

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 18, 2016

M. Bagnulo
UC3M
D. Dolson
Sandvine
March 17, 2016

NFVI PoP Network Topology: Problem Statement
draft-bagnulo-nfvrg-topology-01

Abstract

This documents describes considerations for the design of the interconnection network of an NFVI PoP.

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 18, 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. Considerations for the design of the NFVI PoP network topology	3
2.1. External links	3
2.2. Number of servers	3
2.3. Traffic patterns	4
2.3.1. Macroscopic behaviour	4
2.3.2. Traffic pattern within the PoP	5
2.4. Technological considerations	8
2.4.1. Direct and Indirect networks	8
2.4.2. SFC technology	8
2.4.3. Network Virtualization Technology	8
2.4.4. Software or Hardware Switching	9
3. Design goals	9
3.1. Effective load	9
3.2. Latency	10
3.3. Scalability	10
3.4. Fault Tolerance	11
3.5. Cost	12
3.6. Backward compatibility	12
4. Topologies	12
5. Security considerations	12
6. IANA Considerations	12
7. Acknowledgments	12
8. Informative References	13
Authors' Addresses	13

1. Introduction

An NFVI PoP is defined as a "single geographic location where a number of NFVI-Nodes are sited" where an NFVI-Node is "a physical device deployed and managed as a single entity providing the NFVI functions required to support the execution environment for VNFs" [ETSI_GS_NFV-INF_001]. In other words, an NFVI PoP is the premises where the processing, storage and networking resources (i.e., servers and switches) used to execute the network virtual functions (VNFs) are deployed. The servers and switches in a NFVI PoP will be interconnected forming the NFVI PoP interconnection network. The goal of this document is to explore the different design considerations for the NFVI PoP interconnection network topology, including design goals and constraints.

The NFVI PoP is essentially a data center, and the NFVI PoP interconnection network is essentially a data center network. As such it is only natural to use the current state of the art in data

center networking as a starting point for the design of the NFVI PoP network.

2. Considerations for the design of the NFVI PoP network topology

This section describes different pieces of information that are relevant input for the design of the NFVI PoP network topology. In some cases, the information is known (and sometimes ready available), while in other cases, the information is not known at this stage.

2.1. External links

The NFVI PoP is part of the operator's infrastructure and as such it is connected to the rest of the operator's network. Information about the number of links and their respective capacity is naturally a required in order to properly design the NFVI PoP topology. Different types of PoPs have different number of links with different capacity to connect to the rest of the network. In particular, the so-called "local PoPs" that connect the links from end users (either DSL lines or FTTH or else) and also connect to the rest of the operator's network. The "regional" PoPs or "regional data centers" have links to the "local PoPs" and to other "regional PoPs" and other parts of the operator's infrastructure.

For instance, a local PoP in a DSL access network can have between 15.000 and 150.000 DSL lines with speeds between 10 Mbps and 100 Mbps and tens of links to the core network of the operator with links between 20 Gbps and 80 Gbps.

It would be useful to confirm these numbers and to have information about other types of PoPs.

2.2. Number of servers

While knowing the exact number of servers is not required to design the PoP network topology, knowing the order of the number of servers is at least useful. If the resulting topology have tens of servers, then the topology is likely to be very simple (e.g., a tree-like topology with access/aggregation/core switches may be suitable). On the other hand, if the topology should encompass several hundreds of servers or even a few thousands of servers, then the problem is more challenging as we are likely to reach the available capacity of existing switches and more sophisticated topologies may be required.

The number of servers on a PoP depends on several factors, including the number and capacity of external links (i.e., the offered load to the PoP), the number and type of Virtual Network Functions that will be provided by the PoP, the performance of the VNF implementations

and the number and length of service function chains that will be provided.

The number of external links is discussed in the previous section. The number and capacity of the external links is relevant to determine the number of servers because they will carry the load offered to the PoP. In other words, traffic coming through the external links will require processing by the different VNF hosted in the servers, influencing the number of servers needed.

The number of different VNFs provided in the PoP as well as the number and length of service functions chains provided in the PoP will also influence the number of servers required in the PoP. The more demanding the VNFs provided, the more servers will be needed to provide it and the longer the service function chain a higher number of servers will be required to support it.

Finally, the performance of the NFV implementations also affects the number of servers required in a PoP. In particular, some VNF implementations are capable of processing at line speed, while other implementations of other VNFs are not capable of that, requiring additional servers to provide the VNF for the same line speed. While there is some initial work assessing the performance of the different VNFs (e.g., [swBRAS]), still more work is needed to have a full picture for the different VNFs at different line speeds.

Overall, we need to have a rough estimate of the range of the number servers that will be part of the PoP network in order to provide a successful design and we need to take into account the aforementioned considerations to obtain it.

2.3. Traffic patterns

The pattern of the expected traffic of the NFVI PoP network is of course essential to properly design the network topology. In this section we describe different characteristics of the traffic pattern that we believe are relevant and that it would be useful to have information about.

2.3.1. Macroscopic behaviour

There are essentially 4 types of traffic direction within a NFVI PoP network, namely, cross-through traffic, intra-PoP traffic, PoP-generated traffic and PoP-terminated traffic, as depicted in Figure 1.

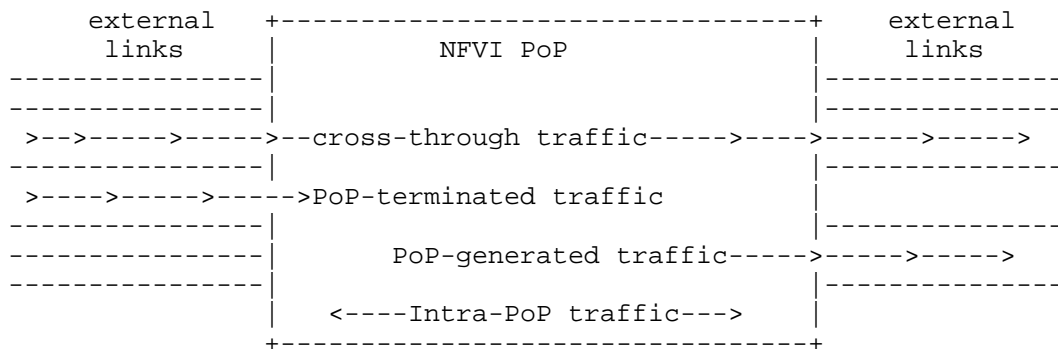


Figure 1: Types of traffic in NFVI PoP network

The cross-through traffic is the traffic that reaches the PoP through an external link, it is processed by a service function chain (i.e., it is processed by a number of VNFs inside the PoP) and then is forwarded through an external link. Processing such type of traffic is one of the main purposes of the PoP since the PoP is part of the operator's infrastructure whose main purpose is to forward user's traffic.

The PoP-generated traffic is generated by VNFs located within the PoP. An example of such VNF would be a cache located inside the PoP which serves content to users. Similarly, PoP-terminated traffic is external traffic that is terminated by one VNF located inside the PoP for example a firewall.

Finally, Intra-PoP traffic is traffic generated and terminated inside the PoP that never leaves the PoP. This traffic includes much of the management traffic, deploying and moving virtual machines and VNFs across different servers and other signaling traffic (e.g., the one associated with voice calls).

In order to properly design the PoP network topology, it is relevant to know the distribution of the expected traffic in these categories.

2.3.2. Traffic pattern within the PoP

The traffic within the PoP will be composed of essentially two types of traffic:

1. The traffic served by the PoP. This is the traffic coming from and/or going to external links and that should traverse a number of servers where the different VNFs are placed. This includes the cross-through traffic, the PoP-generated traffic and the PoP-terminated traffic.

2. The operation and management traffic that includes all the traffic resulting from the management of the virtual machines and the VNFs, as well as signaling traffic required to provides the VNFs. This is the Intra-PoP traffic.

The traffic pattern of the traffic served by the PoP is basically determined by the location of the input link, the location of the output link and the mapping of the service function chain to servers.

2.3.2.1. Mapping of service function chains to servers

There are multiple possible strategies to deploy VNFs and SFCs in servers.

- o Parallel SFC deployment strategy: One possible approach is to deploy all the VNFs of a given service function chain in a single server and deploy as many of these servers in parallel in order to server the different flows. When more flows arrive to the PoP, more servers are used in parallel.
- o Sequential SFC deployment strategy: Another possible approach would be to deploy each VNF in a different server and have one (or more) servers dedicated to process this particular VNF for all the flows of the PoP. When the number of flows increases, the number of servers providing each VNF is also increased.
- o Hybrid strategy: it is also possible to use a hybrid strategy, where several VNFs of the SFC are deployed together in a server and other VNFs of the SFC are deployed in separated servers.

There are many factors that influence this decision, including the performance of the implementation of the VNF (maybe the VNF is too demanding to be executed with other VNFs in the same server) or licensing conditions (maybe some VNF licenses are based on the number of servers deployed, while maybe others depend on the number of users served, or even the time the VNF is being executed).

In any case, to design the PoP topology it would be relevant to know:

The number of servers that the traffic served by the PoP will traverse (which is determined by the length of the SFCs and the deployment strategy of SFCs in servers).

The number of different SFCs that will be simultaneously available in the PoP at any point in time. At any point in time, different flows coming from a particular external link can be served by one or more different SFCs. These SFCs can be mapped to different sequences of servers. Depending on this, different flows coming

from any external links will have to traverse different sequences of servers, affecting the Intra-PoP traffic pattern.

2.3.2.2. Locality

There are two locality aspects that affect the pattern of the traffic served by the PoP. First, whether the servers providing the different VNFs of each SFC can be assumed to be topologically close (e.g., in the same rack). If the SFCs that process the majority of the flows can be assumed to be topologically close, topologies that exploit locality can be useful.

The other locality related aspect that affects the topology design is the distribution of output links of the traffic arriving through the different input links. Consider the case of a Local PoP, which has links connecting to users (DSL, FTTH, etc) and links connecting to the rest of the provider's network. Let's call the first type of links user's links and the second type of links, core links. It is reasonable to assume that most of the traffic coming from a user's link will go to a core link and vice-versa. We can expect that the traffic between two user's links will be low and the same for the traffic between two core links. If we now consider the case of a regional PoP, it is not so clear we can make such assumption about the traffic between links. In case this assumption can be made, it would be possible to design the topology to pair user's link with core link to optimize the transit between them.

2.3.2.3. Churn

There is also the question about how often the provided SFCs will change and frequently VNFs and virtual machines will be deployed in servers. This affects the amount of churn traffic in the PoP. There may be more to it...?

2.3.2.4. Growth

Another relevant aspect is the expected growth in terms of offered load to the PoP and also in terms of VNFs in the PoP. We should understand if the capacity of the PoP is expected to increase linearly or exponentially in time. Similarly, we need to understand if the number of VNFs and the length of the SFCs will remain more or less constant or will evolve. If it does evolve, which is the expected pace. The reason for this is that different topologies support growth in different manners so depending on the expectation in this aspects, different topologies may be more or less suitable.

2.4. Technological considerations

2.4.1. Direct and Indirect networks

A network is called an Indirect network there are two types of nodes, nodes that source/sink traffic and nodes that forward traffic. A network is called a Direct network if every node plays both roles. Usually data center networks are Indirect networks, with switches that forward packets and servers that source/sink packets. While there have are proposals that use both switches are servers to forward packets (e.g., [BCube]), the main concern expressed against them is that the resources available in the servers should be used to execute applications (which is the final goal of the data center) rather than be used in forwarding packets.

In the case of an NFVI PoP network, the actual purpose of the servers is in many cases to forward packets through the VNFs provided by the server, so it may make perfect sense to use servers to forward packets. From this perspective, either direct networks or networks that use both switches and servers to forward packets may be attractive for NFVI PoPs.

2.4.2. SFC technology

Service Function Chaining can be accomplished using the IETF SFC protocol [I-D.ietf-sfc-architecture] or using a SDN approach, where a controller instructs the switches where to forward the different flows using Openflow. The two approaches have a different architecture with different components and it is possible that different topologies accommodate more naturally the elements of the different SFC architectures.

If using an OpenFlow approach, determine what form the rules take, consider when forwarding rules must be dynamically updated due to the arrival of new flows, and determine what peak update rate is required. Evaluate the SDN switches against all of these requirements.

2.4.3. Network Virtualization Technology

Technologies exist to improve performance of NFV functions, including PCI-passthrough, SRIOV and NUMA-aware process pinning. Other technologies are likely to become available in the future to offload network function.

Consider selecting infrastructure with these features if the NFV functions can utilize them and if the orchestration and control-plane infrastructure can configure them optimally.

Performance of individual NFV implementations may vary by an order of magnitude with or without the hardware features, so PoP planning and sizing must include consideration of how the functions fit with the hardware and whether the infrastructure can deploy virtual machines in a manner that allows the hardware to be used.

2.4.4. Software or Hardware Switching

Some network architectures require software switches (such as OpenVSwitch), whereas other architectures only require top-of-rack and backplane Ethernet switching.

Although software switches perform very well, they consume processing cores. PoP design must consider how many processor cores are required for software switches.

3. Design goals

In this section we describe the goals for the design of a NFVI PoP network topology. In broad terms, they include scalability, performance, costs, fault tolerance, operation, management and backward compatibility

3.1. Effective load

A first performance parameter that we should take into account when considering different topologies is the effective load supported by the network. The main goal of the NFVI PoP is to forward traffic between the different external links connected to the PoP. The performance of the PoP is measured by the traffic it is able to forward i.e., the more effective load it manages. In order to assess the effective load supported by the different topologies, we increase the offered load coming to the PoP through the different links and we measure the effective load that the PoP is able to deliver.

The effective load supported by a topology is likely to be affected by multiple factors, including the the different aspects we described in the traffic patterns section Section 2.3 (such as the traffic matrix between the different external links, the characteristics of the SFCs and so on), the routing inside the PoP, the different locality considerations, and the intra-PoP traffic. Moreover, in order for the comparison of two topologies to make sense, they need to be "equal" in some other dimension (e.g., cost, number of servers, number of links, number of switches or else).

For example, as a starting point, we can assume a purely random traffic matrix, i.e., every packet arriving through an external link is forwarded through n random servers in the topology and exits

through a randomly picked external link, and assume shortest-path, equal cost multi-path routing. We can compare different topologies with the same number N of servers. We perform the comparison by measuring the effective load when increasing the offered load and for different values of N and n . Of course, these conditions may greatly differ from the real operation condition, this is why it is useful to have information about the items described in section Section 2.

When performing this evaluation, it is useful to also measure the packet loss and to track the occurrences of hot-spots in the topology, in order to identify the bottlenecks of the topology which may be useful to improve it.

Related to this, it may be useful to consider the bisection bandwidth of the different topologies.

3.2. Latency

Another relevant performance indicator is the latency suffered by packets while traversing the PoP network. That is, for a topology of N servers, which is the latency for a packet that arrives through an external link, traverses n servers and exits through an external link. Since we only care about the latency cause by the topology itself (in order to assess the topology) we can measure the "latency" as the number of hops that the packet should traverse.

It is useful to measure the mean latency, but also the maximum latency, since an upper bound for the time a packet stays in the PoP is also relevant. Again, the latency/Hop count depends on the traffic matrix (i.e., the relation of the input and output links), the routing and the different locality aspects, hence it is useful to have information about these aspects. In any case, a purely random case as the one described for the effective load measurement could be used as a starting point.

Queuing between software elements can introduce latency, so it is important to include extra hops caused by software components (such as software switches) that may be required to deliver packets to virtual machines from physical interfaces, in contrast to technologies (e.g., SRIOV) that allow virtual machines to receive traffic directly from network interface cards.

3.3. Scalability

Scalability refers to how well the proposed topology supports the growth in terms of number of servers, line speed of the servers and capacity of the external links. there are some topologies that in order to support an increased number of servers require growing some

components beyond what is technically feasible (or what is economically efficient). For instance it is well known that tree topologies require the core switches to grow in order to support more servers, which is not feasible beyond certain point (or it becomes very expensive). That being said, we should consider scalability in the range of servers that we expect that a PoP will have to support in a reasonable time frame.

Another dimension to consider is the size of forwarding tables required by switches in the network. E.g., do the switches have capacity to learn the required number of MAC addresses? Some service chaining technologies utilize many private Ethernet addresses; is there capacity to learn the number that are required? The same reasoning should be applied to whichever types of forwarding tables are required, whether IP routing, MPLS, NSH, etc.

Another aspect somehow related to scalability is how well the different topologies support incremental growth. It is unclear at this point which will be the growth pace for the NFVI PoPs. In other words, given that we have a PoP with N servers operational, then next time we need to increase the number of servers, will it increase to $N+1$, to $2*N$ or to $N*N$? Different topologies have different growth models. Some support growing lineally indefinitely, others can be over-dimensioned in order to support some linear growth, but after a given number of additional servers, they need to grow exponentially.

3.4. Fault Tolerance

Fault tolerance is of course paramount for an NFVI PoP network. So, when considering topologies, we must consider fault tolerance aspects. We basically care about how well the topology handles link failures, switch failures and server failures.

We can assess the fault tolerance of topology by measuring the following parameters of the topology [DC-networks]:

- o Node-disjoint paths: The minimum of the number of paths that share no common intermediate nodes between any arbitrary servers.
- o Edge disjoint paths: The minimum of the total of number of paths that share no common edges between any arbitrary servers.
- o f -fault tolerance: A network is f -fault tolerant if for any f failed components, the network is still connected.
- o Redundancy level: A network has redundancy level of r if and only if after removing any set of r components, it remains connected

and exists a set of $r+1$ components such that after removing them, the network is no longer connected.

3.5. Cost

The cost of the resulting network is also a relevant aspect to be consider. In order to assess the cost, we can consider the number of switches and the number of interfaces in topology for the same number of servers. We should also take into account that type of switches required, as we know that the cost of a switch does not scale linearly with the number of interfaces of the switch and with the speed of the interfaces.

3.6. Backward compatibility

Another relevant aspect to consider is compatibility with existent hardware. It is unlikely that operators will throw away all their current infrastructure based on specialized hardware and replace it for VNFs running in COTS servers. It is more likely that there will be an incremental deployment where some functions will be virtualized and some function will be executed in hardware. It is then important to consider how the different topologies support such hybrid scenarios.

4. Topologies

In this section, we plan to describe different topologies that have been proposed for data centers and include some considerations about the different design goals described in section Section 3.

5. Security considerations

TBD, not sure if there is any.

6. IANA Considerations

There are no IANA considerations in this memo.

7. Acknowledgments

We would like to thank Bob Briscoe, Pedro Aranda, Diego Lopez, Al Morton, Joel Halpern and Costin Raiciu for their input. Marcelo Bagnulo is partially funded by the EU Trilogy 2 project.

8. Informative References

- [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.
- [ETSI_GS_NFV-INF_001]
., ETSI., "Network Functions Virtualisation (NFV); Infrastructure Overview", NFV ISG, 2015.
- [swBRAS] Bifulco, R., Dietz, T., Huici, F., Ahmed, M., and J. Martins, "Rethinking Access Networks with High Performance Virtual Software BRASes", EWSDN 2013, 2013.
- [BCube] Guo, C., Lu, G., Li, D., Wu, H., and X. Zhang, "BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers", SIGCOMM 2009, 2009.
- [DC-networks]
Liu, Y., Muppala, J., Veeraraghavan, M., Lin, D., and M. Hamdi, "Data Center Networks - Topologies, Architectures and Fault-Tolerance Characteristics", Springer Briefs in Computer Science Springer 2013, 2013.

Authors' Addresses

Marcelo Bagnulo
Universidad Carlos III de Madrid
Av. Universidad 30
Leganes, Madrid 28911
SPAIN

Phone: 34 91 6249500
Email: marcelo@it.uc3m.es
URI: <http://www.it.uc3m.es>

David Dolson
Sandvine
408 Albert Street
Waterloo, ON N2L 3V3
Canada

Phone: +1 519 880 2400
Email: ddolson@sandvine.com

NFVRG
Internet-Draft
Intended status: Informational
Expires: September 22, 2016

CJ. Bernardos
UC3M
A. Rahman
JC. Zuniga
InterDigital
LM. Contreras
P. Aranda
TID
March 21, 2016

Gap Analysis on Network Virtualization Activities
draft-bernardos-nfvrg-gaps-network-virtualization-04

Abstract

The main goal of this document is to serve as a survey of the different efforts that have been taken and are currently taking place at IETF and IRTF in regards to network virtualization, automation and orchestration, putting them into context considering efforts by other SDOs, and identifying current gaps and challenges that can be tackled at IETF or researched at the IRTF.

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. Terminology	4
3. Background	5
3.1. Network Function Virtualization	5
3.2. Software Defined Networking	7
3.3. Mobile Edge Computing	11
3.4. IEEE 802.1CF (OmniRAN)	12
3.5. Distributed Management Task Force	12
3.6. Open Source initiatives	12
4. Network Virtualization at IETF/IRTF	14
4.1. SDN RG	14
4.2. SFC WG	14
4.3. NVO3 WG	15
4.4. DMM WG	16
4.5. I2RS WG	17
4.6. BESS WG	18
4.7. BM WG	19
4.8. TEAS WG	20
4.9. I2NSF WG	20
4.10. IPPM WG	21
4.11. NFV RG	22
4.12. VNFpool BoF	22
5. Summary of Gaps	23
6. IANA Considerations	24
7. Security Considerations	25
8. Acknowledgments	25
9. Informative References	25
Appendix A. The mobile network use case	28
A.1. The 3GPP Evolved Packet System	28
A.2. Virtualizing the 3GPP EPS	30
Authors' Addresses	30

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.

Network Function Virtualization (NFV) and Software Defined Networking (SDN) are changing the way the telecommunications sector will deploy, extend and operate their networks. This document provides a survey of the different efforts that have taken and are currently taking place at IETF and IRTF in regards of network virtualization, looking at how they relate to the ETSI NFV ISG, ETSI MEC ISG and ONF architectural frameworks. Based on this analysis, we also go a step farther, identifying which are the potential work areas where IETF/IRTF can work on to complement the complex network virtualization map of technologies being standardized today.

2. Terminology

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

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.

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

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

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.

virtual EPC (vEPC): control plane of 3GPPs EPC operated on NFV framework (as defined by [I-D.matsushima-stateless-uplane-vepc]).

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 Virtualisation Infrastructure (NFVI).

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

3. Background

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. To date, ETSI NFV is by far the most accepted NFV reference framework and architectural footprint [etsi_nfv_whitepaper]. The ETSI NFV framework architecture framework is composed of three domains (Figure 1):

- o Virtualized Network Function, running over the NFVI.
- o NFV Infrastructure (NFVI), including the diversity of physical resources and how these can be virtualized. NFVI supports the execution of the VNFs.
- o 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.

