed 10 updates per day. Assuming an integration of 100 components, each with an average time to upgrade of 180 days then deployments on the order of every 1.8 days might be expected. The telecom infrastructure is also very distributed - consider the case of cloud RAN use cases where the number of locations for deployment is of the order of the number of cell tower locations (˜10^4..10^6). Deployments may need to be incremental across the infrastructure to reduce the risk of large-scale failures. Conversely, there may need to be rapid rollbacks to prior stable deployment configurations in the event of significant failures.

5. Consistency, Availability and Partitioning Challenges

The CAP theorem [CAP] states that any networked shared-data system can have at most two of following three properties: 1) Consistency (C) equivalent to having a single up-to-date copy of the data; 2) high Availability (A) of that data (for updates); and 3) tolerance to network Partitions (P).

Looking at a telecom SDI as a distributed computational system (routing/forwarding packets can be seen as a computational problem), just two of the three CAP properties will be possible at the same time. The general idea is that 2 of the 3 have to be chosen. CP favor consistency, AP favor availability, CA there are no partition. This has profound implications for technologies that need to be developed in line with the "deploy with repeatable, reliable processes"

principle for configuring SDI states. Latency or delay and partitioning properties are closely related, and such relation becomes more important in the case of telecom service providers where Devs and Ops interact with widely distributed infrastructure. Limitations of interactions between centralized management and distributed control need to be carefully examined in such environments. Traditionally connectivity was the main concern: C and A was about delivering packets to destination. The features and capabilities of SDN and NFV are changing the concerns: for example in SDN, control plane Partitions no longer imply data plane Partitions, so A does not imply C. In practice, CAP reflects the need for a balance between local/distributed operations and remote/centralized operations.

Furthermore to CAP aspects related to individual protocols, interdependencies between CAP choices for both resources and VNFs that are interconnected in a forwarding graph need to be considered. This is particularly relevant for the "Monitor and Validate Operational Quality" principle, as apart from transport protocols, most OAM functionality is generally configured in processes that are separated from the configuration of the monitored entities. Also, partitioning in a monitoring plane implemented through VNFs executed on compute resources does not necessarily mean that the dataplane of the monitored VNF was partitioned as well.

6. Stability and Real-Time Change Challenges

The dimensions, dynamicity and heterogeneity of networks are growing continuously. Monitoring and managing the network behavior in order to meet technical and business objectives is becoming increasingly complicated and challenging, especially when considering the need of predicting and taming potential instabilities.

In general, instability in networks may have primary effects both jeopardizing the performance and compromising an optimized use of resources, even across multiple layers: in fact, instability of end-to-end communication paths may depend both on the underlying transport network, as well as the higher level components specific to flow control and dynamic routing. For example, arguments for introducing advanced flow admission control are essentially derived from the observation that the network otherwise behaves in an inefficient and potentially unstable manner. Even with resources over provisioning, a network without an efficient flow admission control has instability regions that can even lead to congestion collapse in certain configurations. Another example is the instability which is

characteristic of any dynamically adaptive routing system. Routing instability, which can be (informally) defined as the quick change of network reachability and topology information, has a number of possible origins, including problems with connections, router failures, high levels of congestion, software configuration errors, transient physical and data link problems, and software bugs.

As a matter of fact, the states monitored and used to implement the different control and management functions in network nodes are governed by several low-level configuration commands. There are several dependencies among these states and the logic updating the states in real time (most of which are not synchronized automatically). Normally, high-level network goals (such as the connectivity matrix, load-balancing, traffic engineering goals, survivability requirements, etc) are translated into low-level configuration commands (mostly manually) individually executed on the network elements (e.g., forwarding table, packet filters, link-scheduling weights, and queue-management parameters, as well as tunnels and NAT mappings). Network instabilities due to configuration errors can spread from node to node and propagate throughout the network.

DevOps in the data center is a source of inspiration regarding how to simplify and automate management processes for software-defined infrastructure. Although the low-level configuration could be automated by DevOps tools such as CFEngine [C2015], Puppet [P2015] and Ansible [A2015], the high-level goal translation towards tool-specific syntax is still a manual process. In addition, while carrier-grade configuration tools using the NETCONF protocol support complex atomic transaction management (which reduces the potential for instability), Ansible requires third-party components to support rollbacks and the Puppet transactions are not atomic.