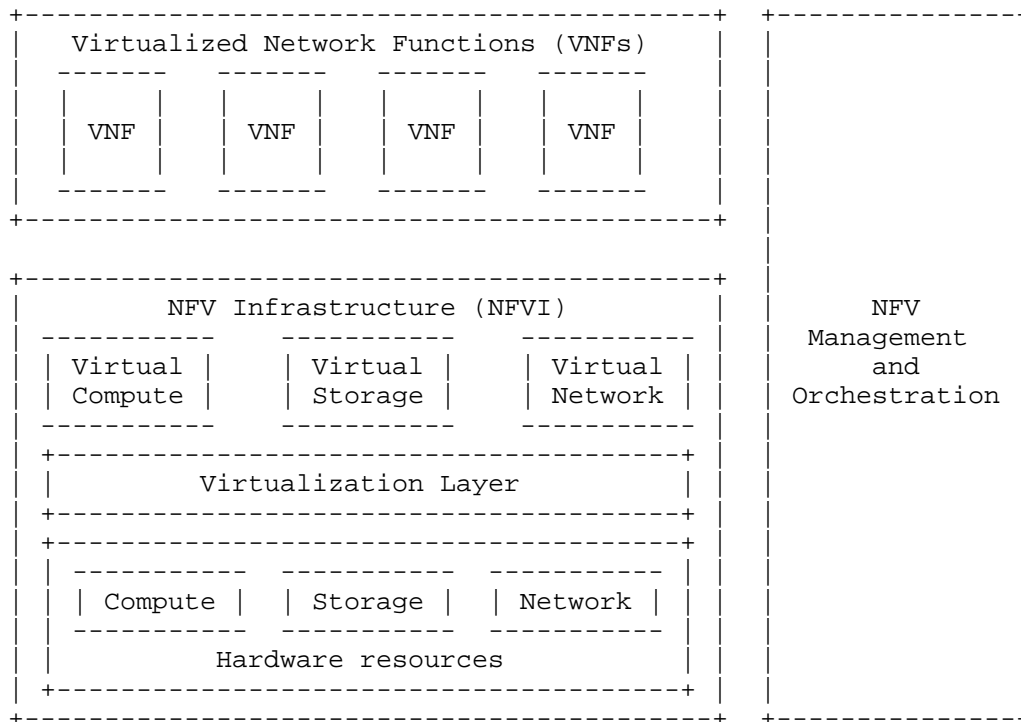


Figure 1: ETSI NFV framework

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):

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

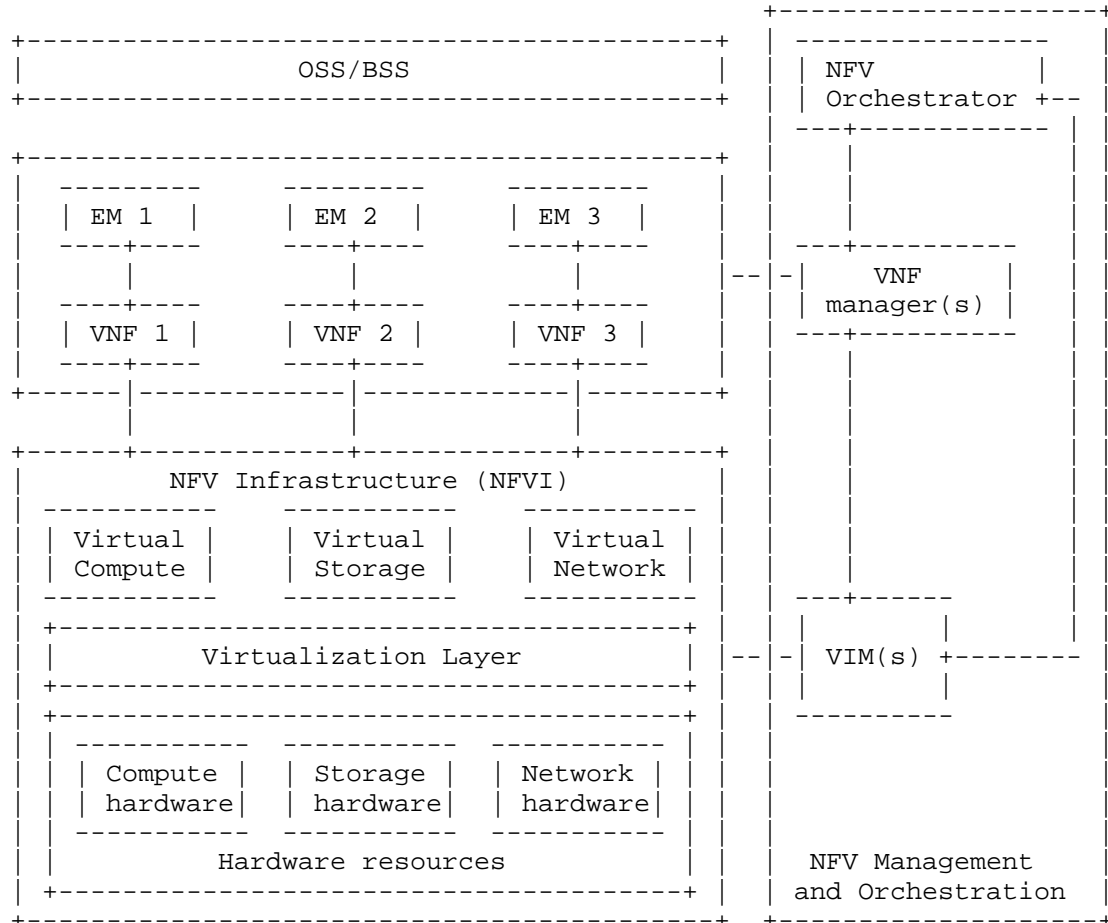


Figure 2: ETSI NFV reference architecture

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 of 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 with to manage their network, i.e., an operator can easily change its policies on how traffic is distributed throughout the network.

The most visible of the SDN protocol stacks 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.

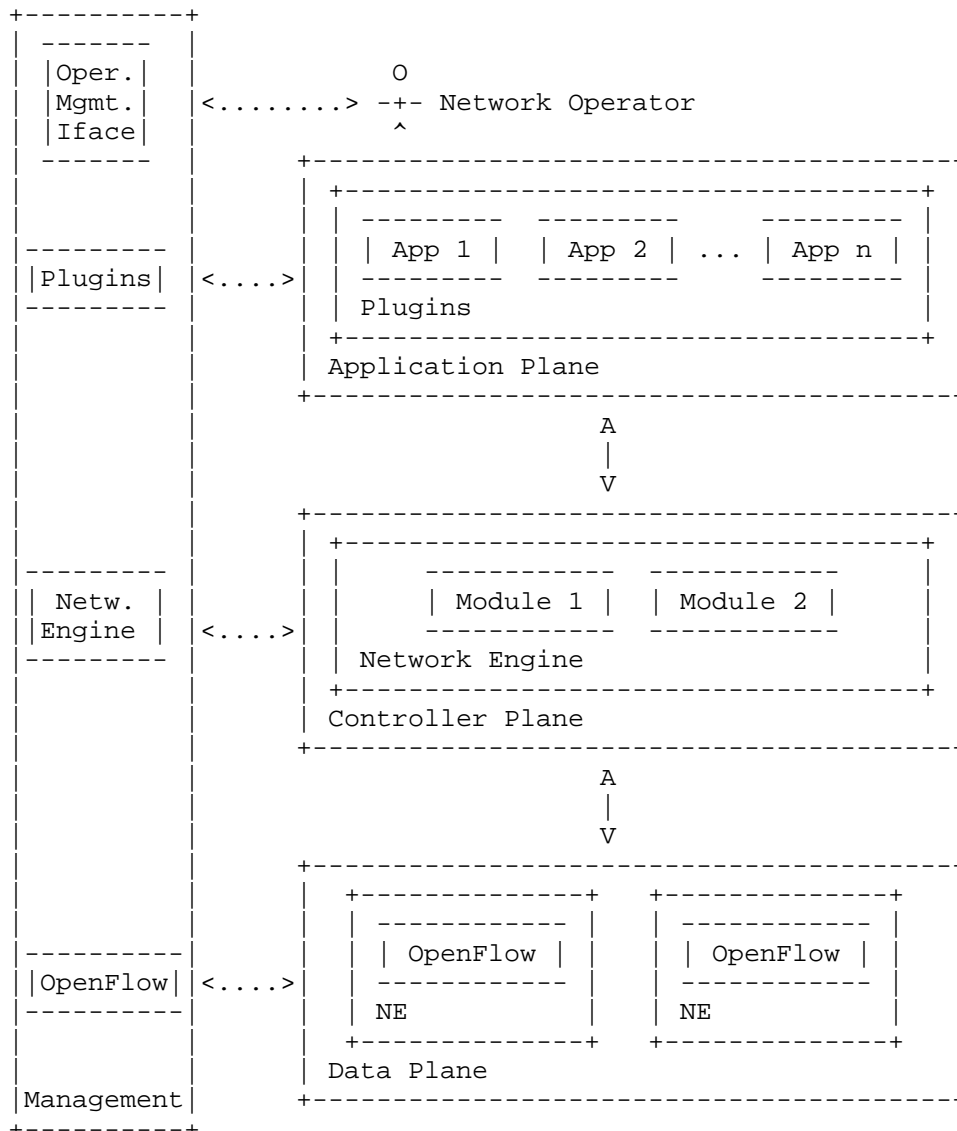


Figure 3: High level SDN ONF architecture

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 underneath 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.

The SDNRG has documented a reference layer model in RFC7426 [RFC7426], which is reproduced in Figure 4. This model structures SDN in planes and layers which are glued together by different abstraction layers. This architecture differentiates between the control and the management planes and provides for differentiated southbound interfaces (SBIs).

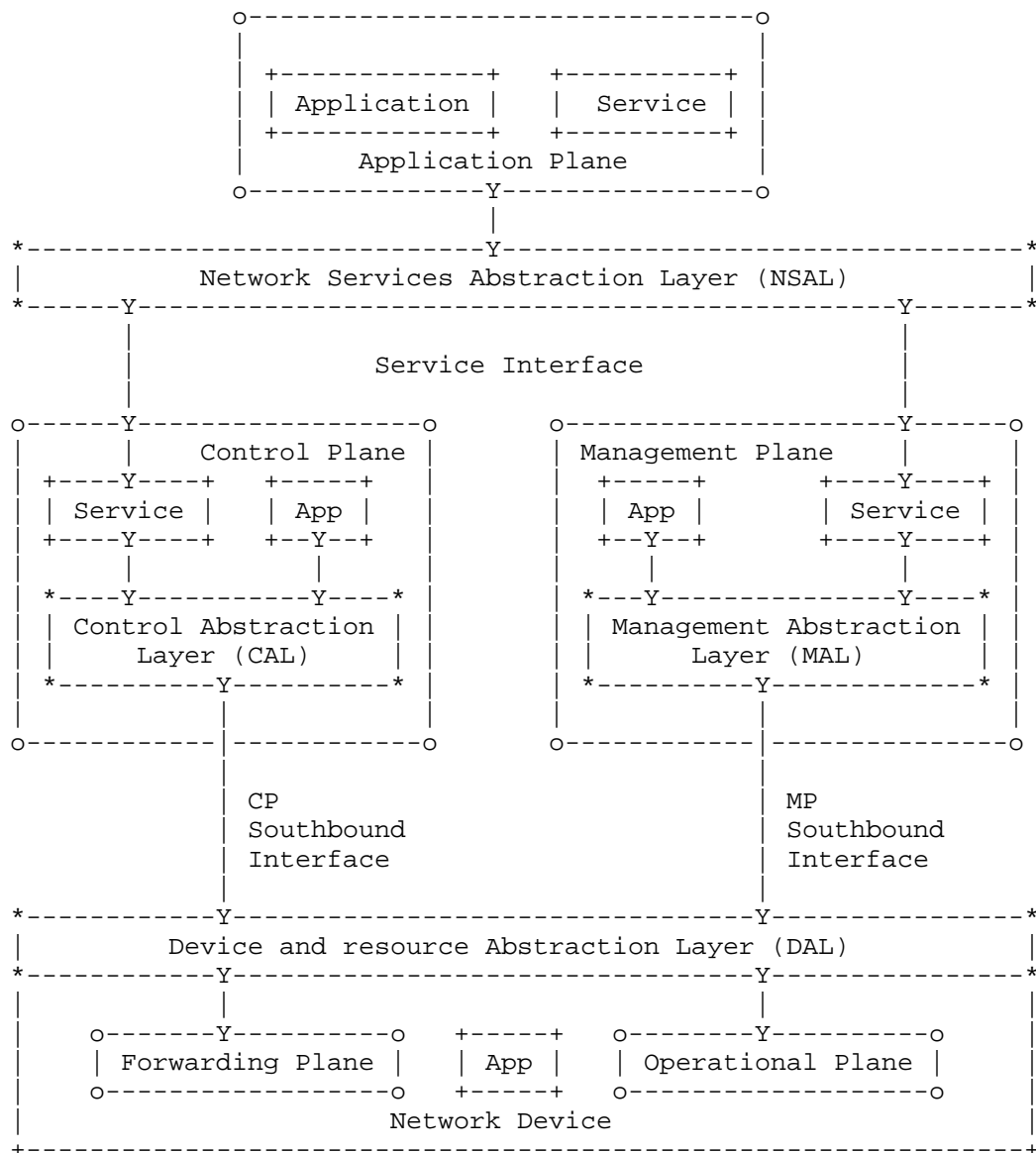


Figure 4: SDN Layer Architecture

3.3. Mobile Edge Computing

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.

3.4. IEEE 802.1CF (OmniRAN)

The IEEE 802.1CF Recommended Practice 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 software defined network (SDN) principles, thereby lowering the barriers to new network technologies, to new network operators, and to new service providers.

3.5. Distributed Management Task Force

The DMTF 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.

3.6. Open Source initiatives

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

- o 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.

- o OpenDayLight. OpenDaylight (ODL) is a highly available, modular, extensible, scalable and multi-protocol controller infrastructure built for SDN deployments on modern heterogeneous multi-vendor networks. It provides a model-driven service abstraction platform that allows users to write apps that easily work across a wide variety of hardware and southbound protocols.
- o 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 software-defined networking (SDN) operating system for communications service providers that is designed for scalability, high performance and high availability.
- o 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.
- o 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 increase performance and power efficiency; improve reliability, availability, and serviceability; and deliver comprehensive platform instrumentation.
- o 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 as Telefonica's OpenMANO or Canonical's Juju, among others.
- o 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.

Among the main areas that are being developed by the former open source activities that related 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 at IETF/IRTF

4.1. SDN RG

The SDNRG provides the grounds for an open-minded investigation of Software Defined Networking. They aim at identifying approaches that can be defined and used in the near term as well as the research challenges in the field. As such, they SDNRG will not define standards, but provide inputs to standards defining and standards producing organizations.

It is working on classifying SDN models, including definitions and taxonomies. It is also studying complexity, scalability and applicability of the SDN model. Additionally, the SDNRG is working on network description languages (and associated tools), abstractions and interfaces. They also investigate the verification of correct operation of network or node function.

The SDNRG has produced a reference layer model RFC7426 [RFC7426], which structures SDNs in planes and layers which are glued together by different abstraction layers. This architecture differentiates between the control and the management planes and provides for differentiated southbound interfaces (SBIs).

4.2. SFC WG

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 necessary on the direct data path. This requires traffic to be steered through the required service functions, wherever they are deployed.

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), 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.).

The SFC working group is working on an architecture for service function chaining 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 chartered to deliver: (i) a problem statement document [RFC7498], (ii) an architecture document [RFC7665], (iii) a service-level data plane encapsulation format (the encapsulation should indicate the sequence of service functions that make up the Service Function Chain, specify the Service Function Path, and communicate context information between nodes that implement service functions and Service Function Chains), and (iv) a document describing requirements for conveying information between control or management elements and SFC implementation points.

Potential gap: as stated in the SFC charter, any work on the management and configuration of SFC components related to the support of Service Function Chaining will not be done yet, until better understood and scoped. This part is of special interest for operators and would be required in order to actually put SFC mechanisms into operation.

Potential gap: redundancy and reliability mechanisms are currently not dealt with by any WG in the IETF. While this has been the main goal of the VNFpool BoF efforts, it still remains un-addressed.

4.3. NVO3 WG

The Network Virtualization Overlays (NVO3) WG is developing protocols that enable network virtualization overlays within large Data Center (DC) environments. Specifically NVO3 assumes an underlying physical Layer 3 (IP) fabric on which multiple tenant networks are virtualized on top (i.e. overlays). With overlays, data traffic between tenants is tunneled across the underlying DC's IP network. The use of tunnels provides a number of benefits by decoupling the network as viewed by tenants from the underlying physical network across which they communicate [I-D.ietf-nvo3-arch].

Potential gap: It would be worthwhile to see if some of the specific approaches developed in this WG (e.g. overlays, traffic isolation, VM

migration) can be applied outside the DC, and specifically if they can be applicable to network virtualization (NFV). These approaches would be most relevant to the ETSI Network Function Virtualization Infrastructure (NFVI), and the Virtualized Infrastructure Manager part of the MANO.

4.4. DMM WG

The Distributed Mobility Management (DMM) WG is looking at solutions for IP networks that enable traffic between mobile and correspondent nodes taking an optimal route, preventing some of the issues caused by the use of centralized mobility solutions, which anchor all the traffic at a given node (or a very limited set of nodes). The DMM WG is considering the latest developments in mobile networking research and operational practices (i.e., flattening network architectures, the impact of virtualization, new deployment needs as wireless access technologies evolve in the coming years) and aims at describing how distributed mobility management addresses the new needs in this area better than previously standardized solutions.

Although network virtualization is not the main area of the DMM work, the impact of SDN and NFV mechanisms is clear on the work that is currently being done in the WG. One example is architecture defined for the virtual Evolved Packet Core (vEPC) in [I-D.matsushima-stateless-uplane-vepc]. Here, the authors describe a particular realization of the vEPC concept, which is designed to support NFV. In the defined architecture, the user plane of EPC is decoupled from the control-plane and uses routing information to forward packets of mobile nodes. This proposal does not modify the signaling of the EPC control plane, although the EPC control plane runs on an hypervisor.

Potential gap: in a vEPC/DMM context, how to run the EPC control plane on NFV.

The DMM WG is also looking at ways to supporting the separation of the Control-Plane for mobility- and session management from the actual Data-Plane [I-D.ietf-dmm-fpc-cpdp]. The protocol semantics being defined abstract from the actual details for the configuration of Data-Plane nodes and apply between a Client function, which is used by an application of the mobility Control-Plane, and an Agent function, which is associated with the configuration of Data-Plane nodes according to the policies issued by the mobility Control-Plane.

Potential gap: the actual mappings between these generic protocol semantics and the configuration commands required on the data plane network elements are not in the scope of this document, and are

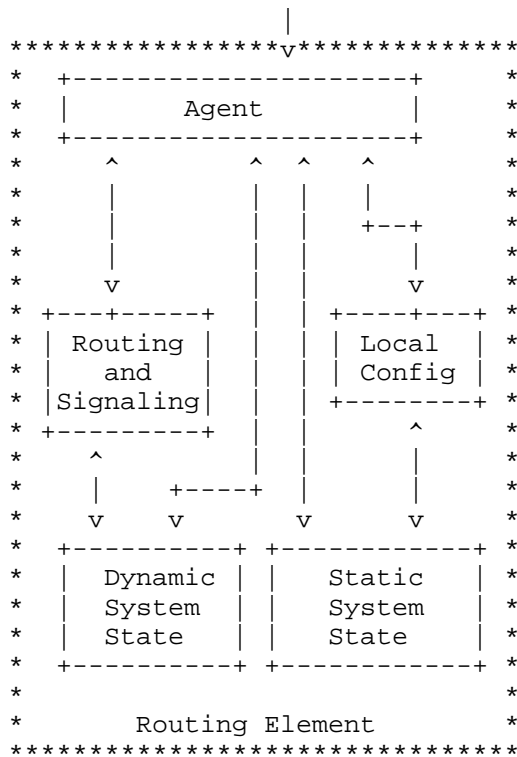


Figure 6: Architecture of a routing element

The I2RS architecture proposes to use model-driven APIs. Services can correspond to different data-models and agents can indicate which model they support.

Potential gap: network virtualization is not the main aim of the I2RS WG. However, they provide an infrastructure that can be part of an SDN deployment.

4.6. BESS WG

BGP is already used as a protocol for provisioning and operating Layer-3 (routed) Virtual Private Networks (L3VPNs). The BGP Enabled Services (BESS) working group is responsible for defining, specifying, and extending network services based on BGP. In particular, the working group will work on the following services:

- o BGP-enabled VPN solutions for use in the data center networking. This work includes consideration of VPN scaling issues and mechanisms applicable to such environments.

- o Extensions to BGP-enabled VPN solutions for the construction of virtual topologies in support of services such as Service Function Chaining.

Potential gap: The most relevant activity in BESS that would be worthwhile to investigate for relevance to network virtualization (NFV) is the extensions to BGP-enabled VPN solutions to support of Service Function Chaining [I-D.rfernando-bess-service-chaining].

4.7. BM WG

The Benchmarking Methodology Working Group (BMWG) provides recommendations concerning the key performance characteristics of internetworking technologies, or benchmarks for network devices, systems, and services. The scope of BMWG includes benchmarks for the management, control, and forwarding planes, and is.

The main distinguishing characteristic of BMWG from other IETF measurement initiatives like the IPPM WG is that BMWG is limited to characterization of implementations using controlled stimuli in a lab environment. The BMWG does not attempt to produce benchmarks for live, operational networks.

As part of the tasks of the BMWG, it is explicitly tasked to develop benchmarks and methodologies for VNF and related infrastructure benchmarking. Benchmarking Methodologies have reliably characterized many physical devices. This work item extends and enhances the methods to virtual network functions (VNF) and their unique supporting infrastructure. The first deliverable from this activity mentioned in the charter of the WG is a document [I-D.ietf-bmwg-virtual-net] that considers the new benchmarking space to ensure that common issues are recognized from the start, using background materials from industry and SDOs (e.g., IETF, ETSI NFV). This document investigates the additional methodological considerations necessary when benchmarking VNFs instantiated and hosted in general-purpose hardware. The approach is to benchmark physical and virtual network functions in the same way when possible, thereby allowing direct comparison. Also defining benchmarking combinations of physical and virtual devices in a System Under Test.

Benchmarks for platform capacity and performance characteristics of virtual routers, switches, and related components will be also addressed, including comparisons between physical and virtual network functions. In many cases, the traditional benchmarks should be applicable to VNFs, but the lab set-ups, configurations, and measurement methods will likely need to be revised or enhanced.

There are additional documents of the BMWG relevant to the virtualization area, such as:
[I-D.ietf-bmwg-sdn-controller-benchmark-term],
[I-D.ietf-bmwg-sdn-controller-benchmark-meth], [I-D.kim-bmwg-ha-nfvi]
and [I-D.vspcrf-bmwg-vswitch-opnfv].

4.8. TEAS WG

Transport network infrastructure provides end-to-end connectivity for networked applications and services. Network virtualization facilitates effective sharing (or 'slicing') of physical infrastructure by representing resources and topologies via abstractions, even in a multi-administration, multi-vendor, multi-technology environment. In this way, it becomes possible to operate, control and manage multiple physical networks elements as single virtualized network. The users of such virtualized network can control the allocated resources in an optimal and flexible way, better adapting to the specific circumstances of higher layer applications.

Abstraction and Control of Transport Networks (ACTN) intends to define methods and capabilities for the deployment and operation of transport network resources [I-D.ceccarelli-teas-actn-framework]. This activity is currently being carried out within the Traffic Engineering Architecture and Signaling (TEAS) WG.

Several use cases are being proposed for both fixed and mobile scenarios [I-D.leeking-teas-actn-problem-statement].

Potential gap: Several use cases in ACTN are relevant to network virtualization (NFV) in mobile environments. Control of multi-tenant mobile backhaul transport networks, mobile virtual network operation, etc, can be influenced by the location of the network functions. A control architecture allowing for inter-operation of NFV and transport network (e.g., for combined optimization) is one relevant area for research.

4.9. I2NSF WG

The I2NSF WG at defining interfaces to the flow based network security functions (NSFs) hosted by service providers at different premises. Network Security Function (NSF) is to ensure integrity, confidentiality and availability of network communications, to detect unwanted activity, and to block it or at least mitigate its effects. NSFs are provided and consumed in increasingly diverse environments. Users of NSFs could consume network security services hosted by one or more providers, which may be their own enterprise, service

providers, or a combination of both. The NSF's may be provided by physical and/or virtualized infrastructure.

Without standard interfaces to express, monitor, and control security policies that govern the behavior of NSF's, it becomes virtually impossible for security service providers to automate their service offerings that utilize different security functions from multiple vendors. Based on this, the main goal of I2NSF is to define an information model, a set of software interfaces and data models for controlling and monitoring aspects of NSF's (both physical and virtual) [I-D.jeong-i2nsf-sdn-security-services].

Since different security vendors may support different features and functions on their devices, I2NSF focuses on flow based NSF's that provide treatment to packets/flow.

The I2NSF WG's target deliverables include: (i) a use cases, problem statement, gap analysis document, (ii) a framework document, presenting an overview of the use of NSF's and the purpose of the models developed by the WG, (iii) a single, unified, Information Model for controlling and monitoring flow-based NSF's, (iv) the corresponding YANG Data Models derived from the Information Model, (v) a vendor-neutral vocabulary to enable the characteristics and behavior of NSF's to be specified without requiring the NSF's themselves to be standardized, and (vi) an examination of existing secure communication mechanisms to identify the appropriate ones for carrying the controlling and monitoring information between the NSF's and their management entities. The WG is also targeted to work closely with I2RS, Netconf and Netmod WGs, as well as to communicate with external SDOs like ETSI NFV.

Potential gap: aspects of NSF's such as device or network provisioning and configuration are out of scope.

Potential gap: the use of SDN tools to interact with security functions is not explicitly considered, but seems a potential approach, as for example described for the particular case of IPsec flow protection in [I-D.abad-sdnrg-sdn-ipsec-flow-protection].

4.10. IPPM WG

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. The IPPM WG is a long running WG that started in 1997. The architecture (framework) for IPPM WG metrics and associated protocols are defined in RFC 2330 [RFC2330]. Some

examples of recent output by IPPM WG include "A Reference Path and Measurement Points for Large-Scale Measurement of Broadband Performance" (RFC 7398 [RFC7398]) and "Framework for TCP Throughput Testing" (RFC 6349 [RFC6349]).

The IPPM WG currently does not have a charter item or active drafts related to the topic of network virtualization. On the automation and orchestration side, there is an ongoing effort [I-D.cmzrjp-ippm-twamp-yang] to define a YANG model for the IPPM protocol.

Potential gap: 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. A potential topic for the IPPM WG is defining packet delay, throughput, and test framework for the application traffic flowing through the NFVI.

4.11. NFV RG

The NFVRG focuses on research problems associated with virtualization of fixed and mobile network infrastructures, new network architectures based on virtualized network functions, virtualization of the home and enterprise network environments, co-existence with non-virtualized infrastructure and services, and application to growing areas of concern such as Internet of Things (IoT) and next generation content distribution. Another goal of the NFVRG is to bring a research community together that can jointly address such problems, concentrating on problems that relate not just to networking but also to computing and storage constraints in such environments.

Since the NFVRG is a research group, it has a wide scope. In order to keep the focus, the group has identified some near term work items: (i) Policy based Resource Management, (ii) Analytics for Visibility and Orchestration, (iii) Virtual Network Function (VNF) Performance Modelling to facilitate transition to NFV and (iv) Security and Service Verification.

4.12. VNFpool BoF

The VNFPOOL BoF proposed to work on the way to group Virtual Network Function (VNF) into pools to improve resilience, provide better scale-out and scale-in characteristics, implement stateful failover among VNF members of a pool, etc. Additionally, they propose to create VNF sets from VNF pools. For this, the BoF proposed to study

signaling (both between members of a pool and across pools), state sharing mechanisms between members of a VNFPOOL, the exchange of reliability information between VNF sets, their users and the underlying network, and the reliability and security of the control plane needed to transport the exchanged information.

The use cases initially considered by VNFPOOL include Content Deliver Networks (CDNs), the LTE mobile core network and reliable server pooling. The VNFPOOL work has been dropped in the IETF.

Potential gap: VNFPOOL tried to introduce and manage resilience in virtualized networking environments and therefore addresses a desirable feature for any software defined network. VNFPOOL has also been integrated into the NFV architecture [I-D.bernini-nfvrg-vnf-orchestration].

5. Summary of Gaps

Potential Gap-1: as stated in the SFC charter, any work on the management and configuration of SFC components related to the support of Service Function Chaining will not be done yet, until better understood and scoped. This part is of special interest for operators and would be required in order to actually put SFC mechanisms into operation.

Potential Gap-2: redundancy and reliability mechanisms are currently not dealt with by SFC or any other WG in the IETF. While this has been the main goal of the VNFpool BoF efforts, since VNFPOOL work has been dropped for the time being without any WG being chartered, the technical topics it aimed at targetting still remain un-addressed.

Potential Gap-3: it would be worthwhile to see if some of the specific approaches developed in the NVO3 WG (e.g. overlays, traffic isolation, VM migration) can be applied outside the DC, and specifically if they can be applicable to network virtualization (NFV). These approaches would be most relevant to the ETSI Network Function Virtualization Infrastructure (NFVI), and the Virtualized Infrastructure Manager part of the MANO.

Potential Gap-4: the most relevant activity in BESS that would be worthwhile to investigate for relevance to network virtualization (NFV) is the extensions to BGP-enabled VPN solutions to support of Service Function Chaining.

Potential Gap-5: in a vEPC/DMM context, how to run the EPC control plane on NFV.

Potential Gap-6: in DMM, on the work item addressing the separation of the Control-Plane for mobility- and session management from the actual Data-Plane, the actual mappings between these generic protocol semantics and the configuration commands required on the data plane network elements (e.g., OpenFlow switches) are not currently in the scope of the DMM WG.

Potential Gap-7: network virtualization is not the main aim of the I2RS WG. However, they provide an infrastructure that can be part of an SDN deployment.

Potential Gap-8: VNFPOOL tries to introduce and manage resilience in virtualized networking environments and therefore addresses a desirable feature for any software defined network. VNFPOOL has also been integrated into the NFV architecture [I-D.bernini-nfvrg-vnf-orchestration].

Potential Gap-9: within the Traffic Engineering Architecture and Signaling (TEAS) WG, several use cases in ACTN are relevant to network virtualization (NFV) in mobile environments. Control of multi-tenant mobile backhaul transport networks, mobile virtual network operation, etc, can be influenced by the location of the network functions. A control architecture allowing for inter-operation of NFV and transport network (e.g., for combined optimization) is one relevant area for research.

Potential Gap-10: within I2NSF', aspects of NSFs such as device or network provisioning and configuration are out of scope.

Potential Gap-11: the use of SDN tools to interact with security functions is not explicitly considered in I2NSF, but seems a potential approach, as for example described for the particular case of IPsec flow protection in [I-D.abad-sdnrg-sdn-ipsec-flow-protection].

Potential Gap-12: 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. A potential topic for the IPPM WG is defining packet delay, throughput, and test framework for the application traffic flowing through the NFVI.

6. IANA Considerations

N/A.

7. Security Considerations

TBD.

8. Acknowledgments

The authors want to thank Dirk von Hugo, Rafa Marin, Diego Lopez, Ramki Krishnan, Kostas Pentikousis, Rana Pratap Sircar and Alfred Morton for their very useful reviews and comments to the document.

The work of Pedro Aranda is supported by the European FP7 Project Trilogy2 under grant agreement 317756.

9. Informative References

[etsi_nvf_whitepaper]

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

[I-D.abad-sdnrg-sdn-ipsec-flow-protection]

Abad-Carrascosa, A., Lopez, R., and G. Lopez-Millan, "Software-Defined Networking (SDN)-based IPsec Flow Protection", draft-abad-sdnrg-sdn-ipsec-flow-protection-01 (work in progress), October 2015.

[I-D.bernini-nfvrg-vnf-orchestration]

Bernini, G., Maffione, V., Lopez, D., and P. Aranda, "VNF Pool Orchestration For Automated Resiliency in Service Chains", draft-bernini-nfvrg-vnf-orchestration-01 (work in progress), October 2015.

[I-D.ceccarelli-teas-actn-framework]

Ceccarelli, D. and Y. Lee, "Framework for Abstraction and Control of Traffic Engineered Networks", draft-ceccarelli-teas-actn-framework-01 (work in progress), March 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.

[I-D.ietf-bmwg-sdn-controller-benchmark-meth]

Vengainathan, B., Basil, A., Tassinari, M., Manral, V., and S. Banks, "Benchmarking Methodology for SDN Controller Performance", draft-ietf-bmwg-sdn-controller-benchmark-meth-01 (work in progress), March 2016.

- [I-D.ietf-bmwg-sdn-controller-benchmark-term]
Vengainathan, B., Basil, A., Tassinari, M., Manral, V.,
and S. Banks, "Terminology for Benchmarking SDN Controller
Performance", draft-ietf-bmwg-sdn-controller-benchmark-
term-01 (work in progress), March 2016.
- [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-dmm-fpc-cpdp]
Liebsch, M., Matsushima, S., Gundavelli, S., and D. Moses,
"Protocol for Forwarding Policy Configuration (FPC) in
DMM", draft-ietf-dmm-fpc-cpdp-01 (work in progress), July
2015.
- [I-D.ietf-i2rs-architecture]
Atlas, A., Halpern, J., Hares, S., Ward, D., and T.
Nadeau, "An Architecture for the Interface to the Routing
System", draft-ietf-i2rs-architecture-13 (work in
progress), February 2016.
- [I-D.ietf-nvo3-arch]
Black, D., Hudson, J., Kreeger, L., Lasserre, M., and T.
Narten, "An Architecture for Overlay Networks (NVO3)",
draft-ietf-nvo3-arch-04 (work in progress), October 2015.
- [I-D.jeong-i2nsf-sdn-security-services]
Jeong, J., Kim, H., Jung-Soo, P., Ahn, T., and s.
sehuilee@kt.com, "Software-Defined Networking Based
Security Services using Interface to Network Security
Functions", draft-jeong-i2nsf-sdn-security-services-04
(work in progress), March 2016.
- [I-D.kim-bmwg-ha-nfvi]
Kim, T. and E. Paik, "Considerations for Benchmarking High
Availability of NFV Infrastructure", draft-kim-bmwg-ha-
nfvi-00 (work in progress), October 2015.
- [I-D.leeking-teas-actn-problem-statement]
Lee, Y., King, D., Boucadair, M., Jing, R., and L.
Contreras, "Problem Statement for Abstraction and Control
of Transport Networks", draft-leeking-teas-actn-problem-
statement-00 (work in progress), June 2015.

- [I-D.matsushima-stateless-uplane-vepc]
Matsushima, S. and R. Wakikawa, "Stateless user-plane architecture for virtualized EPC (vEPC)", draft-matsushima-stateless-uplane-vepc-05 (work in progress), September 2015.
- [I-D.rfernando-bess-service-chaining]
Fernando, R., Rao, D., Fang, L., Napierala, M., So, N., and A. Farrel, "Virtual Topologies for Service Chaining in BGP/IP MPLS VPNs", draft-rfernando-bess-service-chaining-01 (work in progress), April 2015.
- [I-D.vsperf-bmwg-vswitch-opnfv]
Tahhan, M., O'Mahony, B., and A. Morton, "Benchmarking Virtual Switches in OPNFV", draft-vsperf-bmwg-vswitch-opnfv-01 (work in progress), October 2015.
- [RFC2330] Paxson, V., Almes, G., Mahdavi, J., and M. Mathis, "Framework for IP Performance Metrics", RFC 2330, DOI 10.17487/RFC2330, May 1998, <<http://www.rfc-editor.org/info/rfc2330>>.
- [RFC6349] Constantine, B., Forget, G., Geib, R., and R. Schrage, "Framework for TCP Throughput Testing", RFC 6349, DOI 10.17487/RFC6349, August 2011, <<http://www.rfc-editor.org/info/rfc6349>>.
- [RFC7398] Bagnulo, M., Burbridge, T., Crawford, S., Eardley, P., and A. Morton, "A Reference Path and Measurement Points for Large-Scale Measurement of Broadband Performance", RFC 7398, DOI 10.17487/RFC7398, February 2015, <<http://www.rfc-editor.org/info/rfc7398>>.
- [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, <<http://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, <<http://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, <<http://www.rfc-editor.org/info/rfc7665>>.

Appendix A. The mobile network use case

A.1. The 3GPP Evolved Packet System

TBD. This will include a high level summary of the 3GPP EPS architecture, detailing both the EPC (core) and the RAN (access) parts. A link with the two related ETSI NFV use cases (Virtualisation of Mobile Core Network and IMS, and Virtualisation of Mobile base station) will be included.

The EPS architecture and some of its standardized interfaces are depicted in Figure 7. The EPS provides IP connectivity to user equipment (UE) (i.e., mobile nodes) and access to operator services, such as global Internet access and voice communications. The EPS comprises the core network -- called Evolved Packet Core (EPC) -- and different radio access networks: the 3GPP Access Network (AN), the Untrusted non-3GPP AN and the Trusted non-3GPP AN. There are different types of 3GPP ANs, with the evolved UMTS Terrestrial Radio Access Network (E-UTRAN) as the most advanced one. QoS is supported through an EPS bearer concept, providing bindings to resource reservation within the network.

The evolved NodeB (eNB), the Long Term Evolution (LTE) base station, is part of the access network that provides radio resource management, header compression, security and connectivity to the core network through the S1 interface. In an LTE network, the control plane signaling traffic and the data traffic are handled separately. The eNBs transmit the control traffic and data traffic separately via two logically separate interfaces.

The Home Subscriber Server, HSS, is a database that contains user subscriptions and QoS profiles. The Mobility Management Entity, MME, is responsible for mobility management, user authentication, bearer establishment and modification and maintenance of the UE context.

The Serving gateway, S-GW, is the mobility anchor and manages the user plane data tunnels during the inter-eNB handovers. It tunnels all user data packets and buffers downlink IP packets destined for UEs that happen to be in idle mode.

The Packet Data Network (PDN) Gateway, P-GW, is responsible for IP address allocation to the UE and is a tunnel endpoint for user and control plane protocols. It is also responsible for charging, packet filtering, and policy-based control of flows. It interconnects the mobile network to external IP networks, e.g. the Internet.

In this architecture, data packets are not sent directly on an IP network between the eNB and the gateways. Instead, every packet is

tunneled over a tunneling protocol - the GPRS Tunneling Protocol (GTP over UDP/IP). A GTP path is identified in each node with the IP address and a UDP port number on the eNB/gateways. The GTP protocol carries both the data traffic (GTP-U tunnels) and the control traffic (GTP-C tunnels). Alternatively Proxy Mobile IP (PMIPv6) is used on the S5 interface between S-GW and P-GW.

In addition to the above basic functions and entities, there are also additional features being discussed by the 3GPP that are relevant from a network virtualization viewpoint. One example is the Traffic Detection Function (TDF), which can be used by the P-GW, and in general by the whole transport network, to decide how to forward the traffic. In a virtualized infrastructure, this kind of information can be used to elastic and dynamically adapt the network capabilities to the traffic nature and volume.

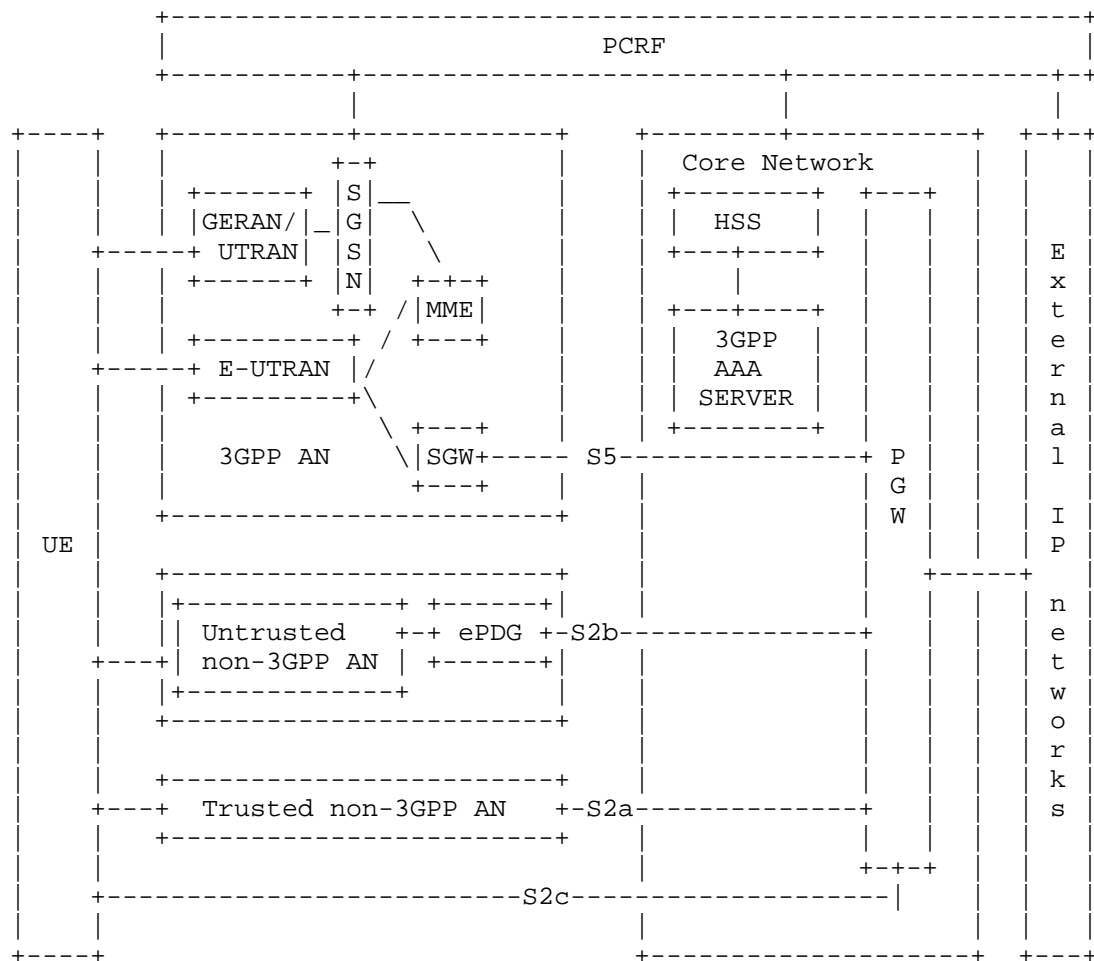


Figure 7: EPS (non-roaming) architecture overview

A.2. Virtualizing the 3GPP EPS

TBD. We describe how a "virtual EPS" (vEPS) would look like and the existing gaps that exist from the point of view of network virtualization.

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

Email: JuanCarlos.Zuniga@InterDigital.com
URI: <http://www.InterDigital.com/>

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

Email: luismiguel.conterasurillo@telefonica.com

Pedro Aranda
Telefonica I+D
Ronda de la Comunicacion, S/N
Madrid 28050
Spain

Email: pedroa.aranda@telefonica.com

NFV Research Group
Internet-Draft
Intended status: Informational
Expires: October 12, 2017

G. Bernini
G. Landi
Nextworks
D. Lopez
Telefonica
P. Aranda Gutierrez
UC3M
April 10, 2017

VNF Pool Orchestration For Automated Resiliency in Service Chains
draft-bernini-nfvrg-vnf-orchestration-04

Abstract

Network Function Virtualisation (NFV) aims at evolving the way network operators design, deploy and provision their networks by leveraging on standard IT virtualisation technologies to move and consolidate a wide range of network functions and services onto industry standard high volume servers, switches and storage. The primary target for operators, stimulated by the recent updates on NFV and SDN, is the network edge. In fact, operators are considering their future datacentres and Points of Presence (PoPs) as increasingly dynamic infrastructures where Virtualised Network Functions (VNFs) and on-demand chained services with high elasticity will be deployed.

This document presents an orchestration framework for automated deployment of highly available VNF chains. Resiliency of VNFs and chained services is a key requirement for operators to improve, ease, automate and speed up services lifecycle management. The proposed VNFs orchestration framework is also positioned with respect to current NFV and Service Function Chaining (SFC) architectures and solutions.

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 October 12, 2017.

Copyright Notice

Copyright (c) 2017 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. Terminology	4
3. VNF Pool Orchestration for Resilient Virtual Appliances . . .	4
3.1. Problem Statement	5
3.2. Orchestration Framework	6
3.2.1. Orchestrator	8
3.2.2. SDN Controller	8
3.2.3. Service Function Path Manager	9
3.2.4. Edge Configurator	10
3.3. Resiliency Control Functions for Chained VNFs	10
4. Positioning in Existing NFV and SFC Frameworks	12
4.1. Mapping into NFV Architecture	12
4.2. Mapping into SFC Architecture	13
5. IANA Considerations	13
6. Security Considerations	13
7. Acknowledgements	13
8. References	14
8.1. Normative References	14
8.2. Informative References	14
Authors' Addresses	14

1. Introduction

Current Telco infrastructures are facing the rapid development of the cloud market, which includes a broad range of emerging virtualised services and distributed applications. Network Function

Virtualisation (NFV) is gaining wide interest across operators as a means to evolve the way networks are operated and provisioned, with network functions and services traditionally integrated in hardware devices executed in virtualised environments.

A Virtualised Network Function (VNF) provides the same function as its non virtualised equivalent (e.g. firewall, load balancer) but is deployed as a software instance running on general purpose servers using virtualisation technologies. The main idea, therefore, is to run network functions in datacentres or commodity network nodes that are, in some cases, close to the end user premises. With NFV, network functions are moved from specialised hardware devices to self-contained virtual machines running in general purpose servers. These virtualised functions can be deployed in multiple instances or moved to various locations in the network, adapting themselves to traffic dynamicity and customer demands without the overhead cost and management of installing new equipment.

Operator networks are populated with a large and increasing variety of proprietary software and hardware tools and appliances. The deployment of new network services in operational environments is often a complex and costly procedure, where additional physical space and power are required to accommodate new boxes. Additionally, current hardware-based appliances rapidly reach end of life. This requires that much of the design integration and deployment cycle be repeated with little revenue benefit. In this context, the transition of network functions and appliances from hardware to software solutions by means of NFV promises to address and overcome these hindrances for network operators.

The considerations above are valid for stand-alone VNFs running independently. However, additional challenges and requirements raise for network operators when services offered to customers are built by the composition of multiple VNFs. In this case, the deployment and provisioning of each (virtual) service component for the customer needs to be coordinated with the other VNFs, applying control functions to steer the traffic through them following a predefined order (i.e. according to the specific service function path). An orchestration framework capable of coordinating the automated deployment, configuration, provisioning and chaining of multiple VNFs would ease the management of the whole lifecycle of services offered to customers. Additionally, when dealing with virtualised functions, resiliency and high availability of chained services pose additional requirements for a VNF orchestration framework, in terms of detection of software failures at various levels (including hypervisors and virtual machines, hardware failure), and dynamic and intelligent reaction (virtual appliance migration, deployment of new VNFs, re-adapt the VNF chain).

This document presents an orchestration framework for automated deployment of high available VNF chains, and introduces its architecture and building blocks. Resiliency for both stand-alone VNFs and chained services is considered in this document as a key control function based on VNF pool concepts. The proposed VNF pool orchestration framework is also positioned with respect to approaches and architectures currently defined for Network Function Virtualisation and Service Function Chaining (SFC).

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].

The following acronyms are used in this document:

NFV: Network Function Virtualisation.

SDN: Software Defined Networking.

VNF: Virtualised Network Function.

SFC: Service Function Chaining.

CPE: Customer Premise Equipment.

VPN: Virtual Private Network.

EMS: Element Management System.

PoP: Point of Presence.

VM: Virtual Machine.

3. VNF Pool Orchestration for Resilient Virtual Appliances

The telco market is rapidly moving towards an "Everything as a Service" model, where the virtualisation of traditionally in-the-box network functions can benefit from Software Defined Networking (SDN) tools and technologies. As said, the recent updates and proposed solutions on NFV and SDN is practically bringing, from an operator perspective, a deep evolution on how the network edge is architected, operated and provisioned, since that is the place where VNFs and virtual services can be deployed and provisioned close to the customers. Operators target to evolve their datacentres and PoPs into increasingly dynamic infrastructures where VNFs and chained services can be deployed with high availability and high elasticity

to scale up and down while optimizing performances and resources utilization.

This section introduces the VNF pool orchestration framework for the deployment, provisioning and chaining of resilient virtual appliances and services within operator data-centres proposed in this document.

3.1. Problem Statement

The orchestration framework proposed in this document aims at solving some of the challenges that operators face when, trying to apply the base NFV concepts, they replace hardware devices implementing well-known network functions with software-based virtual appliances. In particular, this VNF orchestration framework targets an automated, flexible and elastic provisioning of service chains within operators' datacentres.

When operators need to compose and chain multiple VNFs to provision a given service to the customer, they need to operate network and computing resources in a coordinated way, and above all to implement control mechanisms and procedures to steer the traffic through the different VNFs and the customer sites. As an example, the virtualisation of the Customer Premises Equipment (CPE) is emerging as one of the first applications of the Network Functions Virtualisation (NFV) architecture that is currently being commercialized by several software (and hardware) vendors. It has the potential to generate a significant impact on the operators businesses. The term virtual CPE (vCPE) refers to the execution in a virtual environment of the network functions that traditionally integrated in hardware gears at customer premises, like BGP speakers, firewall, NAT, etc.

Different scenarios and use cases exist for the vCPE. Currently, the typical scenario is the vCPE in the PoP, that actually provides softwarization and shift to the first PoP of the operator for those network functions normally deployed at customer premises (e.g. NAT, Firewall, etc.). The goal is to manage the whole LAN environment of the customer, while preserving QoE of its services, providing added value services to users not willing to get involved with technology issues, and reducing maintenance trouble tickets and the need for in-house problem solving. In addition, the vCPE can be used in the operator's datacenter to implement in software those chained network functions to provision automated VPN services for customers [VCPE-T2], in order to dynamically and automatically extend existing L3 VPNs (e.g. connecting remote customer sites) to incorporate new virtual assets (like virtual machines) into a private cloud.

As additional requirements for the proposed orchestration framework, the use of VNFs opens new challenges concerning the reliability of provided virtual services. When network functions are deployed on monolithic hardware platforms, the lifecycle of individual services is strictly bound to the availability of the physical device, and management tools may detect outages and migrate affected services to new instances deployed on backup hardware. When introducing VNFs, individual network functions may still fail, but with more risk factors such as software failure at various levels, including hypervisors and virtual machines, hardware failure, and virtual appliance migration. Moreover, when considering chains of VNFs, the management and control tools used by the operators have to consider and apply reliability mechanisms at the service level, including transparent migration to backup VNFs and synchronization of state information. In this context, VNF pooling mechanisms and concepts are valid and applicable, thus considering VNF instances grouped as pools to provide the same function in a reliable way.