As a specific example, automated configuration functions are expected to take the form of a "control loop" that monitors (i.e., measures) current states of the network, performs a computation, and then reconfigures the network. These types of functions must work correctly even in the presence of failures, variable delays in communicating with a distributed set of devices, and frequent changes in network conditions. Nevertheless cascading and nesting of automated configuration processes can lead to the emergence of non-linear network behaviors, and as such sudden instabilities (i.e. identical local dynamic can give rise to widely different global dynamics).

7. Observability Challenges

Monitoring algorithms need to operate in a scalable manner while providing the specified level of observability in the network, either for operation purposes (Ops part) or for debugging in a development phase (Dev part). We consider the following challenges:

* Scalability - relates to the granularity of network observability, computational efficiency, communication overhead, and strategic placement of monitoring functions.

* Distributed operation and information exchange between monitoring functions - monitoring functions supported by the nodes may perform specific operations (such as aggregation or filtering) locally on the collected data or within a defined data neighborhood and forward only the result to a management system. Such operation may require modifications of existing standards and development of protocols for efficient information exchange and messaging between monitoring functions. Different levels of granularity may need to be offered for the data exchanged through the interfaces, depending on the Dev or Ops role. Modern messaging systems, such as Apache Kafka [AK2015], widely employed in datacenter environments, were optimized for messages that are considerably larger than reading a single counter value (typical SNMP GET call usage) - note the throughput vs record size from [K2014]. It is also debatable to what extent properties such as message persistence within the bus are needed in a carrier environment, where MIBs practically offer already a certain level of persistence of management data at the node level. Also, they require the use of IP addressing which might not be needed when the monitored data is consumed by a function within the same node.

* Common communication channel between monitoring functions and higher layer entities (orchestration, control or management systems) - a single communication channel for configuration and measurement data of diverse monitoring functions running on heterogeneous hard- and software environments. In telecommunication environments, infrastructure assets span not only large geographical areas, but also a wide range of technology domains, ranging from CPEs, access-, aggregation-, and transport networks, to datacenters. This heterogeneity of hard- and software platforms requires higher layer entities to utilize various parallel communication channels for either configuration or data retrieval of monitoring functions within these technology domains. To address automation and advances in monitoring programmability, software defined telecommunication infrastructures would benefit from a single flexible communication channel, thereby supporting the dynamicity of virtualized environments. Such a channel should ideally support propagation of

configuration, signalling, and results from monitoring functions; carrier-grade operations in terms of availability and multi-tenant features; support highly distributed and hierarchical architectures, keeping messages as local as possible; be lightweight, topology independent, network address agnostic; support flexibility in terms of transport mechanisms and programming language support.
Existing popular state-of-the-art message queuing systems such as RabbitMQ [R2015] fulfill many of these requirements. However, they utilize centralized brokers, posing a single point-of-failure and scalability concerns within vastly distributed NFV environment. Furthermore, transport support is limited to TCP/IP. ZeroMQ [Z2015] on the other hard lacks any advanced features for carrier-grade operations, including high-availability, authentication, and tenant isolation.

* Configurability and conditional observability - monitoring functions that go beyond measuring simple metrics (such as delay, or packet loss) require expressive monitoring annotation languages for describing the functionality such that it can be programmed by a controller. Monitoring algorithms implementing self-adaptive monitoring behavior relative to local network situations may employ such annotation languages to receive high-level objectives (KPIs controlling tradeoffs between accuracy and measurement frequency, for example) and conditions for varying the measurement intensity. Steps in this direction were taken by the DevOps tools such as Splunk [S2015], whose collecting agent has the ability to load particular apps that in turn access specific counters or log files. However, such apps are tool specific and may also require deploying additional agents that are specific to the application, library or infrastructure node being monitored. Choosing which objects to monitor in such environment means deploying a tool-specific script that configures the monitoring app.