3.2. Orchestration Framework

The VNF pool orchestration framework proposed in this document aims to provide automated functions for the deployment, provisioning and composition of resilient VNFs within operators' datacentres. Figure Figure 1 presents the high level architecture, including building blocks and functional components.

This VNF pool orchestration framework is built around two key components: the orchestrator and the SDN controller. The orchestrator includes all the functions related to the management, coordination, and control of VNFs instantiation, configuration and composition. It is the component at the highest level of the architecture and represents the access point to the VNF pool orchestration framework for the operator. On the other hand, the SDN controller provides dynamic traffic steering and flexible network provisioning within the datacenter as needed by the VNF chains. The basic controller functions are augmented by a set of enhanced network applications deployed on top, that might be themselves control and management VNFs for operator use (i.e. not related to customers and users functions).

Therefore, the architecture depicted in Figure Figure 1 is a practical demonstration of how SDN and NFV technologies and concepts can be integrated to provide substantial benefits to network operators in terms of robustness, ease of management, control and provisioning of their network infrastructures and services. SDN and NFV are clearly complementary solutions for enabling virtualisation of network infrastructures, services and functions while supporting dynamic and flexible network traffic engineering.

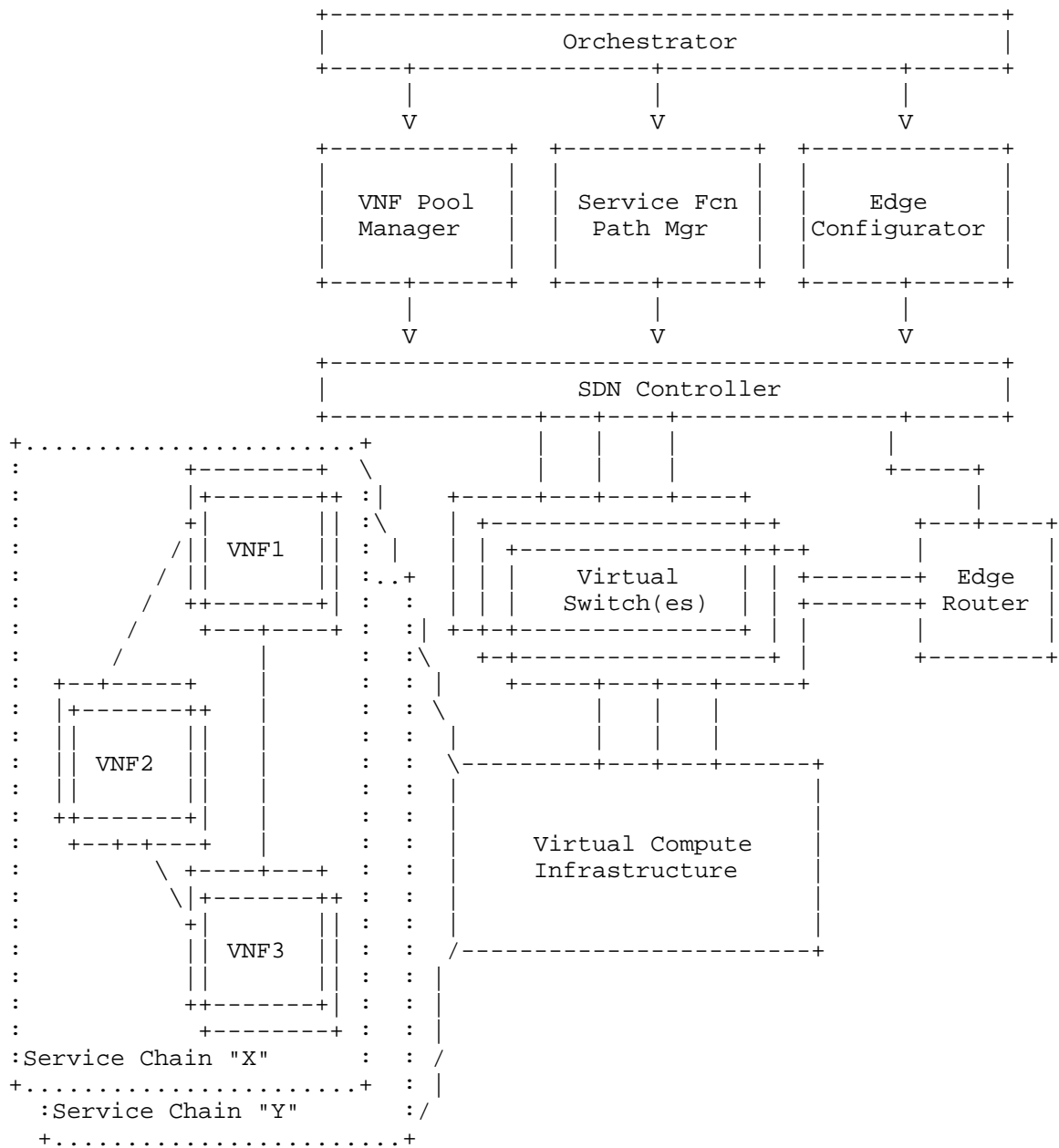


Figure 1: VNF Pool Orchestration Framework Architecture.

SDN focuses on network programmability, traffic steering and multi-tenancy by means of a common, open, and dynamic abstraction of network resources. NFV targets a progressive migration of network elements, network appliances and fixed function boxes into VMs that can be ran on commodity hardware, enabling the benefits of cloud and datacentres to be applied to network functions.

3.2.1. Orchestrator

The VNF orchestrator implements a set of functions to seamlessly control and manage in a coordinated way the instantiation and deployment of VNFs on one hand, and their composition and chaining to steer the traffic through them on the other. It is fully controlled and operated by the network operator, and basically it is the highest control and orchestration layer that sits above all the softwarized and virtualised components in the proposed architecture.

Therefore the VNF orchestrator provides a consistent way to access the system and provision chains of VNFs to the operator. It exposes a set of primitives to instantiate, configure VNFs and compose them according to the specific service chain requirements. Practically, it aims at enabling an efficient and dynamic management of operator's infrastructure resources with great flexibility by means of a consistent set of APIs.

To enable this, the VNF orchestrator can be seen as a composition of several internal functionalities, each providing a given coordination function needed to orchestrate the lower layer control and management functions depicted in Figure Figure 1 (i.e. VNF chain configuration, VNF pool provisioning, etc.). In practice, the VNF orchestrator needs to include at least an internal component to manage the instantiation and configuration of stand-alone VNFs (e.g. implemented by a self-contained VM) that might be directly interfaced with the physical servers in the datacenter. And also a dedicated component for programmatic coordination and provisioning of VNF chains is needed to properly orchestrate the traffic steering through VNFs belonging to the same service chain. This should also provide multi-tenant functionalities and maintain isolation across VNF chains deployed for different customers. It is then clear that the VNF orchestrator is the overall coordinator of the proposed framework, and it drives all the lower layer components that implement the actual control logic.

3.2.2. SDN Controller

The SDN controller provides the logic for network control, provisioning and monitoring. It is the component where the SDN abstraction happens. This means it exposes a set of primitives to

configure the datacenter network according to the requirements of the VNF chains to be provisioned, while hiding the specific technology constraints and capabilities of the software switches and edge routers underneath. The deployment of an SDN controller allows to implement a software driven VNF orchestration, with flexible and programmable network functions for service chaining and resilient virtual appliances.

At its southbound interface, the SDN controller interfaces with software switches running in servers, physical switches interconnecting them and edge routers connecting the datacenter with external networks. Multiple control protocols can be used at this southbound interface to actually provision the datacenter network and enable traffic steering through VNFs, including OpenFlow, OVSDB, NETCONF and others.

Therefore the SDN controller provides the basic network provisioning functions needed by upper layer coordination functions to perform service chain and VNF pool-wide actions. Indeed, the logic and the state at the service level is only maintained and coordinated by network applications on top of the SDN controller.

3.2.3. Service Function Path Manager

The service function path manager is deployed as a bridging component between the orchestrator and the SDN controller, and it is mostly dedicated to the implementation of VNF chaining and composition logic. It computes a suitable path to interconnect the involved VNFs (already instantiated and identified by the orchestrator) and forwards the network configuration request to the SDN controller for each new VNF chain requested by the orchestrator.

Following the datacenter service chains and related traffic types defined in [I-D.ietf-sfc-dc-use-cases], the service function path manager should implement its coordination logic to support both north-south and east-west chains. The former refer to network traffic staying within the datacenter but coming from a remote datacenter or a user through the edge router connecting to an external network. In this case, the service function path manager should also coordinate with the edge configurator to properly provision the datacenter edge router. Moreover, this north-south case may also refer to VNF chains spanning multiple datacentres, thus requiring a further inter-datacenter coordination between service function path managers and orchestrators. These coordination functions are out of the scope of this document. On the other hand, the east-west chains refer to VNFs treating network traffic that do not exit the datacenter. For both cases, the service function path manager (in combination with the SDN controller) should implement and

support proper service chains encapsulation solutions [I-D.ietf-sfc-nsh] to isolate and segregate traffic related to VNF chains belonging to different tenants.

Different deployment models may exist for the service function path manager: a dedicated configurator for each chain, or a single configurator for all the VNF chains. In the first approach, the orchestrator needs to implement some coordination logic related to the dynamic instantiation of configurators when new VNF chains are provisioned.

3.2.4. Edge Configurator

The edge configurator is a network control application deployed on top of the SDN controller. Its main role is to coordinate the provisioning and configuration of the edge router for those north-south VNF chains exiting the datacenter. In particular, it keeps the binding between the traffic steered through the VNFs and the related network service outside the datacenter terminated at the edge router (e.g. a L3 VPN, VLAN, VXLAN, VRF etc), possibly considering the service chain encapsulation implemented within the VNF chain. The mediation of the SDN controller allows to support a variety of control and management protocols for the actual configuration of the datacenter edge router.

3.3. Resiliency Control Functions for Chained VNFs

In the proposed orchestration architecture, the resiliency control functions that have been identified as a key feature for a flexible and dynamic provisioning of chained VNF services are implemented by the VNF pool manager depicted in Figure Figure 1. It is the entity that manages and coordinates VNFs reliability providing high availability and resiliency features at both stand-alone and chained VNFs level.

The deployment of VNF based services requires moving the resiliency capabilities and mechanisms from physical network devices (which are typically highly available and often specialized) to entities (like self-contained VMs) running VNFs in the context of pools of virtualised resources. When moving towards a resilient approach for VNF deployment and operation, in line with ETSI NFV Resiliency Requirements (NFV-REL001), the generic high availability requirements to be matched are translated into:

Service continuity: when a hardware failure or capacity limits (memory and CPU) occur on platforms hosting VMs (and therefore VNFs), it is necessary to migrate VNFs to other VMs and/or

hardware platforms to guarantee service continuity with minimum impact on the users

Topological transparency: the hand-over between live and backup VNFs must be implemented in a transparent way for the user and also for the service chain itself. The backup VNF instances need to replicate the necessary information (configuration, addressing, etc.) so that the network function is taken over without any topological disruption (i.e. at the VNF chain level)

Load balancing or scaling: migration of VNF instances may also happen for load-balancing purposes (e.g. for CPU, memory overload in virtualised platforms) or scaling of network services (with VNFs moved to new hardware platforms). In both cases the working network function is moved to a new VNF instance and the service continuity must be maintained.

Auto scale of VNFs instances: when a VNF requires increased resource allocation to improve overall service performance, the network function could be distributed across multiple VMs, and to guarantee the performance improvement dedicated pooling mechanisms for scaling up or down resources to each VNF in a consistent way are needed.

Multiple VNF resiliency classes: each type of end-to-end service (e.g. web, financial backend, video streaming, etc.) has its own specific resiliency requirements for the related VNFs. While for operators it is not easy to achieve service resiliency SLAs without building to peak, a basic set of VNF resiliency classes can be defined to identify some metrics, such as: if a VNF needs status synchronization; fault detection and restoration time objective (e.g. real-time); service availability metrics; service quality metrics; service latency metrics for VNF chain components.

The aim of the VNF pool orchestration presented in this document is to address the above requirements by introducing the VNF pool manager that follows the principles of the IETF VNFPOOL architecture [I-D.zong-vnfpool-arch], where a pool manager coordinates the reliability of stand-alone VNFs, by selecting the active instance and interacting with the Service Control Entity for consistent end-to-end service chain reliability and provisioning. In the VNF pool orchestration architecture illustrated in Figure Figure 1, the Service Control Entity is implemented by the combination of the orchestrator (for overall coordination of service chains) and the VNF chain configuration (for actual provisioning and coordination of individual service chains).

Different deployment models may exist for the VNF pool manager: a dedicated manager for each VNF chain, or a single one for all the chains.

In terms of offered resiliency functionalities, the VNF pool manager provides some post-configuration functions to instantiate VNFs (as self-contained VMs) with the desired degree of reliability and redundancy. This translates into further actions to create and configure additional VMs as backups, therefore building a pool for each VNF in the chain.

The VNF pool manager is conceived to offer several types and degrees of reliability functions. First, it provides specific functions for the persistence of VNFs configuration, including making periodic snapshots of the VMs running the VNF. Moreover, at runtime (i.e. with the service chain in place), it monitors the operational status and performances of the master VNFs VMs, and collects notifications about VMs status, e.g. by registering as an observer to dedicated services offered by the virtualisation platform used within the virtual compute infrastructure. Moreover, VNF pool manager reacts to any failure condition by autonomously replacing the master VNF with one of its backup on the pool, basically implementing a swap of VMs for service chain recovery purposes. Thus, the VNF pool manager also takes care in coordination with the service function path manager of implementing those resiliency mechanisms at the chain level. Two options have been identified so far: cold recovery and hot recovery. In the former, backup VNFs, properly configured with the same master configuration, are kept ready (but switched off) to be started when the master dies. In this case the recovery time depends on the specific VNF and its type of function, e.g. it may depend on convergence time for a virtual BGP router. In the hot recovery, active backup VNFs are kept synchronized with the master ones, and the recovery of the service chain (mostly performed at the service function path manager) in case of failure is faster than cold recovery.

4. Positioning in Existing NFV and SFC Frameworks

4.1. Mapping into NFV Architecture

For the presented solution to be integrated in the ETSI NFV reference architecture, some modifications need to be applied to it, with main focus on the Management and Orchestration (MANO) functions. The VNF pools replace the VNFs in the architecture. They are then controlled by the Element Management System (EMS) on the northbound. So the EMS has to be made VNFPOOL aware. Additional elements that need to support the mechanisms proposed by VNFPOOL are the VNF managers, which need to implement the resiliency and VNF scaling

(up-/downscale) functions. This has also implications on the NFV Orchestrator, which has to be aware of the augmented functionality offered by the VNF Manager. In fact, the NFV Orchestrator also provides primitives for VNFs chaining, matching the Service Control Entity in the VNFPool architecture. Therefore, even if ETSI NFV MANO does not include explicitly SDN, the Edge Configurator, and part of the Service Function Path Manager features might be also covered by an augmented NFV Orchestrator.

4.2. Mapping into SFC Architecture

[I-D.ietf-sfc-architecture] describes the Service Function Chaining (SFC) architecture. It describes the concept of a service function (SF) and how to chain SFs and provides only little detail of the SFC control plane, which is responsible with the coordination of the SFs and their stitching into SFCs. The combination of orchestrator, service function path manager and VNF pool manager functionalities described in this document cover most of the functions expected from the SFC control plane.

The interaction with the SFC Classifier is left for further study. We expect the VNFPOOL architecture to leverage on it to make sure that all VNFPOOL instances will be served traffic on during scale-up and that no traffic will be lost during scale-down.

5. IANA Considerations

This draft does not have any IANA consideration.

6. Security Considerations

Security issues related to VNF pool orchestration and resiliency of service chains are left for further study.

7. Acknowledgements

This work has been partially supported by the European Commission through the H2020 5G Crosshaul (The integrated fronthaul/backhaul, grant agreement no:H2020-671598) and Selfnet (Framework for Self-organized network management in virtualized and software defined networks, grant agreement no:H2020-671672) projects. 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.

Authors would also like to thank V. Maffione and G. Carrozzo from Nextworks for valuable discussions and contributions to the topics addressed in this document.

8. References

8.1. 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>>.

8.2. Informative References

- [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-06 (work in progress), February 2017.
- [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-nsh]
Quinn, P. and U. Elzur, "Network Service Header", draft-ietf-sfc-nsh-12 (work in progress), February 2017.
- [I-D.zong-vnfpool-problem-statement]
Zong, N., Dunbar, L., Shore, M., Lopez, D., and G. Karagiannis, "Virtualized Network Function (VNF) Pool Problem Statement", draft-zong-vnfpool-problem-statement-06 (work in progress), July 2014.
- [VCPE-T2] G. Bernini, G. Carrozzo, P. A. Gutierrez, D. R. Lopez, , "Virtualising the Network Edge: Virtual CPE for the datacenter and the PoP", European Conference on Networks and Communications , June 2014.

Authors' Addresses

Giacomo Bernini
Nextworks
Via Livornese 1027
San Piero a Grado, Pisa 56122
Italy

Phone: +39 050 3871600
Email: g.bernini@nextworks.it

Giada Landi
Nextworks
Via Livornese 1027
San Piero a Grado, Pisa 56122
Italy

Phone: +39 050 3871600
Email: g.landi@nextworks.it

Diego R. Lopez
Telefonica
C. Zurbaran, 12
Madrid 28010
Spain

Email: diego.r.lopez@telefonica.com

Pedro A. Aranda Gutierrez
Universidad Carlos III Madrid
Leganes 28911
Spain

Email: paranda@it.uc3m.es

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:

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

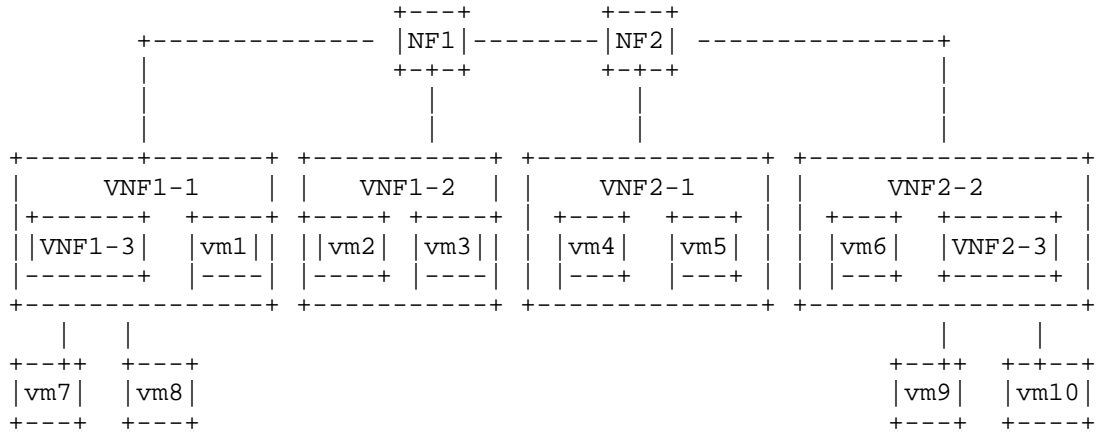


Figure 1: The sample VNFFG of network service

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 specification, understanding and maintenance. In addition, the neat formulation 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 commercial Datalog interpreter available 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:

$$h \leftarrow p_1, p_2, \dots, p_n$$

which can be defined as "p₁ and p₂ and ... and p_n implies h". "h" is the head of the rule, and "p₁, p₂, ..., p_n" is a list of literals that constitutes the body of the rule. Literals "p(x₁, ..., x_i, ..., x_n)" are either predicates applied to arguments "x_i" (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, ..., y_n)", in which "query" is a predicate contains arguments "m" and "y_i". "m" represents the monitoring function to be queried, e.g., end to end delay, average CPU usage, and etc. "y_i" 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 (vdul) and one Connection Point (CP) (cp11), it can be denoted as 'sub(vnf1, vdul)' 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 recursive 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>

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/lid-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.

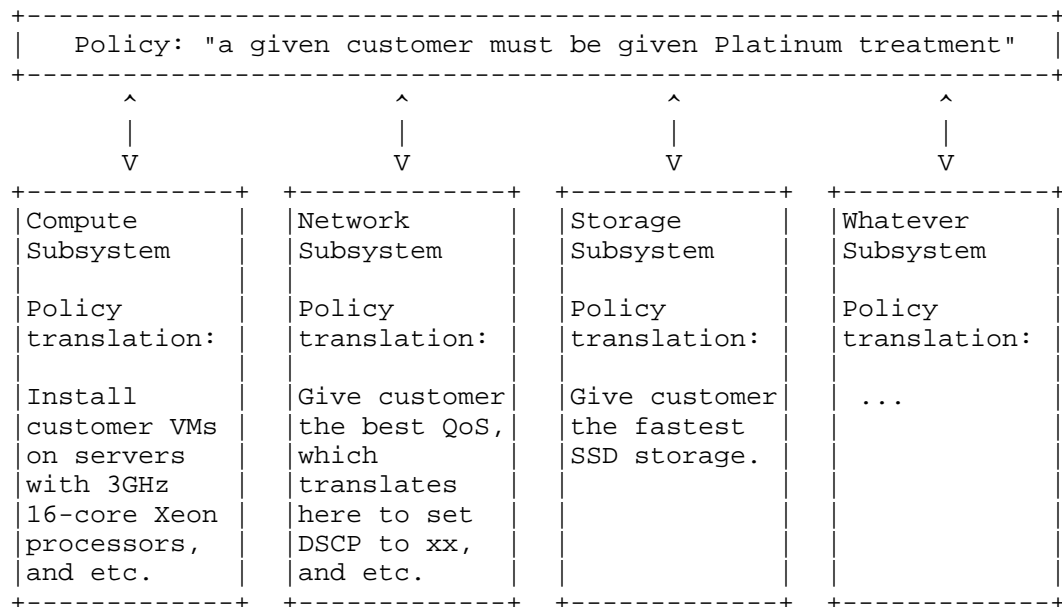


Figure 1: Example of Subsystem Translations of Policy Actions

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.

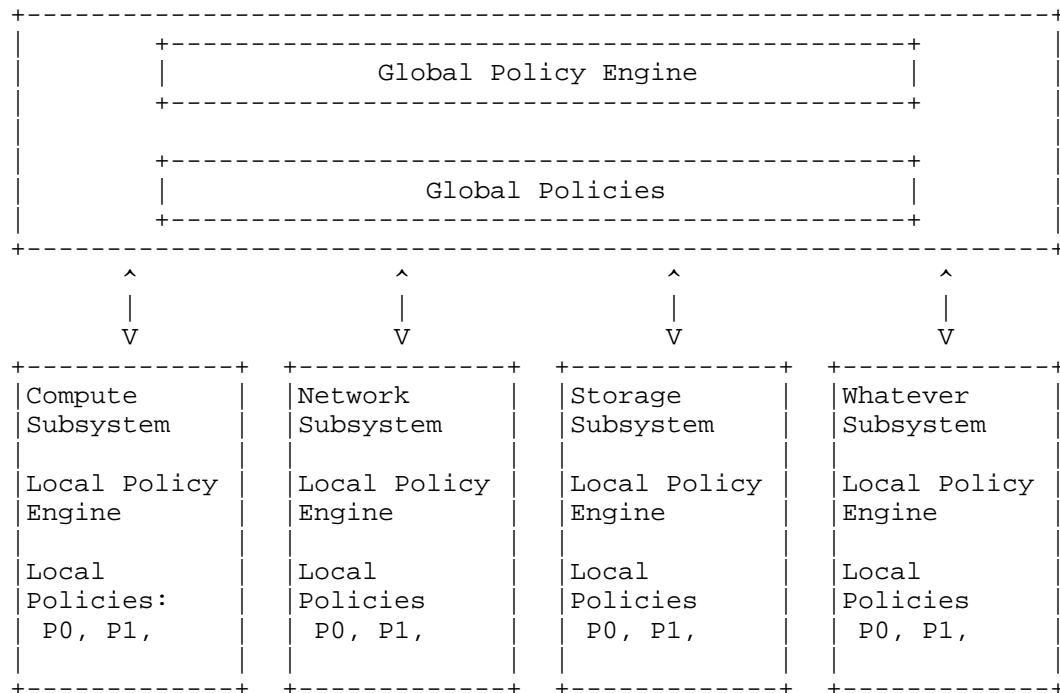


Figure 2: Global versus Local Policy Engines

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.

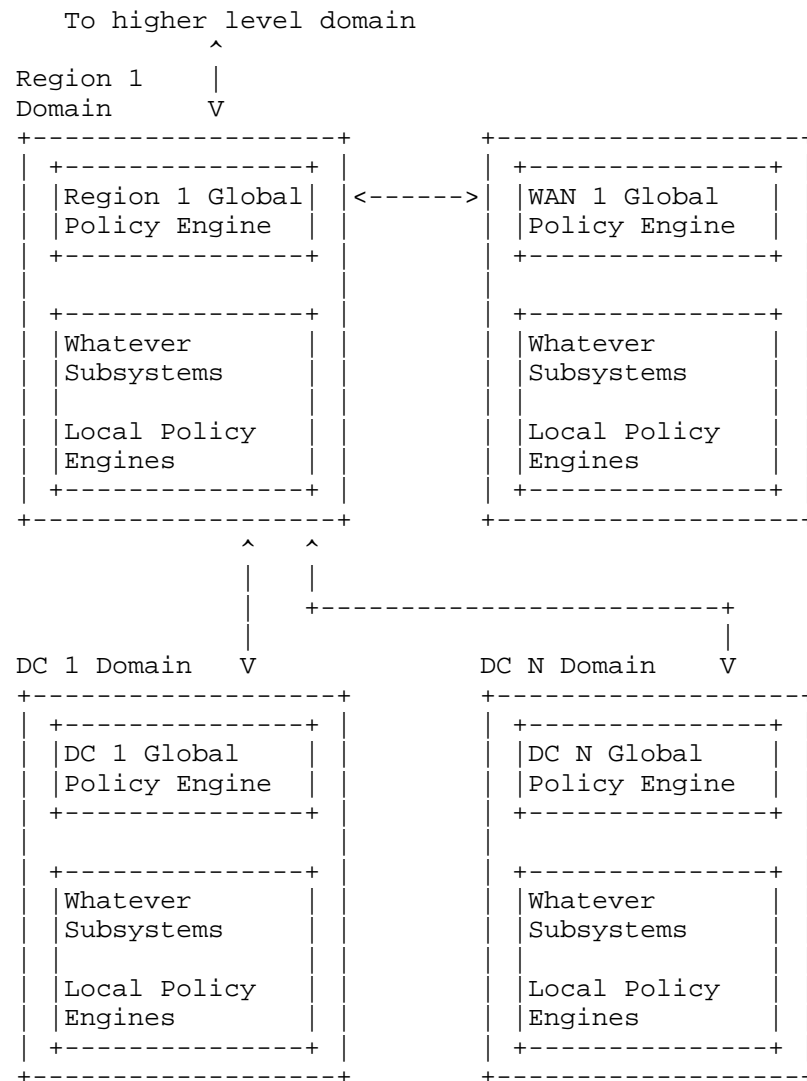


Figure 3: A Hierarchical Policy Framework

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 policies 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.

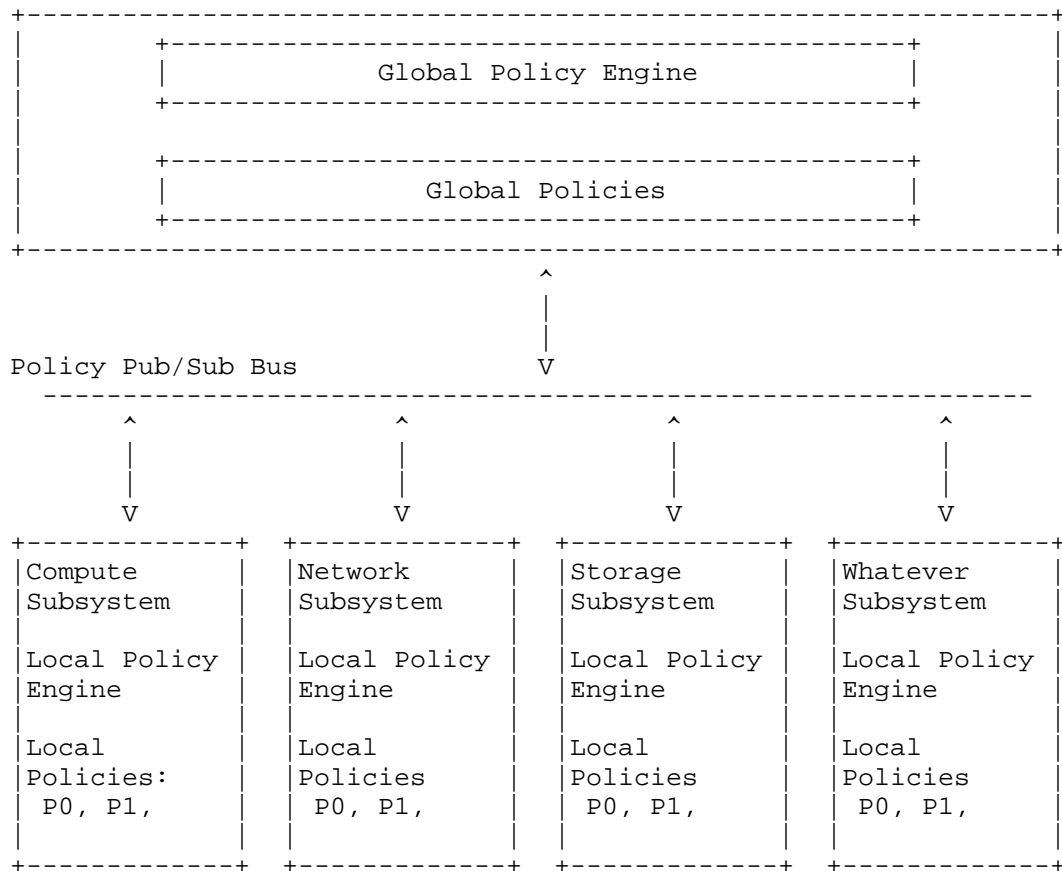


Figure 4: A Policy Pub/Sub Bus

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).

Pub/Sub bus to higher level

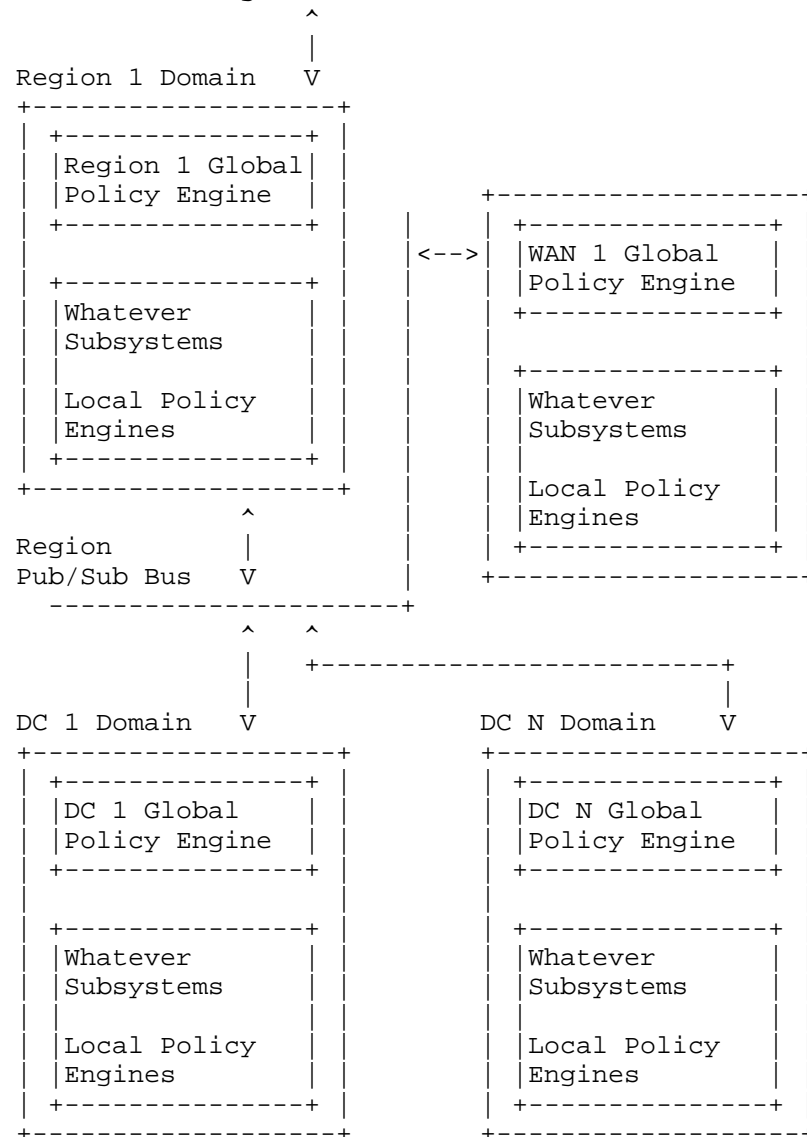


Figure 5: Pub/Sub Bus - Hierarchical Policy Framework

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.

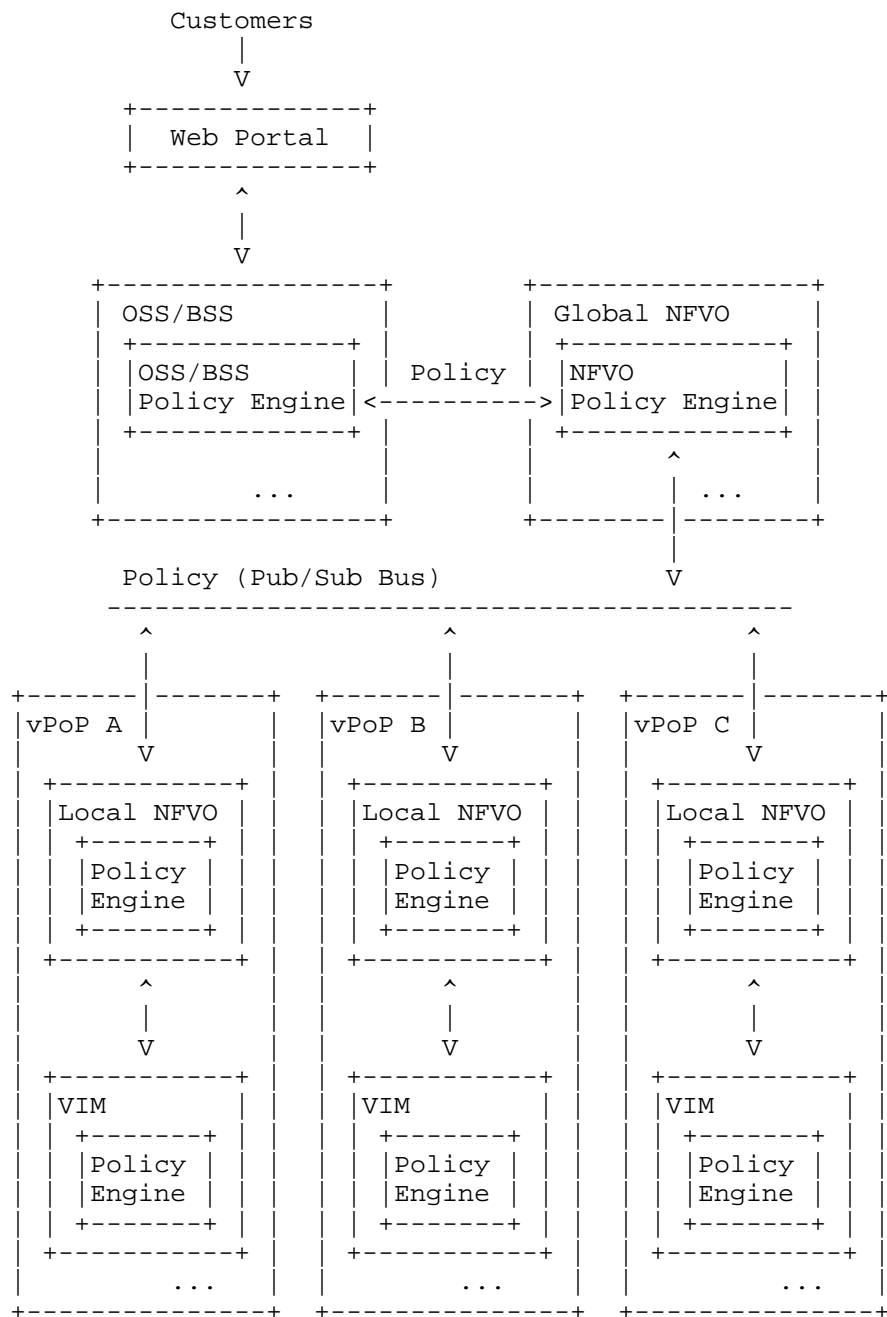


Figure 6: Simplified view of a service provider's NFV Architecture:
Multipoint Ethernet Service Example

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.

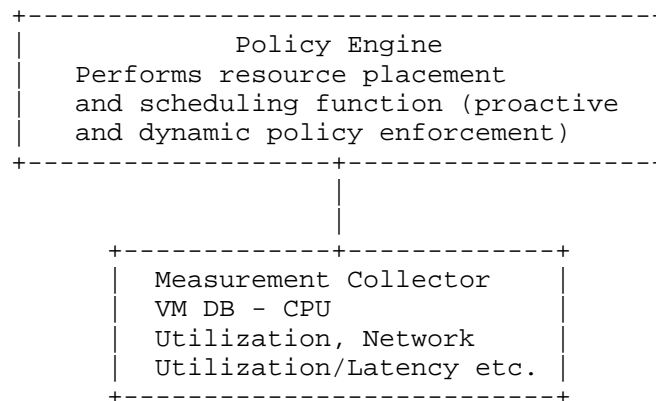


Figure 7: NFVIaaS Architecture for Policy Based Resource Placement and Scheduling

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 large.2.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-nfvias/>
- [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

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].

- o Network Functions (NF): A functional building block within a network infrastructure, which has well-defined external interfaces and a well-defined functional behavior.
- o Network service: A composition of network functions and defined by its functional and behavioural specification.
- o NFV Framework: The totality of all entities, reference points, information models and other constructs defined by the specifications published by the ETSI ISG NFV.
- o Virtualised Network Function (VNF): An implementation of an NF that can be deployed on a Network Function Virtualisation Infrastructure (NFVI).
- o 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.
- o NFVI-PoP: A location or point of presence that hosts NFV infrastructure
- o VNF Forwarding Graph (VNF-FG): A NF forwarding graph where at least one node is a VNF.
- o 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.
- o 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:

- o Resource availability,
- o 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
- o 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.

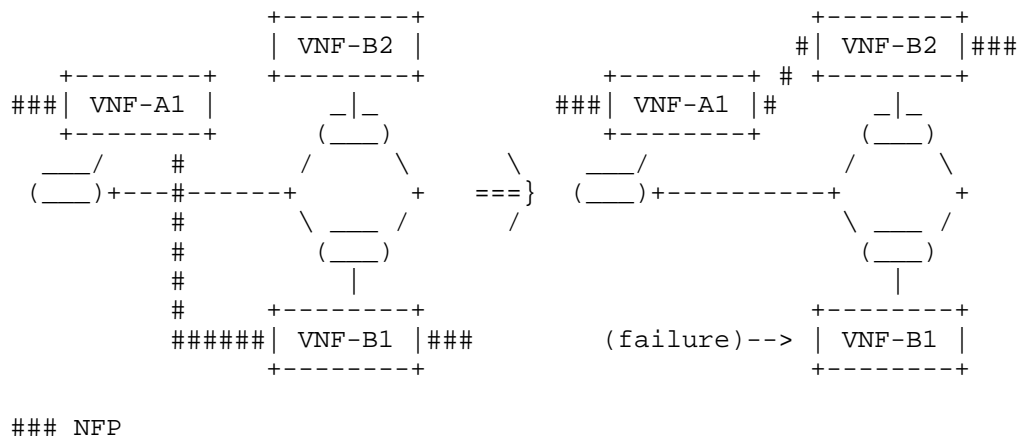


Figure 1: A fail-over use case

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:

- o 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:

- o 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.

- o 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.

- o 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.
- o 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:

- o Locate VNF instances
- o Evaluate the performance of VNF instances and VL instances
- o Relocate (or scale) VNF instances to update a NFP instance
- o 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:

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

- o latency of a VL instance
- o 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.

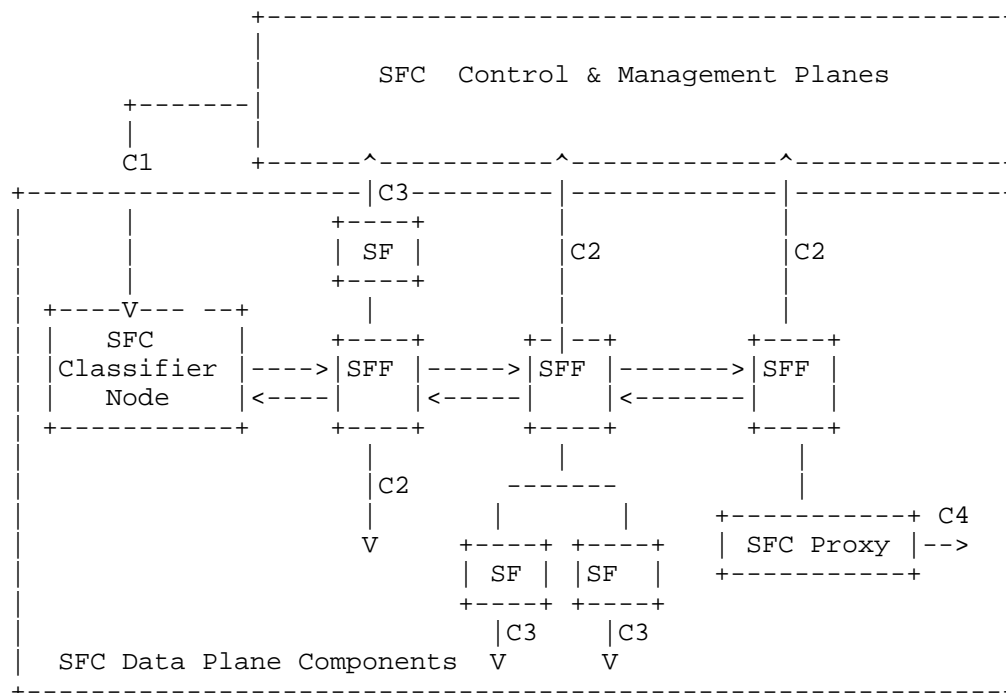


Figure 2: SFC control plane overview

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.

- o Top level application control at the SFC Control & Management Planes
- o An abstraction layer between the application layer and the probing layer. This would decouple NFVI and link monitoring methods from the application layer
- o 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 RG
Internet-Draft
Intended status: Informational
Expires: April 19, 2016

L. Mo
B. Khasnabish, Ed.
ZTE (TX) Inc.
October 17, 2015

NFV Reliability using COTS Hardware
draft-mlk-nfvrg-nfv-reliability-using-cots-01

Abstract

This draft discusses the results of a recent study on the feasibility of using Commercial Off-The-Shelf (COTS) hardware for virtualized network functions in telecom equipment. In particular, it explores the conditions under which the COTS hardware can be used in the NFV (Network Function Virtualization) environment. The concept of silent error probability is introduced in order to take software error or undetectable hardware failures into account. The silent error probability is included in both the theoretical work and the simulation work. It is difficult to theoretically analyze the impact of site maintenance and site failure events. Therefore, simulation is used for evaluating the impact of these site management related events which constitute the undesirable feature of using COTS hardware in telecom environment.

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 April 19, 2016.

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
2. Conventions used in this document	4
2.1. Abbreviations	4
3. Network Reliability	5
4. Network Part of the Availability	7
5. Theoretical Analysis of Server Part of System Availability . .	9
6. Simulation Study of Server Part of Availability	12
6.1. Methodology	13
6.2. Validation of the Simulator	16
6.3. Simulation Results	17
6.4. Multiple Servers Sharing the Load	19
7. Conclusions	20
8. Security considerations	21
9. IANA considerations	21
10. Acknowledgements	21
11. References	22
11.1. Normative References	22
11.2. Informative References	22
Authors' Addresses	22

1. Introduction

Using COTS hardware for network functions (e.g. IMS, EPC) have drawn considerable attention in the recent years. Some operators do have legitimate concern regarding the reliability of using the COTS hardware, with reduced MTBF (mean time between failures) and many undesirable attributes of COTS hardware unfamiliar in the traditional telecom industry.

In the previous reliability studies (e.g. GR-77 [1]), the emphasis were place on hardware failures only. In this work, besides hardware failures, which characterized by the MTBF (mean time between failures) and MTTR (mean time to repair), the silent error is also introduced to take account the software error and hardware failure which is undetectable by the management system.

The silent error affecting the system availability in different ways, depending on the particular scenarios.

In a typical system, a server performing certain network functions will have another dedicated server as backup. This is normal master-slave or 1+1 redundancy configuration of the telecom equipment.

The server performing the network function is called the "master server" and the dedicated backup is called the "slave server." In order to differentiate the 1+1 redundancy scheme and 1:1 redundancy scheme, the slave server is deemed "dedicated" for 1+1 case. In 1:1 redundancy, both servers will perform network functions while protecting each other at the same time.

In any protection scheme, on assuming single fault for clarity of discussion, the system availability will not be impacted if the slave part experience silent error and such silent error eventually becoming observable in behavior. In this case, another slave will be identified and the master server will continue to serve the network function. Before the slave server becoming fully functional, the system will operate at reduced error correction capabilities.

On the other hand, if the master server experience the silent error, the data transmitted to the slave server could be corrupted. In this case, the system availability will be impacted when such error becoming observable. On detection of such error, both the master server and the slave server need time to recover. The time for such recovery is fixed in the NFV environment, which is deemed to be a NFV MTTR time. During this time interval, the network function is not available and will be considered to be the downtime in the availability calculations.

Comparing the MTBF of COTS hardware and the typical telecom grade hardware, the COTS may have less MTBF due to its relaxed design criteria.

Comparing the MTTR of COTS hardware and the typical telecom grade hardware, the COTS time to repair is not a random variable and actually is fixed. Hence the COTS MTTR is the time required to bring up a server and ready to serve. In the traditional telecom hardware, the time to repair is a random variable and MTTR is the mean of this random variable. Because manual intervention is normally required in the telecom environment, the NFV COTS MTTR is normally assumed to be less than the traditional telecom equipment MTTR.

The most obvious difference between those two hardware types (COTS hardware and the telecom grade hardware) is related to its maintenance procedure and practice. While telecom equipment takes pains to minimize the impact of maintenance on system availability, the COTS hardware normally is maintained in a cowboy fashion (e.g. reset first and ask questions later).

In this study, a closed solution is available if the site and maintenance related issues are absent for one or two dedicated backup COTS servers in the NFV environment. In order to evaluate the site and maintenance related issues, a simulator is constructed to study the system availability with one or two dedicated backup servers.

It is shown that, with COTS hardware and all its undesirable features, it is still possible to satisfy the telecom requirements under reasonable conditions.

2. 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 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

2.1. Abbreviations

- o A-N: Network Availability
- o A-S: Server Availability

- o A-Sys: System Availability
- o COTS: Commercial Off-The-Shelf
- o DC: Data Center
- o MTBF: Mean Time Between Failures
- o MTTF: Mean Time To Failure
- o MTTR: Mean Time To Repair
- o NFV: Network Function Virtualization
- o PGUP: Protection Group Up Time
- o PSTN: Public Switched Telephone Network
- o SEPSA: Silent Error Probability, Server Availability
- o SDN: Software-Defined Network/Networking
- o TET: Total Elapsed Time
- o VM: Virtual Machine
- o WDT: Weighted Down Time

3. Network Reliability

In the NFV environment, the reliability analysis can be divided into two distinct parts: the server part and the network part, where the network part is to connect all the servers with vSwitch and the server part is to provide the actual network functions. This can be illustrated by using a diagram as shown in Figure-1.