* Automation - includes mapping of monitoring functionality from a logical forwarding graph to virtual or physical instances executing in the infrastructure, as well as placement and re-placement of monitoring functionality for required observability coverage and configuration consistency upon updates in a dynamic network environment. Puppet [P2015] manifests or Ansible [A2015] playbooks could be used for automating the deployment of monitoring agents, for example those used by Splunk [S2015]. However, both manifests and playbooks were designed to represent the desired system configuration snapshot at a particular moment in time - they would now need to be generated automatically by the orchestration tools instead of a DevOps person.

* Actionable data

Data produced by observability tools could be utilized in a wide category of processes, ranging from billing and dimensioning to real-time troubleshooting and optimization. In order to allow for data-driven automated decisions and actuations based on these decisions, the data needs to be actionable. We define actionable data as being representative for a particular context or situation and an adequate input towards a decision. Ensuring actionable data is challenging in a number of ways, including: defining adaptive correlation and sampling windows, filtering and aggregation methods that are adapted or coordinated with the actual consumer of the data, and developing analytical and predictive methods that account for the uncertainty or incompleteness of the data.

* Data Virtualization

Data is key in helping both Developers and Operators perform their tasks. Traditional Network Management Systems were optimized for using one database that contains the master copy of the operational statistics and logs of network nodes. Ensuring access to this data from across the organization is challenging because strict privacy and business secrets need to be protected. In DevOps-driven environments, data needs to be made available to Developers and their test environments. Data virtualization collectively defines a set of technologies that ensure that restricted copies of the partial data needed for a particular task may be made available while enforcing strict access control. Further than simple access control, data virtualization needs to address scalability challenges involved in copying large amounts of operational data as well as automatically disposing of it when the task authorized for using it has finished.

8. Verification Challenges

Enabling ongoing verification of code is an important goal of continuous integration as part of the data center DevOps concept. In a telecom SDI, service definitions, decompositions and configurations need to be expressed in machine-readable encodings. For example, configuration parameters could be expressed in terms of YANG data models. However, the infrastructure management layers (such as Software-Defined Network Controllers and Orchestration functions) might not always export such machine-readable descriptions of the runtime configuration state. In this case, the management layer itself could be expected to include a verification process that has the same challenges as the stand-alone verification processes we outline later in this section. In that sense, verification can be considered as a set of features providing gatekeeper functions to

verify both the abstract service models and the proposed resource configuration before or right after the actual instantiation on the infrastructure layer takes place.

A verification process can involve different layers of the network and service architecture. Starting from a high-level verification of the customer input (for example, a Service Graph as defined in [I-D.unify-nfvrg-challenges]), the verification process could go more in depth to reflect on the Service Function Chain configuration. At the lowest layer, the verification would handle the actual set of forwarding rules and other configuration parameters associated to a Service Function Chain instance. This enables the verification of more quantitative properties (e.g. compliance with resource availability), as well as a more detailed and precise verification of the abovementioned topological ones. Existing SDN verification tools could be deployed in this context, but the majority of them only operate on flow space rules commonly expressed using OpenFlow syntax.

Moreover, such verification tools were designed for networks where the flow rules are necessary and sufficient to determine the forwarding state. This assumption is valid in networks composed only by network functions that forward traffic by analyzing only the packet headers (e.g. simple routers, stateless firewalls, etc.). Unfortunately, most of the real networks contain active network functions, represented by middle-boxes that dynamically change the forwarding path of a flow according to function-local algorithms and an internal state (that is based on the received packets), e.g. load balancers, packet marking modules and intrusion detection systems. The existing verification tools do not consider active network functions because they do not account for the dynamic transformation of an internal state into the verification process.

Defining a set of verification tools that can account for active network functions is a significant challenge. In order to perform verification based on formal properties of the system, the internal states of an active (virtual or not) network function would need to be represented. Although these states would increase the verification process complexity (e.g., using simple model checking would not be feasible due to state explosion), they help to better represent the forwarding behavior in real networks. A way to address this challenge is by attempting to summarize the internal state of an active network function in a way that allows for the verification process to finish within a reasonable time interval.

9. Testing Challenges

Testing in an NFV environment does impact the methodology used. The main challenge is the ability to isolate the Device Under Test (DUT). When testing physical devices, which are dedicated to a specific function, isolation of this function is relatively simple: isolate the DUT by surrounding it with emulations from test devices. This achieves isolation of the DUT, in a black box fashion, for any type of testing. In an NFV environment, the DUT become a component of a software infrastructure which can't be isolated. For example, testing a VNF can't be achieved without the presence if the NFVI and MANO components. In addition, the NFVI and MANO components can greatly influence the behavior and the performance of the VNF under test.