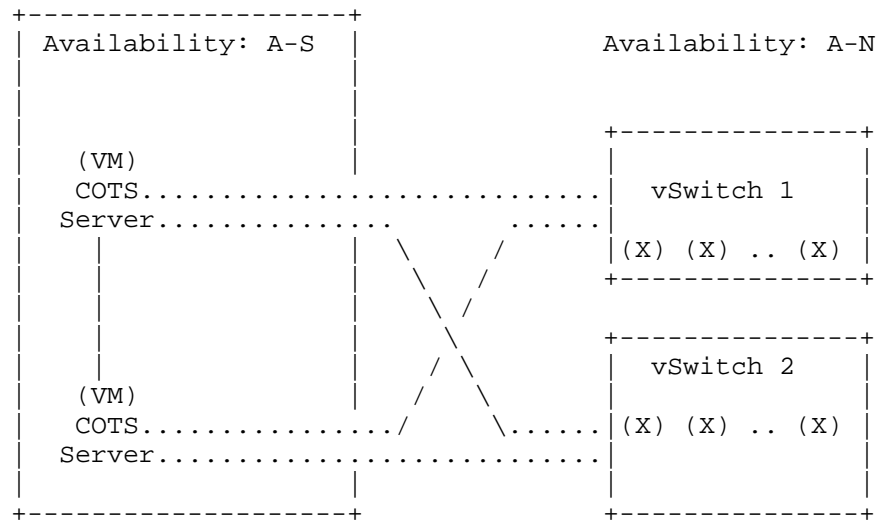


Figure 1: System Availability - Network Part and Server Part

If the overall system availability is denoted by the symbol (A-Sys), the overall system availability will be the product of server part of the system availability (A-S) and the network part of the system availability (A-N).

$$EQ(1) \dots \dots \dots A-Sys = [A-S \times A-N]$$

Given the fact that both A-S and A-N are "less than" 1 (one), we have A-Sys "less than" A-S and A-Sys "less than" A-N. In another words, if FIVE 9s are required for system availability, both the server part and the network part of the availability need to be better than FIVE 9s so their products can be more than FIVE 9s.

To improve the network part of the network availability, as illustrated in Figure 1, the normal 1+1 protection scheme is utilized. It shall be noted that it is possible for the vSwitch to cover long distance transmission network to connect multiple data centers.

The mechanisms in the server part for improving availability is not specified. In this study, it is assumed that one active server will be supported by one or two backup servers. Normally, if the active server is faulty, one of the backup server(s) will take over the responsibility and hence there will be no loss of availability on the server part.

There is a significant difference between the NFV environment and the dedicated traditional telecom equipment related to the time to recover from the server fault. In the traditional telecom equipment case, a manual change of some equipment (e.g. a faulty board) is normally required and hence the time for restoration after experiencing fault, normally denoted as MTTR (Mean Time to Repair) is long.

In the NFV environment, the time for restoration is the time required to boot another virtual machine (VM) with the needed software and re-synchronization of data. Hence the MTTR in the NFV environment can be considered to be shorter than the traditional telecom equipment. More importantly, the MTTR in the NFV environment can be considered to be a fixed constant.

It is also understood that multiple servers will be active to share the load. Contradictory to common sense believe, this arrangement will not increase nor decrease the overall network availability if those active servers are supported by one or two backup servers. This fact will be elaborated in the later section from both theoretical point of view and simulations.

4. Network Part of the Availability

The traditional analysis can be applied to the network part of the availability. In fact, the network part of the availability is impacted by the availability of the switch which is part of the vSwitch and the maximum hops in the vSwitch. The vSwitch is to connect the VMs in the NFV environment.

If A_n is the denote the availability of the network element, for a vSwitch with maximum of h hops, the availability of the vSwitch would be " $(A_n)^h$." Hence, considering the 1+1 configuration of the vSwitch, the A_N can be expressed by

$$EQ(2) \dots \dots \dots A_N = [1 - (1 - (A_n)^h)^2]$$

The network availability, as a function of number of hops (h) and the per network element availability (A_n), can be illustrated by using the diagram as shown in Figure-2.

While this 3-D illustration shows the general trend in network availability, the following data table is able to give more details regarding the network availability with different hop counts and different network element availability, as shown in Table-1.

Table-1: Network Part of System Availability with Various Network

Elements Availability and Hop Counts

Network Element Availability/ Hop	10	16	22	26	30
0.99	0.99086	0.977935	0.96065	0.94712	0.932244
0.999	0.99990	0.999748	0.999526	0.999341	0.999126
0.9999	0.99999	0.999997	0.999995	0.999993	0.999991
0.99999	1	1	1	1	1

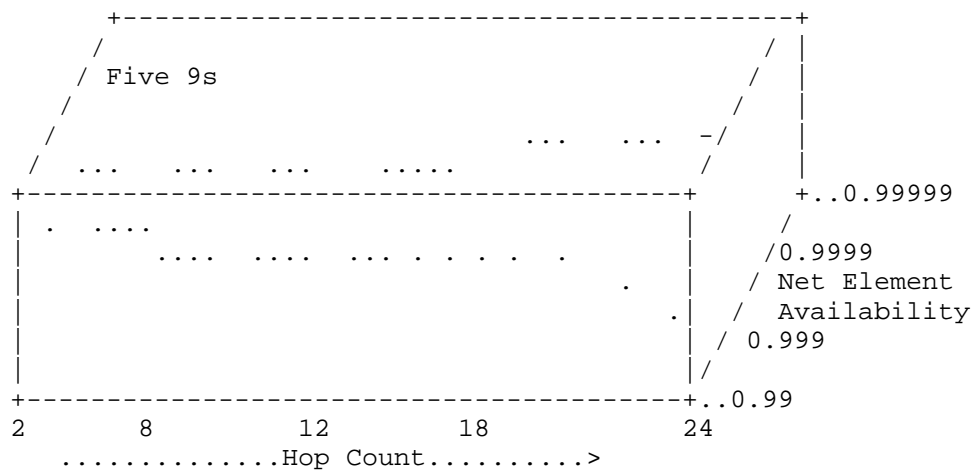


Figure 2: Network Part of the System Availability with Different Hop Counts and Different Network Element Availability

In order to achieve FIVE 9s reliability normally demanded by the telecommunication operators, the network element reliability needs to be at least FOUR 9s if hop counts is more than 10.

In fact, in order to achieve FIVE 9s while per network element availability is only THREE 9s, the hop count needs to be less than two, which is deemed non-practical.

5. Theoretical Analysis of Server Part of System Availability

In GR-77 [1], extensive analysis has been performed for systems under various conditions. In the NFV environment, if the server's availability is denoted as the symbol (A_x), the server part of the system availability (A_s), with a 1+1 master and slave configuration, can be given by [1] in Part D, Chapter 6.

$$\text{EQ(3)} \dots \dots \dots A_s = [1 - ((1 - A_x)^2)]$$

In a more practical environment, there will be silent errors (errors can not be detected by the system under consideration). The silent error probability will be expressed as the symbol (P_{se})

We need to further assume that the silent error only affects the master of the system because it is the one which has the ability to corrupt the data. This assumption can be further articulated, in practical engineering terms, is that "when there is error detected and there is no obvious cause of the error, the master of the "master-slave" configuration will assume the master is correct while the slave will go through a MTTR time to recover."

The state transition can be illustrated as in the following diagram:

Figure 3: State Transition for System with only one Backup, ...
(Note: a dot-and-dash version of the diagram is being developed)....

With this state transition diagram outlined in Figure 1, the system availability in a 1+1 master-slave configuration can be expressed as follows.

$$\text{EQ(4a)} \dots \dots \dots A_s = [1 - ((1 - A_x)^2 + P_{se}A_x(1 - A_x))]$$

$$\text{EQ(4b)} \dots \dots \dots A_s = (2 - P_{se})A_x - (1 - P_{se})((A_x)^2)$$

The following diagram (Figure 4) illustrates the server part of the availability with different per server availability and different silent error probability.

Figure 4: Server Part of the System Availability with Various Server Availability and Silent Error Probability ... (Note: a dot-and-dash version of the diagram is being developed)....

While the graphics illustrate the trends, the following data table will give precise information on a single backup (1+1) configuration.

Table-2: Server Part of availability for different silent error probability and different server availability for single backup

configuration

SEPSA	0.99000	0.99900	0.99990	0.99999
0.0	0.9999	0.999999	0.99999999	1.0
0.1	0.99891	0.9998991	0.999989991	0.99999
0.2	0.99792	0.9997992	0.999979992	0.999998
0.3	0.99693	0.9996993	0.999969993	0.999997
0.4	0.99594	0.9995994	0.999959994	0.999996
0.5	0.99495	0.9994995	0.999949995	0.999995
0.6	0.99396	0.9993996	0.999939996	0.999994
0.7	0.99297	0.9992997	0.999929997	0.999993
0.8	0.99198	0.9991998	0.999919998	0.999992
0.9	0.99099	0.9990999	0.999909999	0.999991
1	0.99	0.999	0.9999	0.99999

In the above table, the entries in the right-most (5th) column, the top entry in the 4th column, and the top-most entry in the 3rd column, outline the area which five 9 availability is possible. As evidenced in the above table, the server part of the network availability deteriorates rapidly with silent error probability. It is possible to achieve five 9s of availability with only server availability of only three 9s, it does demand five 9s server availability when the silent error probability is only 10%.

While the 1+1 configuration illustrated above seems reasonable for server part of the system availability (A_s), there may be cases demanding more than 1+1 configuration for reliability. For systems with two backups, the availability, without consideration of the silent error, can be expressed as [1] (Part D, chapter 6)

$$EQ(5) \dots \dots \dots A_s = [1 - ((1 - A_x)^3)]$$

With the introduction of the silent error probability, the error transition can be expressed in the following diagram:

Figure 5: Error State Transition for System with only two Backups ...
(Note: a dot-and-dash version of the diagram is being developed)....

With the introduction of the silent error and observing the error transition above, assuming the silent error event and the server fault event are independent (e.g., A software error as cause of the silent error and the server fault event as a hardware failure), the server part of the availability for dual backup case can be given by

$$EQ(6a) \dots \dots A_s = 1 - ((1 - A_x)^3 + P_{se}(1 - A_x)((A_x)^2 + 2A_x(1 - A_x)))$$

$$\text{EQ(6b)} \dots \dots A_s = (3-2P_{se})A_x - 3(1-P_{se})(A_x)^2 + (1-P_{se})(A_x)^3$$

It should be noted that, when " $P_{se} = 1$ " for both EQ (4) and EQ (6), the server part of the system availability (A_s) and the server availability (A_x) are the same. This relationship shall be expected since, if the mater always experiences the silent error, the backups are useless and will be corrupted all the time.

The system availability, with dual backup, can be illustrated as follows for different server availability and different silent error including software malfunctions.

Figure 6: Server Part of the System Availability with Various Silent Error Probability and Server Availability for a dual Backup System ... (Note: a dot-and-dash version of the diagram is being developed)....

As with previous case, the diagram will only illustrate the trend while the following table will provide precise data for the system availability under different silent error probability and server availabilities for dual backup case

Table-3: System Availability with different silent error probability and server availability (SEPSA) for dual backup configuration

SEPSA	0.99000	0.99900	0.99990	0.99999
0.0	0.999999	0.99999999	1.0	1.0
0.1	0.9989991	0.99989999	0.99999	0.999999
0.2	0.9979992	0.99979999	0.99998	0.999998
0.3	0.9969993	0.99969999	0.99997	0.999997
0.4	0.9959994	0.99959999	0.99996	0.999996
0.5	0.9949995	0.9995	.99995	0.999995
0.6	0.9939996	0.9994	0.99994	0.999994
0.7	0.9929997	0.9993	0.99993	0.999993
0.8	0.9919998	.9992	.99992	0.999992
0.9	0.9909999	0.9991	0.99991	0.999991
1.0	0.99	0.999	0.9999	0.999990

As shown in Table-3, the entries in the right-most (5th) column, top two entries in the 4th column, and topmost entries in the 2nd and 3rd columns, represent the five 9s capabilities. Comparing those two tables, the dual backups are of marginal advantage over the single backup except for the case there is no silent error. In this case, with only two 9s server availability, the five 9s server part of system availability can be achieved.

From the data above, we can conclude that the silent error, introduced by software error or hardware error not detectable by software, plays an important role in the server part of the system availability and hence the final system availability. In fact, it will be the dominant elements if Pse is more than 10% when the difference between single backup and dual backup are not significant.

Some operators are of the opinion that there need to be a new approach to the availability requirements. COTS hardware are assumed to have less availability than the traditional telecom hardware. But, in the NFV environment, since each server (or VM) in the NFV environment will only affect a small number of users, the requirements for traditional five 9s could be relaxed while keeping the same user experience in downtime. In another words, the weighted downtime, in proportion of the number of users, may be reduced in the NFV environment due to each server affecting only a small number of users for a given server reliability.

Unfortunately, from the theoretical point of view, this is not true. It is possible that, each server downtime will only affect a small number of users. But the multiple active servers will experience more server fault opportunities (this is similar to the famous reliability argument for the twin engine Boeing 777). As long as the protection scheme, or more importantly, the number of backup(s) are the same, the eventual system availability will be the same, regardless what portion of users each server is to serve.

6. Simulation Study of Server Part of Availability

In the above theoretical analysis of server part of availability, the following factors are not considered: (A) Site Maintenance (e.g. software upgrade, patch, etc. affecting the whole site, (B) Site Failure (earth quake, etc.)

While traditional telecom grade equipment putting a lot of emphasis and engineering complexity to ensure smooth migration, smooth software upgrade, and smooth patching procedures, the COTS hardware and its related software are notorious in lacking such capabilities. This is the primary reason for operators to be hesitate on utilizing COTS hardware, even though COTS hardware in the NFV environment does having the improved MTTR as compared to the traditional telecom hardware.

While it is relative easy to obtain a closed for of system availability for the ideal case without site related issues, it is extremely difficult to obtain an analytical solution when site issues are involved. In this case, we resort to numerical simulation under

reasonable assumptions [2, 3, 4].

6.1. Methodology

In this section, the various assumptions and the outline of the simulation mechanisms will be discussed.

A discrete event simulator is constructed to obtain the availability for the server part. In the simulator, an active server (master server which processing the network traffic) will be supported by 1 (single backup) or 2 (dual backup) servers in another site(s).

For the failure probability of the server, it is common to assume the bathtub probability distribution (WeiBull distribution). In practice, we need to enforce that the NFV management will provide servers which is on the flat part of the bathtub distribution. In this case, the familiar exponential distribution can be utilized.

In the discrete event simulator, each server will be scheduled to work for a certain duration of time. This duration will be a random variable with exponential distribution which is common to measure the server behavior during its useful life cycle, with mean given by the MTBF of the server.

In fact, the flat part of the bathtub distribution can related to the normal server MTBF (mean time between failures) with the failure density function expressed as $F(x) = [(1/MTBF) \text{ times } (e^{-x/MTBF})]$.

After the working duration, the server will be down for a fixed time duration which represents the time duration to start another virtual machine to replace the one in trouble. This part is actually different from the traditional telecom grade equipment. Here, the assumption is that there will be another server available to replace the one went down. Hence, regardless the nature of the fault, the down time for a server fault will be fixed which represent the time needed to have another server ready to take over the task.

The following diagram shows this arrangement for a system with only one backup. It shall be noted that, while the server up time duration is variable, the server down time will be fixed.

Figure 7: The life of the Servers ... (Note: a dot-and-dash version of the diagram is being developed)....

The servers will be hosted in "sites" which is considered to be data centers. In this simulation, during initial setup, the servers supporting each other for reliability purposes will be hosted in

different sites. This is to minimize the impact of the site failure and site maintenance.

In order to model the system behavior with one or two backups, the concept of protection group is introduced.

A protection group will consists of a "master" server with one or two "slave" server(s) in other site(s). There may be multiple protection groups inside the network with each protection group serving a fraction of the users.

A protection group will be considered to be "down" if every server in this group is dead. During the time the protection group is "down", the network service will be affected and the network is considered to be "down" for the group of users this protection group is responsible for.

The uptime and downtime of the protection group will be recorded in the discrete event simulator. The server part of the availability is given by (where the total elapsed time is the total simulation time in the discrete event simulator)

EQ(7) Availability(server part)= [(PGUP)/(TET)], whereas

- o PGUP is Protection Group Up Time

- o TET is the Total Elapsed Time

The concept of protection group, site, and server can be illustrated as follows (Figure 8) for a system with two backups. It shall be noted that the protection group is an abstract concept and the portion of the network function is not available if and only if the all the servers in the protection group is not functioning.

Figure 8: Servers, Sites, and Protection Group ... (Note: a dot-and-dash version of the diagram is being developed)....

Even though the simulator will allow each site to have a number of servers, which is configurable, there is little use for this arrangement. The system availability will not change regardless how many servers per site is used to support the system, as long as there is no change in the number of servers in the protection group. The increase of number of servers per site is essentially increase the number of protection groups. For a long time duration, each protection group will experience the similar downtime for the same up time (or will have the same availability).

As in the theoretical analysis, the silent error, due to software or

subtle hardware failure, will only affect the active (or master) server. When the master server failed with silent error, both the master and "slave" servers will go through a MTTR time to recover (e.g. time to incarnate two VMs simultaneously). In this case, this part of the system (or this protection group) is considered to be under fault.

In the reliability study, the focus is the number of the backups for each protection group where 1+1 configuration is a typical configuration for one backup mechanism. For load sharing arrangement such as 1:1, it can be viewed as two protection groups.

In general, the load sharing scheme will have less availability because, in 1:1 case, any server fault will result in two faults in different protection groups. This can be extended to 1:2 case where three protection groups are involved, and any server fault will introduce three faults in different protection groups. In this study, the load sharing mechanisms will not be elaborated further.

The site will also go through its maintenance work. The traditional telecom grade equipment and the COTS hardware mainly defers on this part. In Telecom grade equipment, minimum impact on system performance or system availability is to be maintained during the maintenance window. But, for COTS hardware, the maintenance work may be more frequently and more destructive.

In order to simulate the maintenance aspect of COTS hardware, the simulator will put the site "under maintenance" at random time. The interval for the site to be working is also assumed to be exponentially distributed random variable, with mean to be configurable in the simulator. The duration of the maintenance is also a uniform distributed random variable with a configured mean, minimum, and maximum.

In order to put a site "under maintenance", there shall be no-fault inside the network. All the servers on the site to be "under maintenance" will be moved to other site. Hence, no traffic will be impacted during the process of putting the site under maintenance. Of course, the ability against site failure when some site is under maintenance will be reduced.

When a site is back from maintenance, it will attempt to claim all its server responsibilities transferred due to site maintenance.

- o For each protection group, if every server is working, the protection group will re-arrange the protect relationship so each site will only have one server in the protection group. The new server on the site back from maintenance will need a MTTR time to

be ready for backup. In this case, no loss of service in the system.

- o For each protection group, if there are at least one working and at least one in fault condition, one working server will be added to the protection group. The new server on the site back from maintenance will need a MTTR time to be ready for backup. In this case, no loss of service in the system.
- o For each protection group, if there is no servers working, the protection group will gain a working server from the site back from the maintenance. The new server on the site back from maintenance will need a MTTR time to be ready for service. In this case, the system will provide service after the new server is ready.

A site can also under fault (e.g. loss of power, operating under reduced capability due to thermal issues, and earth quake). The simulator can also simulate the effect of such events, with site up duration as an exponentially distributed random variable with mean to be configured. The site failure duration is expressed as a uniform distributed random variable with configurable mean, minimum, and maximum.

6.2. Validation of the Simulator

In order to verify the correctness of the simulator (e.g. the random number generator, the whole program structure, etc.), the simulation is performed with various server availability and various silent error probability.

For single backup case, the error between the theoretical data and simulation data for system availability on the server part can be illustrated by the following diagram (Figure 9).

Figure 9: Verification of Simulator for Single Backup Case ...
(Note: a dot-and-dash version of the diagram is being developed)....

As we can see, the magnitude of the errors are within 10-to-the-power-(-5) which is very small, considering the nominal value of system availability for the server part is close to 1.0. For the dual backup case, the error between the simulated and theoretical system availability for different silent error probability and server availability can be illustrated as follows (Figure 10).

Figure 10: Verification of Simulator for Dual Backup Case ... (Note: a dot-and-dash version of the diagram is being developed)....

This is also similar to that of the single backup case where the error are within the range. Those error information gives us the needed confidence on the simulation result for complicated case where analytical solutions are evasive.

6.3. Simulation Results

The effect of the MTTR in the NFV environment is studied first. In this study, the effect of the MTTR and the silent error probability can be shown below:

Figure 11: Availability with Various Silent for different MTTRs... (Note: a dot-and-dash version of the diagram is being developed)....

In the diagram (Figure 11), R6 represents MTTR of 6 minutes while R60 represents MTTR of 60 minutes. The x-axis is the silent error probability. As shown, the effect of the MTTR (time to recover from a fault or time to have VM rebirth) will affect the slope of the system availability, which decline with the increase of silent error probability. In the above example, the server MTBF is assumed to be 10000 hours which represents the server availability of 0.9994 for R6 case and 0.994 for the R60 case.

The two curves starting approximate 1.0 are the system availability with dual backups while the other two are for system availability with single backup. It should be noted that, for the dual backup case, there is little difference in availability for different MTTR when there is no silent error. Intuitively, this is expected due to the added number of backup servers.

In this simulation, both site failure (with mean time between failures of 20000 hours) and site maintenance (with mean time between site maintenance of 1000) are considered. The mean time for site failure duration is assumed to be 12 hours (uniform distributed between 4 hours and 24 hours) and the mean time for site maintenance is 24 hours (uniform distributed between 4 hours and 48 hours).

The next step is to evaluate the impact of the site issues (site failure, maintenance). For a very bad site outlined above, which has the mean time between site failures to be 2 times of the server MTBF and the mean time between site maintenance events is assumed to be 0.1 times of the server MTBF. The availability on the server part can be illustrated with different silent error probability and server availability for the single backup configuration.

Figure 12: Availability for the Server Part in Single Backup Configuration... (Note: a dot-and-dash version of the diagram is being developed)....

As the data will illustrate that, in order to achieve high availability, the server availability needs to be very high. In fact, the server availability needs to be in the range of FIVE 9s in order to achieve the system availability of FIVE 9s under various site related issues. The dual backup systems for exactly the same configuration, the result will be better and can be illustrated as follows:

Figure 13: Availability for the Server Part in Single Backup Configuration... (Note: a dot-and-dash version of the diagram is being developed)....

With server availability of FOUR 9s, and with low silent error probabilities, the server part of the availability can achieve FIVE 9s. For a site with less issues, such as the one with mean time between failures is 100 times of the server MTBF and site maintenance is 0.1 times of the server MTBF. The mean time for site failure duration is also assumed to be 12 hours (uniform distributed between 4 hours and 24 hours) and the mean time for site maintenance is 24 hours (uniform distributed between 4 hours and 48 hours). The result for the single backup system can be shown as follows:

Figure 14: Server Part of Availability for a Good Site on Single Backup... (Note: a dot-and-dash version of the diagram is being developed)....

The following data table (Table-4) will give precise information regarding this simulation results.

Table-4: Details Regarding Availability on Server Part for Single Backup on a Good Site

Silent Error/Server Availability	0.990099	0.999001	0.99990001	0.99999
0.0	0.998971	0.999959	0.9999992	1.0
0.1	0.997918	0.999857	0.99998959	0.99999901
0.2	0.996908	0.999771	0.99997957	0.99999804
0.3	0.995999	0.999674	0.99996935	0.99999695

As evidenced in the table above, the server part of the system availability will be impacted by the silent error and a single redundant hardware will only provide marginal improvement when the silent error probability is small.

Figure 15: Server Part of Availability for a Good Site on Dual Backup... (Note: a dot-and-dash version of the diagram is being developed)....

The diagram above give a general trend in system availability and the follow data table will precise the data.

Table 5: Details Regarding Availability on Server Part for Dual Backup on a Good Site

Silent Error/Server Availability	0.99009901	0.999001	0.99990001	0.99999
0.0	0.99999939	0.99999998	1.0	1.0
0.2	0.9981346	0.99980209	0.99998048	0.99999792
0.4	0.99615083	0.99960136	0.99996002	0.99999594
0.5	0.99522474	0.9995184	0.99995225	0.99999503

From the tables for single and dual backup, we can see that dual backup only provides marginal benefit in the face of site issues. Given the fact that site issues are inevitable in practice, a geographically distributed single backup system is recommended for simplicity.

6.4. Multiple Servers Sharing the Load

In this section, we outline the simulation results for cases when there are multiple servers to take care of the active work load. In this case, the impact of a protection group failure will affecting smaller number of users.

In the simulation, each site will have N servers to serve the work. A weighted uptime and weighted down time was introduced. The system availability is the weighted uptime divided by the total of weighted uptime and weighted downtime.

EQ(8)... ... Weighted-Availability[Server-Part]=[(TET - WDT)/TET],
whereas

- o TET is the Total Elapsed Time
- o WDT is the Weighted Down Time

If any protection group (i) is down, the WDT will be updated as follows:

EQ(9)... .. WDT = WDT + [Protection Group (i) Down Time]/N

For a system with three protection groups (i.e. the servers sharing the workload), the availability of each protection group, as well as the weighted availability, are obtained as follows (Table-6):

Table-6: Availability of Protection Groups and the Weighted Availability (Dual Backup)

Availability / Silent Error Probability	Availability of Protection Group 1	Availability of Protection Group 2	Availability of Protection Group 3	Measured Weighted Availability	Protection Group Average - Weighted Availability
0.0	1.0	1.0	1.0	1.0	0.0
0.2	0.999998015	0.999998005	0.999997985	0.999998001	6.66668E-11
0.4	0.999996027	0.999996018	0.999995988	0.999996011	-3.33333E-11

In this case, there is little difference between the different protection groups. The weighted availability is actually the average of availability of all the protection groups. This also illustrates the fact that, regardless of how many servers share the active load, the system availability will be the same as long as (A) The number of backups are the same, and (B) Each server availability is the same.

7. Conclusions

The system availability can be divided into two parts; the availability from the network and the availability from the server. The final system availability is the product of those two parts.

The system availability from the network is determined by the maximum number of hops and individual network element availability, with the fault tolerant setup is assumed to be 1+1. The system availability from the server is mainly determined by the following parameters.

- o Availability of each individual server
- o Silent error probability

- o Site related issues (maintenance, fault)
- o Protection Scheme (one or two dedicated backups)

The introduction of silent error is to take account of software error and errors undetectable by hardware, the system availability on the server part will be dominated by such silent error if the silent error probability is more than 10%. This is shown in both theoretical work and simulations.

It shall be interesting to note that the dual backup scheme provides marginal benefits and the added complexity may not warrant such practice in the real network.

It is possible for COTS hardware to provide as high availability as the traditional telecom hardware if the server itself is of reasonable high-availability. The undesirable attributes of COTS hardware have been modelled into the site related issues, such as site maintenance and site failure which is not applicable for traditional telecom hardware. Hence, in calculating the server availability, the site related issues are to be excluded.

It is critical for the virtualization infrastructure management to provide as much hardware failure information as possible to improve the availability of the application. As seen in both theoretical work and simulation, the silent error probability becomes a dominant factor in the final availability. The silent error probability can be reduced if the virtualization infrastructure management is capable of fault isolation.

8. Security considerations

To be determined.

9. IANA considerations

This Internet Draft includes no request to IANA.

10. Acknowledgements

Authors would like to thank the NFV RG chairs (Diego and Ramki) for encouraging discussions and guidance.

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>>.
- [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-01 (work in progress), August 2015.
- [1] GR-77, "Applied R&M Manual for Defense Systems", 2012.

11.2. Informative References

- [2] Papoulis, A., "Probability, Random Variables, and Stochastic Processes", 2002.
- [3] Bremaud, P., "An Introduction to Probabilistic Modeling", 1994.
- [4] Press, et al, W., "Numerical Recipes in C/C++", 2007.

Authors' Addresses

Li Mo
ZTE (TX) Inc.
2425, N. central expressway
Richardson, TX 75080
USA

Phone: +1-972-454-9661
Email: li.mo@ztetx.com

Bhumip Khasnabish (editor)
ZTE (TX) Inc.
55 Madison Avenue, Suite 160
Morristown, New Jersey 07960
USA

Phone: +001-781-752-8003
Email: vumipl@gmail.com, bhumip.khasnabish@ztetx.com
URI: <http://tinyurl.com/bhumip/>

NFVRG
Internet Draft
Category: Informational

S. Natarajan
Google
R. Krishnan
A. Ghanwani
Dell
D. Krishnaswamy
IBM Research
P. Willis
BT
A. Chaudhary
Verizon
F. Huici
NEC

Expires: January 2017

July 8, 2016

An Analysis of Lightweight Virtualization Technologies for NFV
draft-natarajan-nfvrg-containers-for-nfv-03

Abstract

Traditionally, NFV platforms were limited to using standard virtualization technologies (e.g., Xen, KVM, VMWare, Hyper-V, etc.) running guests based on general-purpose operating systems such as Windows, Linux or FreeBSD. More recently, a number of light-weight virtualization technologies including containers, unikernels (specialized VMs) and minimalistic distributions of general-purpose OSes have widened the spectrum of possibilities when constructing an NFV platform. This draft 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.

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/lid-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 January 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.

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. Lightweight Virtualization Background.....	3
2.1. Containers.....	3
2.2. OS Tinyfication.....	3
2.3. Unikernels.....	4
3. Challenges in Building NFV Platforms.....	4
3.1. Performance (SLA).....	4
3.1.1. Challenges.....	4
3.2. Continuity, Elasticity and Portability.....	5
3.2.1. Challenges:.....	5
3.3. Security.....	6
3.3.1. Challenges.....	6
3.4. Management.....	7
3.4.1. Challenges.....	8
4. Benchmarking Experiments.....	8
4.1. Experimental Setup.....	8

4.2. Instantiation Times.....	9
4.3. Throughput.....	9
4.4. RTT.....	10
4.5. Image Size.....	11
4.6. Memory Usage.....	11
5. Discussion.....	12
6. Conclusion.....	13
7. Future Work.....	13
8. IANA Considerations.....	14
9. Security Considerations.....	14
10. Contributors.....	14
11. Acknowledgements.....	14
12. References.....	14
12.1. Normative References.....	14
12.2. Informative References.....	14
Authors' Addresses.....	16

1. Introduction

This draft describes the challenges when building an NFV platform by describing to what extent different types of lightweight virtualization technologies, such as VMs based on minimalistic distributions, unikernels and containers, are able to address them.

2. Lightweight Virtualization Background

2.1. Containers

Containers are a form of operating-system virtualization. To provide isolation, containers such as Docker rely on features of the Linux kernel such as cgroups, namespaces and a union-capable file system such as aufs and others [AUFSS]. Because they run within a single OS instance, they avoid the overheads typically associated with hypervisors and virtual machines.

2.2. OS Tinyfication

OS tinyfication consists of creating a minimalistic distribution of a general-purpose operating system such as Linux or FreeBSD. This involves two parts: (1) configuring the kernel so that only needed features and modules are enabled/included (e.g., removing extraneous drivers); and (2) including only the required user-level libraries and applications needed for the task at hand, and running only the minimum amount of required processes. The most notable example of a tinyfied OS is the work on Linux tinyfication [LINUX-TINY].

2.3. Unikernels

Unikernels are essentially single-application virtual machines based on minimalistic OSes. Such minimalistic OSes have minimum overhead and are typically single-address space (so no user/kernel space divide and no expensive system calls) and have a co-operative scheduler (so reducing context switch costs). Examples of such minimalistic OSes are MiniOS [MINIOS] which runs on Xen and OSv [OSV] which runs on KVM, Xen and VMWare.

3. Challenges in Building NFV Platforms

In this section, we outline the set of main challenges for an NFV platform in the context of lightweight virtualization technologies as well as traditional VMs.

3.1. Performance (SLA)

Performance requirements vary with each VNF type and configuration. The platform should support the specification, realization and runtime adaptation of different performance metrics. Achievable performance can vary depending on several factors such as the workload type, the size of the workload, the set of virtual machines sharing the underlying infrastructure, etc. Here we highlight some of the challenges based on potential deployment considerations.

3.1.1. Challenges

- . VNF provisioning time (including up/down/update) constitutes the time it takes to spin-up the VNF process, its application-specific dependencies, and additional system dependencies. The resource choices such as the hypervisor type, the guest and host OS flavor and the need for hardware and software accelerators, etc., constitute a significant portion of this processing time (instantiation or down time) when compared to just bringing up the actual VNF process.
- . The runtime performance (achievable throughput, line rate speed, maximum concurrent sessions that can be maintained, number of new sessions that can be added per second) for each VNF is directly dependent on the amount of resources (e.g., virtual CPUs, RAM) allocated to individual VMs. Choosing the right resource setting is a tricky task. If VM resources are over-provisioned, we end up under-utilizing the physical resources. On the contrary if we under-provision the VM resources, then upgrading the resource to an advanced system setting might require scaling out or scaling up of the resources and re-directing traffic to the new VM; scaling up/down operations consume time and add to the latency. This

overhead stems from the need to account resources of components other than the actual VNF process (e.g., guest OS requirements).

- . If each network function is hosted in individual VMs/containers, then an efficient inter-VM networking solution is required for performance.

3.2. Continuity, Elasticity and Portability

VNF service continuity can be interrupted due to several factors: undesired state of the VNF (e.g., VNF upgrade progress), underlying hardware failure, unavailability of virtualized resources, VNF SW failure, etc. Some of the requirements that need consideration are:

3.2.1. Challenges:

- o VNF's are not completely decoupled from the underlying infrastructure. As discussed in the previous section, most VNFs have a dependency on the guest OS, hypervisor type, accelerator used, and the host OS (this last one applies to containers too). Therefore porting VNFs to a new platform might require identifying equivalent resources (e.g., hypervisor support, new hardware model, understanding resource capabilities) and repeating the provisioning steps to bring back the VNF to a working state.
- o Service continuity requirements can be classified as follows: seamless (with zero impact) or non-seamless continuity (accepts measurable impacts to offered services). To achieve this, the virtualization technology needs to provide an efficient high availability solution or a quick restoration mechanism that can bring back the VNF to an operational state. For example, an anomaly caused by a hardware failure can impact all VNFs hosted on that infrastructure resource. To restore the VNF to a working state, the user should first provision the VM/container, spin-up and configure the VNF process inside the VM, setup the interconnects to forward network traffic, manage the VNF-related state, and update any dependent runtime agents.
- o Addressing the service elasticity challenges require a holistic view of the underlying resources. The challenges for presenting a holistic view include the following
 - o Performing Scalable Monitoring: Scalable continuous monitoring of the individual resource's current state is needed to spin-up additional resources (auto-scale or auto-

heal) when the system encounters performance degradation or spin-down idle resources to optimize resource usage.

- o Handling CPU-intensive vs I/O-intensive VNFs: For CPU-intensive VNFs the degradation can primarily depend on the VNF processing functionality. On the other hand, for I/O intense workloads, the overhead is significantly impacted by to the hypervisor/host features, its type, the number of VMs/containers it manages, the modules loaded in the guest OS, etc.

3.3. Security

Broadly speaking, security can be classified into:

- o Security features provided by the VNFs to manage the state, and
- o Security of the VNFs and its resources.

Some considerations on the security of the VNF infrastructure are listed here.

3.3.1. Challenges

- o The adoption of virtualization techniques (e.g., para-virtualization, OS-level) for hosting network functions and the deployment need to support multi-tenancy requires secure slicing of the infrastructure resources. In this regard, it is critical to provide a solution that can ensure the following:
 - o Provision the network functions by guaranteeing complete isolation across resource entities (hardware units, hypervisor, virtual networks, etc.). This includes secure access between VM/container and host interface, VM-VM or container-to-container communication, etc. For maximizing overall resource utilization and improving service agility/elasticity, sharing of resources across network functions must be possible.
 - o When a resource component is compromised, quarantine the compromised entity but ensure service continuity for other resources.
 - o Securely recover from runtime vulnerabilities or attacks and restore the network functions to an operational state. Achieving this with minimal or no downtime is important.

Realizing the above requirements is a complex task in any type of virtualization option (virtual machines, containers, etc.)

- o Resource starvation / Availability: Applications hosted in VMs/containers can starve the underlying physical resources such that co-hosted entities become unavailable. Ideally, countermeasures are required to monitor the usage patterns of individual VMs/containers and ensure fair use of individual resources.

3.4. Management

The management and operational aspects are primarily focused on the VNF lifecycle management and its related functionalities. In addition, the solution is required to handle the management of failures, resource usage, state processing, smooth rollouts, and security as discussed in the previous sections. Some features of management solutions include:

- o Centralized control and visibility: Support for web client, multi-hypervisor management, single sign-on, inventory search, alerts & notifications.
- o Proactive Management: Creating host profiles, resource management of VMs/containers, dynamic resource allocation, auto-restart in HA model, audit trails, patch management.
- o Extensible platform: Define roles, permissions and licenses across resources and use of APIs to integrate with other solutions.

Thus, the key requirements for a management solution

- o Simple to operate and deploy VNFs.
- o Uses well-defined standard interfaces to integrate seamlessly with different vendor implementations.
- o Creates functional automation to handle VNF lifecycle requirements.
- o Provide APIs that abstracts the complex low-level information from external components.
- o Is secure.

3.4.1. Challenges

The key challenge is addressing the aforementioned requirements for a management solution while dealing with the multi-dimensional complexity introduced by the hypervisor, guest OS, VNF functionality, and the state of network.

4. Benchmarking Experiments

Having considered the basic requirements and challenges of building an NFV platform, we now provide a benchmark of a number of lightweight virtualization technologies to quantify to what extent they can be used to build such a platform.

4.1. Experimental Setup

In terms of hardware, all tests are run on an x86-64 server with an Intel Xeon E5-1630 v3 3.7GHz CPU (4 cores) and 32GB RAM.

For the hypervisors we use KVM running on Linux 4.4.1 and Xen version 4.6.0. The virtual machines running on KVM and Xen are of three types:

- (1) Unikernels, on top of the minimalistic operating systems OSv and MiniOS for KVM and Xen, respectively. The only application built into them is iperf. To denote them we use the shorthand `unikernel.osv.kvm` or `unikernels.minios.xen`.
- (2) Tinyfied Linux (a.k.a. Tinyx), consisting of a Linux kernel version 4.4.1 with only a reduced set of drivers (`ext4`, and `netfront/blkfront` for Xen), and a distribution containing only `busybox`, an `ssh` server for convenience, and `iperf`. We use the shorthand `tinyx.kvm` and `tinyx.xen`.
- (3) Standard VM, consisting of a Debian distribution including `iperf` and Linux version 4.4.1. We use the shorthand `standardvm.kvm` and `standardvm.xen` for it.

For containers, we use Docker version 1.11 running on Linux 4.4.1.

It is worth noting that the numbers reported here for virtual machines (whether standard, Tinyx or unikernels) include the following optimizations to the underlying virtualization technologies. For Xen, we use the optimized Xenstore, `toolstack` and `hotplug` scripts reported in [SUPERFLUIDITY] as well as the accelerated packet I/O derived from persistent grants (for Tx)

[PGRANTS]. For KVM, we remove the creation of a tap device from the VM's boot process and use a pre-created tap device instead.

4.2. Instantiation Times

We begin by measuring how long it takes to create and boot a container or VM. The beginning time is when we issue the create operation. To measure the end time, we carry out a SYN flood from an external server and measure the time it takes for the container/VM to respond with a RST packet. The reason for a SYN flood is that it guarantees the shortest reply time after the unikernels/container is booted. It is just to measure boot time, nothing to do with real-world deployments and DoS attacks.

Technology Type	Time (msecs)
standardvm.xen	6500
standardvm.kvm	2988
Container	1711
tinyx.kvm	1081
tinyx.xen	431
unikernel.osv.kvm	330
unikernels.minios.xen	31

The table above shows the results. Unsurprisingly, standard VMs with a regular distribution (in this case Debian) fare the worst, with times in the seconds: 6.5s on Xen and almost 3s on KVM. The Docker container with iperf comes next, clocking in at 1.7s. The next best times are from Tinyx: 1s approximately on KVM and 431ms on Xen. Finally, the best numbers come from unikernels, with 330ms for OSv on KVM and 31ms for MiniOS on Xen. These results show that at least when compared to unoptimized containers, minimalistic VMs or unikernels can have instantiation times comparable to or better than containers.

4.3. Throughput

To measure throughput we use the iperf application that is built in to the unikernels, included as an application in Tinyx and the Debian-based VMs, and containerized for Docker. The experiments in this section are for TCP traffic between the guest and the host

where the guest resides: there are no NICs involved so that rates are not bound by physical medium limitations.

Technology Type	Throughput (Gb/s) Tx	Throughput (Gb/s) Rx
standardvm.xen	23.1	24.5
standardvm.kvm	20.1	38.9
Container	45.1	43.8
tinyx.kvm	21.5	37.9
tinyx.xen	28.6	24.9
unikernel.osv.kvm	47.9	47.7
unikernels.minios.xen	49.5	32.6

The table above shows the results for Tx and Rx. The first thing to note is that throughput is not only dependent on the guest's efficiency, but also on the host's packet I/O framework (e.g., see [CLICKOS] for an example of how optimizing Xen's packet I/O subsystem can lead to large performance gains). This is evident from the Xen numbers, where Tx has been optimized and Rx not. Having said that, the guest also matters, which is why, for example, Tinyx scores somewhat higher throughput than standard VMs. Containers and unikernels (at least for Tx and for Tx/Rx for KVM) are fairly equally matched and perform best, with unikernels having a slight edge.

4.4. RTT

To measure round-trip time (RTT) from an external server to the VM/container we carry out a ping flood and report the average RTT.

Technology Type	Time (msecs)
standardvm.xen	34
standardvm.kvm	18
Container	4
tinyx.kvm	19
tinyx.xen	15
unikernel.osv.kvm	9
unikernels.minios.xen	5

As shown in the table above, the Docker container comes out on top with 4ms, but unikernels achieve for all practical intents and purposes the same RTT (5ms on MiniOS/Xen and 9ms on OSv/KVM). Tinyx fares slightly better than the standard VMs.

4.5. Image Size

We measure image size using the standard "ls" tool.

Technology Type	Size (MBs)
standardvm.xen	913
standardvm.kvm	913
Container	61
tinyx.kvm	3.5
tinyx.xen	3.7
unikernel.osv.kvm	12
unikernels.minios.xen	2

The table shows the standard VMs to be unsurprisingly the largest and, followed by the Docker/iperf container. OSv-based unikernels are next with about 12MB, followed by Tinyx (3.5MB or 3.7MB on KVM and Xen respectively). The smallest image is the one based on MiniOS/Xen with 2MB.

4.6. Memory Usage

For the final experiment we measure memory usage for the various VMs/container. To do so we use standard tools such as "top" and "xl" (Xen's management tool).

Technology Type	Usage (MBs)
standardvm.xen	112
standardvm.kvm	82
Container	3.8
tinyx.kvm	30
tinyx.xen	31
unikernel.osv.kvm	52
unikernels.minios.xen	8

The largest memory consumption, as shown in the table above, comes from the standard VMs. The OSv-based unikernels comes next due to the fact that OSv pre-allocates memory for buffers, among other things. Tinyx is next with about 30MB. From there there's a big jump to the MiniOS-based unikernels with 8MB. The best result comes from the Docker container, which is expected given that it relies on the host and its memory allocations to function.

5. Discussion

In this section we provide a discussion comparing and contrasting the various lightweight virtualization technologies in view of the reported benchmarks. There are a number of issues at stake:

- . Service agility/elasticity: this is largely dependent on the ability to quickly spin up/down VMs/containers and migrate them. Clearly the best numbers in this category come from unikernels and containers.
- . Memory consumption: containers use and share resources from the common host they use and so each container instance uses up less memory than VMs, as shown in the previous section (although unikernels are not far behind). Note: VMs also have a common host (or dom0 in the case of Xen) but they incur the overhead of each having its own guest OS.
- . Security/Isolation: an NFV platform needs to provide good isolation for its tenants. Generally speaking, VM-based technologies have been around for longer and so have had time to iron out most of the security issues they had. Type-1 hypervisors (e.g., Xen), in addition, provide a smaller attack surface than Type-2 ones (e.g., KVM) so should in principle be more robust. Containers are relatively newcomers and as such still have a number of open issues [CONTAINER-SECURITY]. Use of kernel security modules like SELinux [SELINUX], AppArmor [APPARMOR] along with containers can provide at least some of the required features for a secure VNF deployment. Use of resource quota techniques such as those in Kubernetes [KUBERNETES-RESOURCE-QUOTA] can provide at least some of the resource guarantees for a VNF deployment.
- . Management frameworks: both virtual machines and containers have fully-featured management frameworks with large open source communities continually improving them. Unikernels might need a bit of "glue" to adapt them to an existing framework (e.g., OpenStack).

- . Compatibility with applications. Both containers and standard VMs can run any application that is able to run on the general-purpose OS those VMs/containers are based on (typically Linux). Unikernels, on the hand, use minimalistic OSes, which might present a problem. OSv, for example, is able to build a unikernels as long as the application can be recompiled as a shared library. MiniOS requires that the application be directly compiled with it (c/c++ is the default, but MiniOS unikernels based on OCaml, Haskell and other languages exist).

Overall, the choice between standard virtual machines, tinyfied ones, unikernels or containers is often not a black and white one. Rather, these technologies present points in a spectrum where criteria such as security/isolation, performance, and compatibility with existing applications and frameworks may point NFV operators, and their clients, towards a particular solution. For instance, an operator for whom excellent isolation and multi-tenancy is a must might lean towards hypervisor-based solutions. If that operator values ease of application deployment he will further choose guests based on a general-purpose OS (whether tinyfied or not). Another operator might put a prime on performance and so might prefer unikernels. Yet another one might not have a need for multi-tenancy (e.g., Google, Edge use cases such as CPE) and so would lean towards enjoying the benefits of containers. Hybrid solutions, where containers are run within VMs, are also possible. In short, choosing a virtualization technology for an NFV platform is (no longer) as simple as choosing VMs or containers.

6. Conclusion

In this draft we presented the challenges when building an NFV platform. We further introduced a set of benchmark results to quantify to what extent a number of virtualization technologies (standard VMs, tinyfied VMs, unikernels and containers) can meet those challenges. We conclude that choosing a solution is nuanced, and depends on how much value different NFV operators place on criteria such as strong isolation, performance and compatibility with applications and management frameworks.

7. Future Work

Opportunistic areas for future work include but not limited to developing solutions to address the VNF challenges described in Section 3, distributed micro-service network functions, etc.

8. IANA Considerations

This draft does not have any IANA considerations.

9. Security Considerations

VM-based VNFs can offer a greater degree of isolation and security due to technology maturity as well as hardware support. Light-weight virtualization technologies such as unikernels (specialized VMs) and tinyfied VMs which were discussed enjoyed the security benefits of a standard VM. Since container-based VNFs provide abstraction at the OS level, it can introduce potential vulnerabilities in the system when deployed without proper OS-level security features. This is one of the key implementation/deployment challenges that needs to be further investigated.

In addition, as containerization technologies evolve to leverage the virtualization capabilities provided by hardware, they can provide isolation and security assurances similar to VMs.

10. Contributors

11. Acknowledgements

The authors would like to thank Vineed Konkoth for the Virtual Customer CPE Container Performance white paper. The authors would like to acknowledge Louise Krug (BT) for their valuable comments.

12. References

12.1. Normative References

12.2. Informative References

[AUFS] "Advanced Multi-layered Unification Filesystem,"
<https://en.wikipedia.org/wiki/Aufs>

[CONTAINER-SECURITY] "Container Security article,"
<http://www.itworld.com/article/2920349/security/for-containers-security-is-problem-1.html>

[ETSI-NFV-WHITE] "ETSI NFV White Paper,"
http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[ETSI-NFV-USE-CASES] "ETSI NFV Use Cases,"
http://www.etsi.org/deliver/etsi_gs/NFV/001_099/001/01.01.01_60/gs_NFV001v010101p.pdf

[ETSI-NFV-REQ] "ETSI NFV Virtualization Requirements,"
http://www.etsi.org/deliver/etsi_gs/NFV/001_099/004/01.01.01_60/gs_NFV004v010101p.pdf

[ETSI-NFV-ARCH] "ETSI NFV Architectural Framework,"
http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf

[ETSI-NFV-TERM] "Terminology for Main Concepts in NFV,"
http://www.etsi.org/deliver/etsi_gs/NFV/001_099/003/01.01.01_60/gs_nf003v010101p.pdf

[KUBERNETES-RESOURCE-QUOTA] "Kubernetes Resource Quota,"
<http://kubernetes.io/v1.0/docs/admin/resource-quota.html>

[KUBERNETES-SELF-HEALING] "Kubernetes Design Overview,"
<http://kubernetes.io/v1.0/docs/design/README.html>

[LINUX-TINY] "Linux Kernel Tinification,"
<https://tiny.wiki.kernel.org/>

[MINIOS] "Mini-OS - Xen," <http://wiki.xenproject.org/wiki/Mini-OS>

[OSV] "OSv - The Operating System Designed for the Cloud,"
<http://osv.io/>

[PGRANTS] <http://lists.xenproject.org/archives/html/xen-devel/2015-05/msg01498.html>

[SELINUX] "Security Enhanced Linux (SELinux) project,"
<http://selinuxproject.org/>

[SUPERFLUIDITY] "The Case for the Suplerfluid Cloud," F. Manco, J. Martins, K. Yasukata, J. Mendes, S. Kuenzer, and F. Huici. USENIX HotCloud 2015

[APPARMOR] "Mandatory Access Control Framework,"
<https://wiki.debian.org/AppArmor>

[VCPE-CONTAINER-PERF] "Virtual Customer CPE Container Performance White Paper," <http://info.ixiacom.com/rs/098-FRB-840/images/Calsoft-Labs-CaseStudy2015.pdf>

Authors' Addresses

Sriram Natarajan
Google
natarajan.sriram@gmail.com

Ram (Ramki) Krishnan
Dell
ramki_krishnan@dell.com

Anoop Ghanwani
Dell
anoop@alumni.duke.edu

Dilip Krishnaswamy
IBM Research
dilikris@in.ibm.com

Peter Willis
BT
peter.j.willis@bt.com

Ashay Chaudhary
Verizon
the.ashay@gmail.com

Felipe Huici
NEC
felipe.huici@neclab.eu

NFVRG
Internet-Draft
Intended status: Informational
Expires: April 21, 2016

R. Rosa
C. Rothenberg
Unicamp
R. Szabo
Ericsson
October 19, 2015

VNF Benchmark-as-a-Service
draft-rorosz-nfvrg-vbaas-00

Abstract

Network Function Virtualization (NFV) promises the delivery of flexible chaining of Virtualized Network Functions (VNFs) with portability, elasticity, guaranteed performance, reliability among others. Service providers expect similar behavior from NFV as their current appliances based infrastructure, however, with the ease and flexibility of operation and cost effectiveness. Managing for predictable performance in virtualized environments is far from straightforward. Considering resource provisioning, 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. In order to support orchestration decisions, we argue that a framework for VNF benchmarking is need.