With this in mind, in NFV, the isolation of the DUT becomes a new concept: the VNF Under Test (VUT) becomes part of an environment that consists of the rest of the necessary architecture components (the test environment). In the previous example, the VNF becomes the VUT, while the MANO and NFVI become the test environment. Then, isolation of the VUT becomes a matter of configuration management, where the configuration of the test environment is kept fixed for each test of the VUT. So the MANO policies for instantiation, scaling, and placement, as well as the NFVI parameters such as HW used, CPU pinning, etc must remained fixed for each iterative test of the VNF. Only by keeping the configurations constant can the VNF tests can be compared to each other. If any test environment configurations are changed between tests, the behavior of the VNF can be impacted, thus negating any comparison of the results.

Of course, there are instances of testing where the inverse is desired: the configuration of the test environment is changed between each test, while the VNF configuration is kept constant. As an example, this type of methodology would be used in order to discover the optimum configuration of the NFVI for a particular VNF workload. Another similar but daunting challenge is the introduction of co-located tenants in the same environment as the VNF under test. The workload on these "neighbors" can greatly influence the behavior and performance of the VNF under test, but the test itself is invaluable to understand the impact of such a configuration.

Another challenge is the usage of test devices (traffic generator, emulator) that share the same infrastructure as the VNF under test. This can create a situation as above, where the neighbor competes for resources with the VUT itself, which can really negate test results. If a test architecture such as this is necessary (testing east-west traffic, for example), then care must be taken to configure the test devices such as they are isolated from the SUT in terms of allowed

resources, and that they don’t impact the SUT’s ability to acquire resources to operate in all conditions.

NFV offers new features that didn’t exist as such previously, or modifies existing mechanisms. Examples of new features are dynamic scaling of VNFs and network services (NS), standardized acceleration mechanisms and the presence of the virtualization layer, which includes the vSwitch. An example mechanism which changes with NFV how fault detection and fault recovery are handled. Fault recovery could now be handled by MANO in such a way to invoke mechanisms such as live migration or snapshots in order to recover the state of a VNF and restore operation quickly. While the end results are expected to be the same as before, since the mechanism is very different, rigorous testing is highly recommended to validate those results.

Dynamic scaling of VNFs is a new concept in NFV. VNFs that require more resources will have them dynamically allocated on demand, and then subsequently released when not needed anymore. This is clearly a benefit arising from SDI. For each type of VNF, specific metrics will be used as input to conditions that will trigger a scaling operation, orchestrated by MANO. Testing this mechanism requires a methodology tailored to the specific operation of the VNF, in order to properly reach the monitored metrics and exercise the conditions leading to a scaling trigger. For example, a firewall VNF will be triggered for scaling on very different metrics than a 3GPP MME. Both VNFs accomplish different functions. Since there will normally be a collection of metrics that are monitored in order to trigger a scaling operation, the testing methodology must be constructed in such a way as to address all combinations of those metrics. Metrics for a particular VNF may include sessions, session instantiations/second, throughput, etc. These metrics will be observed in relation to the given resources for the VNF.

10. Programmable management

The ability to automate a set of actions to be performed on the infrastructure, be it virtual or physical, is key to productivity increases following the application of DevOps principles. Previous sections in this document touched on different dimensions of programmability:

- Section 5 approached programmability in the context of developing new capabilities for monitoring and for dynamically setting configuration parameters of deployed monitoring functions

- Section 7 reflected on the need to determine the correctness of actions that are to be inflicted on the infrastructure as result of executing a set of high-level instructions

- Section 8 considered programmability in the perspective of an interface to facilitate dynamic orchestration of troubleshooting steps towards building workflows and for reducing the manual steps required in troubleshooting processes

We expect that programmable network management - along the lines of [RFC7426] - will draw more interest as we move forward. For example, in [I-D.unify-nfvrg-challenges], the authors identify the need for presenting programmable interfaces that accept instructions in a standards-supported manner for the Two-way Active Measurement Protocol (TWAMP)TWAMP protocol. More specifically, an excellent example in this case is traffic measurements, which are extensively used today to determine SLA adherence as well as debug and troubleshoot pain points in service delivery. TWAMP is both widely implemented by all established vendors and deployed by most global operators. However, TWAMP management and control today relies solely on diverse and proprietary tools provided by the respective vendors of the equipment. For large, virtualized, and dynamically instantiated infrastructures where network functions are placed according to orchestration algorithms proprietary mechanisms for managing TWAMP measurements have severe limitations. For example, today’s TWAMP implementations are managed by vendor-specific, typically command-line interfaces (CLI), which can be scripted on a platform-by-platform basis. As a result, although the control and test measurement protocols are standardized, their respective management is not. This hinders dramatically the possibility to integrate such deployed functionality in the SP-DevOps concept. In this particular case, recent efforts in the IPPM WG [I-D.cmzrjp-ippm-twamp-yang] aim to define a standard TWAMP data model and effectively increase the programmability of TWAMP deployments in the future.

Data center DevOps tools, such as those surveyed in [D4.1], developed proprietary methods for describing and interacting through interfaces with the managed infrastructure. Within certain communities, they became de-facto standards in the same way particular CLIs became de-facto standards for Internet professionals. Although open-source components and a strong community involvement exists, the diversity of the new languages and interfaces creates a burden for both vendors in terms of choosing which ones to prioritize for support, and then developing the functionality and operators that determine what fits best for the requirements of their systems.

11. Security Considerations

DevOps principles are typically practiced within the context of a single organization ie a single trust domain. Extending DevOps practices across strong organizational boundaries (e.g. between commercial organizations) requires consideration of additional threat models. Additional validation procedures may be required to ingest and accept code changes arising from outside an organization.

12. IANA Considerations

This memo includes no request to IANA.

13. References

13.1. Informative References

[NFVMANO] ETSI, "Network Function Virtualization (NFV) Management and Orchestration V0.6.1 (draft)", Jul. 2014

[I-D.aldrin-sfc-oam-framework] S. Aldrin, R. Pignataro, N. Akiya. "Service Function Chaining Operations, Administration and Maintenance Framework", draft-aldrin-sfc-oam-framework-02, (work in progress), July 2015.

[I-D.lee-sfc-verification] S. Lee and M. Shin. "Service Function Chaining Verification", draft-lee-sfc-verification-00, (work in progress), February 2014.

[RFC7426] E. Haleplidis (Ed.), K. Pentikousis (Ed.), S. Denazis, J. Hadi Salim, D. Meyer, and O. Koufopavlou, "Software Defined Networking (SDN): Layers and Architecture Terminology", RFC 7426, January 2015

[RFC7149] M. Boucadair and C Jaquenet. "Software-Defined Networking: A Perspective from within a Service Provider Environment", RFC 7149, March 2014.

[TR228] TMForum Gap Analysis Related to MANO Work. TR228, May 2014

[I-D.unify-nfvrg-challenges] R. Szabo et al. "Unifying Carrier and Cloud Networks: Problem Statement and Challenges", draft-unify-nfvrg-challenges-03 (work in progress), October 2016

[I-D.cmzrjp-ippm-twamp-yang] Civil, R., Morton, A., Zheng, L., Rahman, R., Jethanandani, M., and K. Pentikousis, "Two-Way Active Measurement Protocol (TWAMP) Data Model", draft-cmzrjp-ippm-twamp-yang-02 (work in progress), October 2015.

[D4.1] W. John et al. D4.1 Initial requirements for the SP-DevOps concept, universal node capabilities and proposed tools, August 2014.

[SDNsurvey] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. Esteve Rothenberg, S. Azodolmolky, S. Uhlig. "Software-Defined Networking: A Comprehensive Survey." To appear in proceedings of the IEEE, 2015.

[DevOpsP] "DevOps, the IBM Approach" 2013. [Online].

[Y1564] ITU-R Recommendation Y.1564: Ethernet service activation test methodology, March 2011

[CAP] E. Brewer, "CAP twelve years later: How the "rules" have changed", IEEE Computer, vol.45, no.2, pp.23,29, Feb. 2012.

[H2014] N. Handigol, B. Heller, V. Jeyakumar, D. Mazieres, N. McKeown; "I Know What Your Packet Did Last Hop: Using Packet Histories to Troubleshoot Networks", In Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), pp.71-95

[W2011] A. Wundsam, D. Levin, S. Seetharaman, A. Feldmann; "OFRewind: Enabling Record and Replay Troubleshooting for Networks". In Proceedings of the Usenix Anual Technical Conference (Usenix ATC ’11), pp 327-340

[S2010] E. Al-Shaer and S. Al-Haj. "FlowChecker: configuration analysis and verification of federated Openflow infrastructures" In Proceedings of the 3rd ACM workshop on Assurable and usable security configuration (SafeConfig ’10). Pp. 37-44

[OSandS] S. Wright, D. Druta, "Open Source and Standards: The Role of Open Source in the Dialogue between Research and Standardization" Globecom Workshops (GC Wkshps), 2014 , pp.650,655, 8-12 Dec. 2014

[C2015] CFEngine. Online: http://cfengine.com/product/what-is-cfengine/, retrieved Sep 23, 2015.

[P2015] Puppet. Online: http://puppetlabs.com/puppet/what-is-puppet, retrieved Sep 23, 2015.

[A2015] Ansible. Online: http://docs.ansible.com/ , retrieved Sep 23, 2015.

[AK2015] Apache Kafka. Online: http://kafka.apache.org/documentation.html, retrieved Sep 23, 2015.

[S2015] Splunk. Online: http://www.splunk.com/en_us/products/splunk-light.html , retrieved Sep 23, 2015.

[K2014] J. Kreps. Benchmarking Apache Kafka: 2 Million Writes Per Second (On Three Cheap Machines). Online: https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines, retrieved Sep 23, 2015.

[R2015] RabbitMQ. Online: https://www.rabbitmq.com/ , retrieved Oct 13, 2015

[IFA014] ETSI, Network Functions Virtualisation (NFV); Management and Orchestration Network Service Templates Specification , DGS/NFV-IFA014, Work In Progress

[IFA011] ETSI, Network Functions Virtualisation (NFV); Management and Orchestration; VNF Packaging Specification, DGS/NFV-IFA011, Work in Progress

[NFVSWA] ETSI, Network functions Virtualisation; Virtual Network Functions Architecture, GS NFV-SWA 001 v1.1.1 (2014)

[Z2015] ZeroMQ. Online: http://zeromq.org/ , retrieved Oct 13, 2015

14. Contributors to earlier versions

J. Kim (Deutsche Telekom), S. Sharma (iMinds), I. Papafili (OTE)

15. Acknowledgments

The research leading to these results has received funding from the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement no. 619609 - the UNIFY project. The views expressed here are those of the authors only. The European Commission is not liable for any use that may be made of the information in this document.

We would like to thank in particular the UNIFY WP4 contributors, the internal reviewers of the UNIFY WP4 deliverables and Russ White and Ramki Krishnan for their suggestions.

This document was prepared using 2-Word-v2.0.template.dot.

16. Authors' Addresses

Catalin Meirosu
Ericsson Research
S-16480 Stockholm, Sweden
Email: catalin.meirosu@ericsson.com

Antonio Manzalini
Telecom Italia
Via Reiss Romoli, 274
10148 - Torino, Italy
Email: antonio.manzalini@telecomitalia.it

Rebecca Steinert
SICS Swedish ICT AB
Box 1263, SE-16429 Kista, Sweden
Email: rebste@sics.se

Guido Marchetto
Politecnico di Torino
Corso Duca degli Abruzzi 24
10129 - Torino, Italy
Email: guido.marchetto@polito.it

Kostas Pentikousis
Travelping GmbH
Koernerstrasse 7-10
Berlin 10785
Germany
Email: k.pentikousis@travelping.com

Steven Wright
AT&T Services Inc.
1057 Lenox Park Blvd NE, STE 4D28
Atlanta, GA 30319
USA
Email: sw3588@att.com

Pierre Lynch
Ixia
800 Perimeter Park Drive, Suite A
Morrisville, NC 27560

USA
Email: plynch@ixiacom.com

Wolfgang John
Ericsson Research
S-16480 Stockholm, Sweden
Email: wolfgang.john@ericsson.com