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 April 21, 2016.

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	2
2. Terms and Definitions	3
3. Motivation	4
4. Problem Statement and Challenges	5
5. Proposed Approach	6
6. Use Case: Unify Modelling	11
7. Related drafts and open source projects	16
8. IANA Considerations	18
9. Security Considerations	18
10. Acknowledgement	18
11. Informative References	18
Authors' Addresses	20

1. Introduction

Network Function Virtualization (NFV) pursues delivering Virtualized Network Functions (VNFs) in NFV Infrastructure (NFVI) where requirements of portability, performance, elasticity, among others, are demanded by carriers and network operators [ETS14a]. Particularly, in the short term, performance and elasticity are important factors to realize NFV concepts [BVLS15]. Associated to these goals VNFs need to be deployed with predictable performance, taking into account the variable processing overheads of virtualization and the underlying available NFVI resources.

In essence, NFV Management and Orchestration (MANO) plane manages infrastructure and network services resources [ETS14c]. This involves taking into account performance considerations for VNF-FGs allocation (e.g., fulfilling NFVI resources by orchestration demands). Furthermore, when attempting to manage service assurance of embedded VNF-FGs [PSPL15] MANO tasks may depend on analytics

measurements for scaling and migration purposes. As a consequence Network Function Virtualization Orchestrator (NFVO) needs a coherent understanding of performance vs. resource needs for VNFs specific to NFVI Points of Presence (PoPs).

In what follows we motivate the need for a VNF benchmarking, define related terms and introduce a framework proposal.

2. Terms and Definitions

The reader is assumed to be familiar with the terminology as defined in the European Telecommunications Standards Institute (ETSI) NFV document [ETSI14b]. Some of these terms, and others commonly used in this document, are defined below.

NFV: Network Function Virtualization - The principle of separating network functions from the hardware they run on by using virtual hardware abstraction.

NFVI PoP: NFV Infrastructure Point of Presence - Any combination of virtualized compute, storage and network resources.

NFVI: NFV Infrastructure - Collection of NFVI PoPs under one orchestrator.

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

NFVO: NFV Orchestrator - 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

VNF: Virtualized Network Function - a software-based network function.

VNFD: Virtualised Network Function Descriptor - configuration template that describes a VNF in terms of its deployment and operational behaviour, and is used in the process of VNF onboarding and managing the lifecycle of a VNF instance.

VNF-FG: Virtualized Network Function Forwarding Graph - an ordered list of VNFs creating a service chain.

MANO: Management and Orchestration - In the ETSI NFV framework [ETSI14c], this is the global entity responsible for management and orchestration of NFV lifecycle.

VBaaS: VNF Benchmark-as-a-Service - is a VNF Profile extraction service of a Virtualized Infrastructure Manager (VIM) overseeing an NFVI PoP.

VNF-BP: VNF Benchmarking Profile - the specification how to measure a VNF Profile. VNF-BP may be specific to a VNF or applicable to several VNF types. The specification includes structural and functional instructions, and variable parameters (metrics) at different abstractions (e.g., vCPU, memory, bandwidth, delay; session, transaction, tenants, etc.).

VNF Profile: is a mapping between virtualized resources (e.g., vCPU, memory) and VNF performance (e.g., bandwidth, delay between in/out or ports) at a given NFVI PoP. An orchestration function can use the VNF Profile to select host for a VNF and to allocate the desired resources to deliver a given (predictable) service quality.

3. Motivation

Let us take an example where a VNF Developer creates a new code base for a VNF. She is able to optimize her VNF for multiple execution environments and offers these as a set of different but equivalent implementations of VNF1 for Service Providers (SPs) (see Figure 1). Orchestration managers (e.g., NFVO) must know adequate infrastructure resources to allocate when deploying VNF1, specially if it belongs to a network service (VNF-FG) with a target SLA, e.g., min throughput or max latency. However, instances of VNF1 optimized for heterogeneous NFVI PoPs may need different resources/settings for the same behavior (performance). Orchestrator (NFVO) needs to take into account all these requirements to embed VNFs and allocate proper infrastructure resources in accordance with network service intents (SLAs).

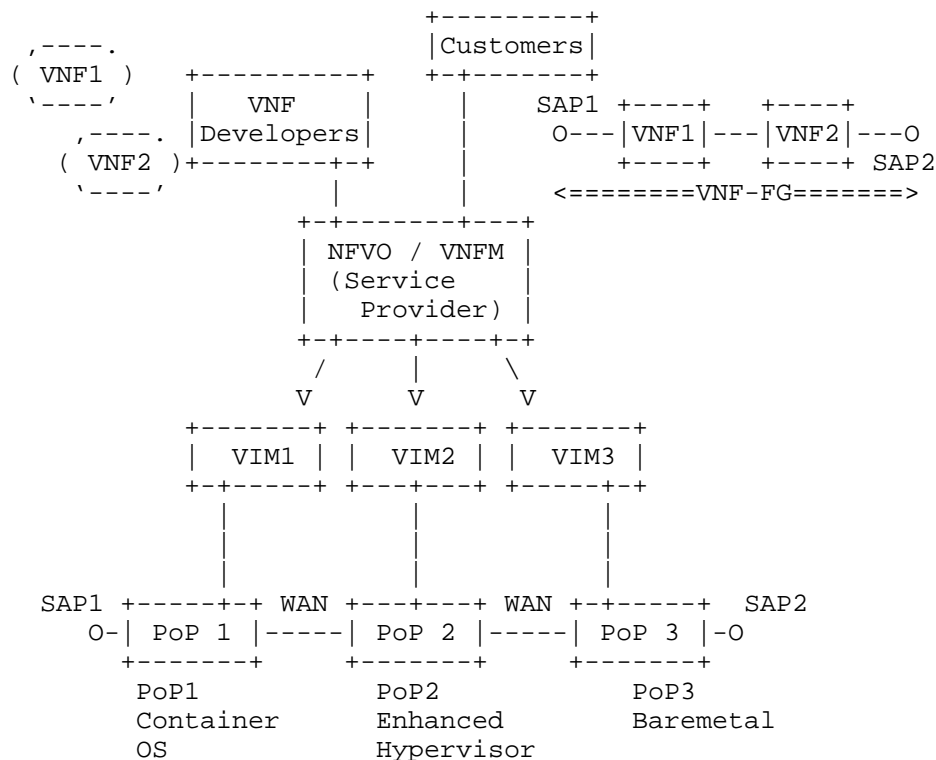


Figure 1: NFV MANO: Customers and VNF Developers

4. Problem Statement and Challenges

Previous section motivates NFVO needs of knowledge about correlations between VNFs performance and NFVI PoP resources.

Suppose that specifications, based on developers' definitions, exist in VNF Descriptors [ETS14d] describing performance associated to specific NFVI PoP resources. Such VNF, when deployed, may present oscillations in its performance because of changes in the underlying infrastructure (due to varying customers workloads, traffic engineering, scaling/migration, etc). Therefore, we believe we need descriptions on how to benchmark VNFs' resources and performance characteristics (e.g., VNF Profiles) on demand and in a flexible process supporting different NFVI PoPs, which could create service level specification associations.

Some of the associated challenges are:

Consistency: Will the VNF deployed at a NFVI PoP deliver the performance given in the VNF Profile for the NFVI PoP under different workloads? The performance of a VNF may not only depend on the resources allocated to the VNF but on the overall workload at the target NFVI PoP.

Stability: Benchmark measurements need to present consistent results when applied over different NFVI PoPs. How express benchmarking descriptions to fit in different virtualized environments? How to uniformly handle service/resource definitions and metrics?

Goodness: A given VIM / NFVI PoP may virtualize (i.e., hide) certain resource details; hence a benchmark evaluation might be executed in a different resource set of the NFVI PoP unlike the final production environment. How good benchmark results can correspond to actual workloads in virtualized environments?

VNF benchmarking service description: How to define the VNF benchmarking process at the abstraction of the corresponding component (e.g., VIM)? How to offer benchmarking as a service to be consumed by the different stakeholders?

VNF benchmarking versus monitoring: How can a VNF benchmarking process complement continuous monitoring of NFVI? When does a VNF benchmarking process surpass in performance and costs the execution of continuous monitoring process? And for which type of NFVI PoPs and/or VNFs is it necessary/sufficient?

5. Proposed Approach

Definition: VNF Benchmarking as a Service (VBaaS) -- a standalone service that can be hosted by orchestration managers (e.g., NFVO) to report a VNF Profile. VBaaS might take inputs consisting of VNF-FGs determining VNFs to be evaluated under specific NFVI targets. As output, VBaaS returns a Service Graph based on the VNF-Benchmark Profile (VNF-BP) containing different parameters used to evaluate all required candidates (PoPs) for the specified VNF. Such interactions are abstracted by the VBaaS API. A VBaaS Service Graph is deployed by orchestration managers (NFVO), VBaaS receives benchmarking results and builds VNF Profiles. VNF performance values or VNF resource values can be provided, and VBaaS can only complement the VNF-FG missing parts to build VNF-BPs, or report an existing up-to-date VNF Profile for pre-defined values.

In comparison with Figure 1, the VBaaS approach introduces two information bases, a VBaaS component and an API to interact with NFVO, as shown in Figure 2. On one hand, the Network Function

Information Base (NF-IB) holds the VNF Profiles and on the other hand the VBaaS-IB holds the VNF Benchmark Profiles (VNF-BPs).

Definition: VNF Profile -- is a mapping between virtualized resources of a certain NFVI PoP (e.g., virtual machine with vCPU, memory) and VNF performance (e.g., throughput, delay, packet loss between in/out or ports).

An orchestration function can use the VNF Profile to select hosts for VNFs in a VNF-FG and to allocate the necessary resources to deliver a given (predictable) service quality (SLA).

Figure 2 shows an example VNF1 and VNF2, which are implemented by a VNF Developer, who provides metrics for VBaaS build VNF-BPs. The VNF-BP is then used by VBaaS when NFVO needs to evaluate the available NFVI-PoPs for different resources configuration. In this case, NFVO instantiates the VNF-FG (VBaaS Service Graph) as defined in VNF1-BP; configures the variable parameters in the configuration template, e.g., CPU and memory values and handles the configuration file to the measurement controller (part of the VNF-FG definition) to execute the benchmarking. The measurement controller make the measurement agents execute the benchmark tests (the execution environment might also be monitored by the agents) and reports the results back to the VBaaS. As a result, resource and performance associations for the VNF is created for the evaluated the NFVI PoPs into the NF-IB. Figure 2 shows, as an example, that VNF1 needs 2CPU (cores) and 8GByte of memory to handle up 10Mbps input rate with transaction delay less than 200ms at NFVI PoP1.

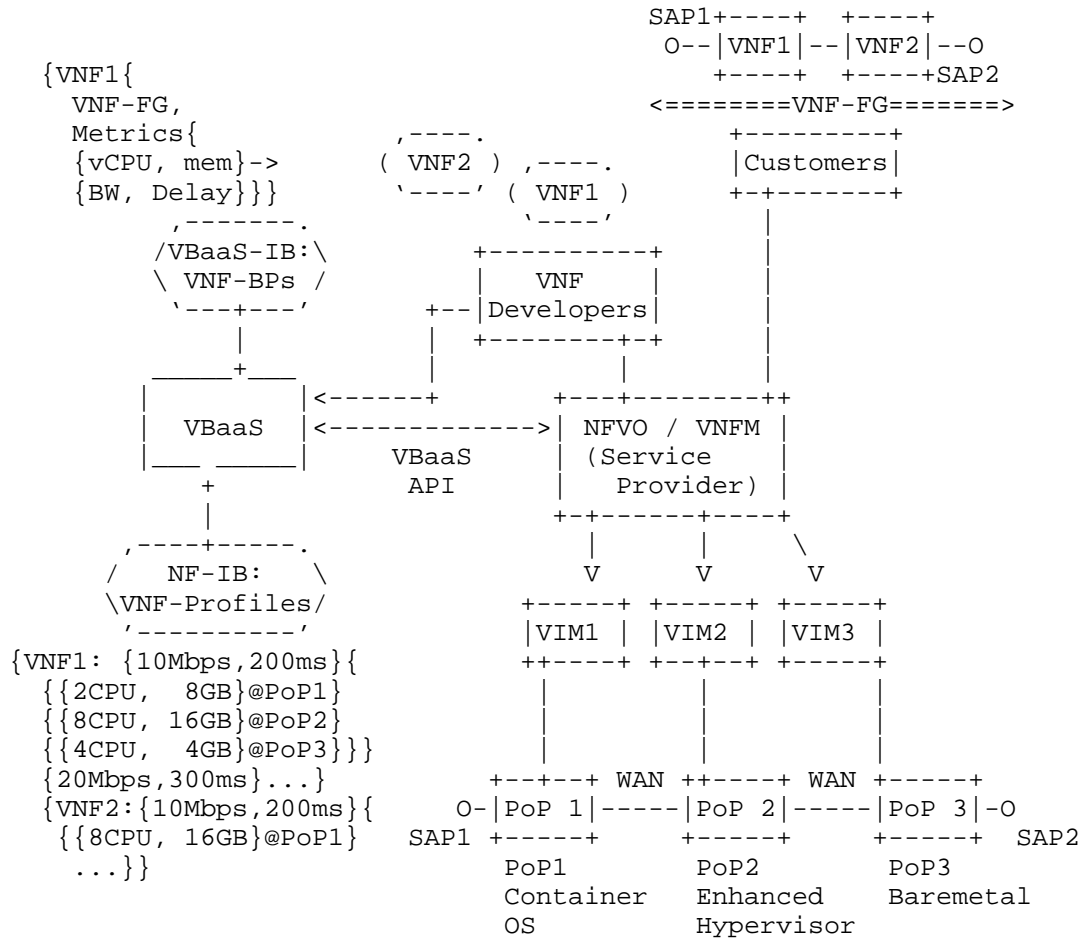


Figure 2: NFV MANO and VBaaS Approach

On the left side of the NFVO in Figure 2 is the resulting NF-IB by the application of VBaaS-IB profiles on the extraction of VNFs performance characteristics in different NFVI PoPs. Basically, a VNF profile corresponds to performance metrics in different environments. For example, VNF1 to offer 10Mbps bandwidth with 200ms latency needs (if deployed) in PoP1 2CPU (cores) and 8GB of memory, meanwhile in PoP2 requires 8CPU and 16GB. Similar results can be presented for VNF2, if catalogued as a VNF profile. In this scenario, VBaaS can be seen featuring VNFs in different PoPs in an on-demand composition of profiles, for example, to be used in provisioning VNF-FGs containing VNF1 or VNF2, attending PoPs 1, 2 or 3 resources consumption. Such profiles present performance metrics of VNFs in different NFVI PoPs

when, for example, NFVOs need to deploy particular VNF-FGs that contain elements already certified in NF-IB profiles.

Definition: VNF Benchmarking Profile (VNF-BPs) -- the specification of how to measure the VNF Profile for a given VNF. The specification includes structural (e.g., measurement components defined as a VNF-FG) and functional (e.g., measurement process definitions and configurable parameters) instructions, and variable parameters (metrics) at different abstractions (e.g., vCPU, memory, bandwidth, delay; session, transaction, tenants, etc.).

Note: a VNF-BP may be specific to a VNF or applicable to several VNF types.

Figure 3 presents details of VBaaS API regarding the interactions with NFVO and the construction of VBaaS Service Graphs based on VNF-BPs. As a VNF-FG (deployment request), NFVO demands VNF1 allocated to a specific PoP1 and the remaining virtualized view of the current resources for the VNF-FG deployment (e.g., interconnected PoPs). VBaaS (Manager Service) looks for VNF1-BP in VBaaS-IB, complements the requested VNF-FG with benchmark instances (described below) to perform evaluations and monitor resources in VNF1 and PoP1. NFVO receiving a VNF-FG reply (deployment order of a VBaaS Service Graph) instantiates it as a normal provisioning process of network services. Benchmark reports of the deployed VNF-FG are sent to VBaaS and it defines the construction of the VNF-Profile associated with that evaluation.

We believe in the existence of VBaaS instances (VNFs) which compose a VBaaS Service Graph to perform probing and monitoring activities on selected targets (VNF in a specific NFVI PoP). Those are defined as Manager, Monitor and Agent, described in details below.

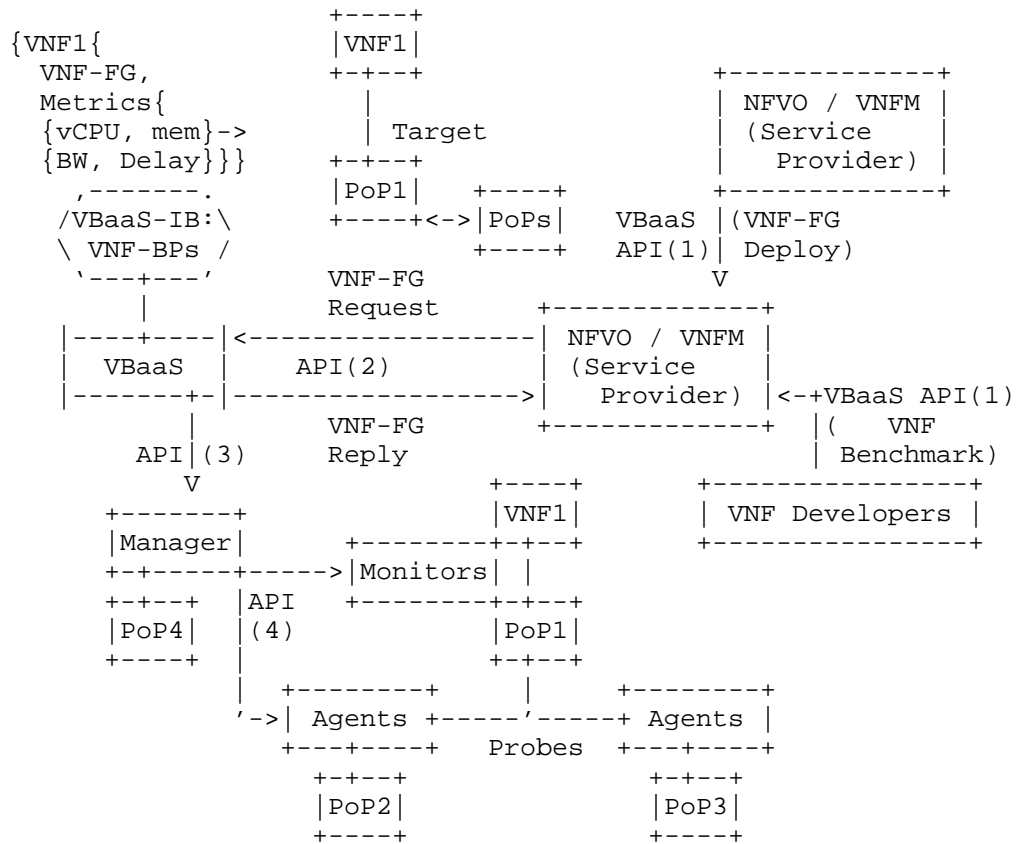


Figure 3: VBaaS APIs

Agent -- executes active probes using benchmark tools to collect network and system performance metrics by interacting with other agents. While a single Agent is capable of performing localized benchmarks (e.g., Stress tests on CPU, memory, I/O), the interaction among distributed agents enable the generation and collection of end-to-end metrics (e.g., packet loss, delay). Consider here the possibility of one end be the VNF itself where, for example, one-way packet delay is evaluated. For instance, it can be used for probe measurements of throughput and latency in a multi-point distributed topology. Agent's APIs are open and extensible (e.g., to plug-and-bench new tools and metrics) and receive configuration and run-time commands from the Manager for synchronization purposes (e.g., test duration/repetition) with other Agent and Monitor instances.

Monitor -- when possible it is placed inside the target VNF and NFVI PoP to perform active and passive monitoring and metric collection based on benchmarks evaluated according to Agents' workloads. For instance, to monitor CPU utilization of NFVI PoP considering packet length traffic variations sent by Agents. Different from the active approach of Agents that can be seen as generic benchmarking VNFs, monitors observe particular properties according to NFVI PoPs and VNFs capabilities. Monitors can plug-and-bench new tools/metrics and maintain an interface with Manager to synchronize its activities with Agents. Besides, they can be instantiated, if allowed, inside the target VNF (e.g., as a plug-in process in a virtualized environment) to check internal behaviours, alongside the VNF deployment environment, i.e., NFVI PoP, or in both places.

Manager -- is responsible for (i) the coordination and synchronization of activities between agents and monitors, (ii) collecting all benchmark raw results, and (iii) aggregating the inputs to construct a profile report that correlates different metrics as required by the VNF-BP. Therefore, it executes the main configuration, operation and management actions to deliver the results as specified by NFVI PoPs or VNF-BPs (e.g., instantiation of agents and monitors activities along-side tools and metrics configuration). Manager uses APIs to read and set configuration parameters for benchmarking instances such as metric parameters (e.g., bandwidth or packet loss probes) according to the VBaaS process. APIs are also used to receive benchmark instructions from VBaaS and send the reports of finished benchmark procedures.

The benchmarking process, however, can be offered as a service (see API (1) in right side of Figure 3). For example, a NFVO may be able to handle a VBaaS-IB and offer a VNF Benchmarking as a Service to its client, e.g., another NFVO. Similarly, a second NFVO may store benchmark measurements in its NF-IB so it can report it to multiple clients if it shares resources among other NFVOs. Alternatively, an NFVO may offer to its developers the VNF Benchmarking as a Service, to assess the performance of a VNF in the operator's environment. This may be part of a DevOps primitive (e.g., VNFs continuous integration).

6. Use Case: Unify Modelling

Unify looks for a generic description of compute and network resources as a yang model named virtualizer [Szab15]. It allows the aggregation of resources (e.g., nodes capabilities and NF instances) and their abstractions in a structure nicknamed Big Switch with Big Software (BiS-BiS) (joint software and network resource abstraction

with a joint control API), which allows recursive control plane structuring for multi-domain hierarchies. For example, supported network functions of a joint compute and network virtualization (a BiS-BiS node) can be described with their associated resources and performance mapping, e.g., {cpu, memory, storage} -> {bandwidth, delay} (see Figure 5 based on virtualizer xml example in [Szab15]).

In this example, the interface between NFVOs (Producer and Consumer of VBaaS API (1)), as shown in Figure 3, is described by the UNIFY virtualization model [Szab15]. Figure 4 shows an example netconf message exchange over the domain virtualization model. First the consumer reads in (1) and (2) the virtualization view, which includes the supported NFs reported in the capabilities section. Next in (3) the consumer requests a capability report of NF1 at a given virtualized node (UUID001), by defining the required delay and bandwidth performance parameters of the NF1 (see Figure 5. This is a dimensioning request according to [ETS14t], i.e., the VBaaS service must evaluate and provide cpu, memory and storage values at the given virtualized node to deliver the given network performance characteristics. The start of the VBaaS process is acknowledged by a (4) rpc-ok. Once the VBaaS process completes a notification is sent (5) to the consumer, which reads in (6) and (7) the updates to the NF1 capability report, i.e., the missing cpu, memory and storage parameters (see Figure 6).

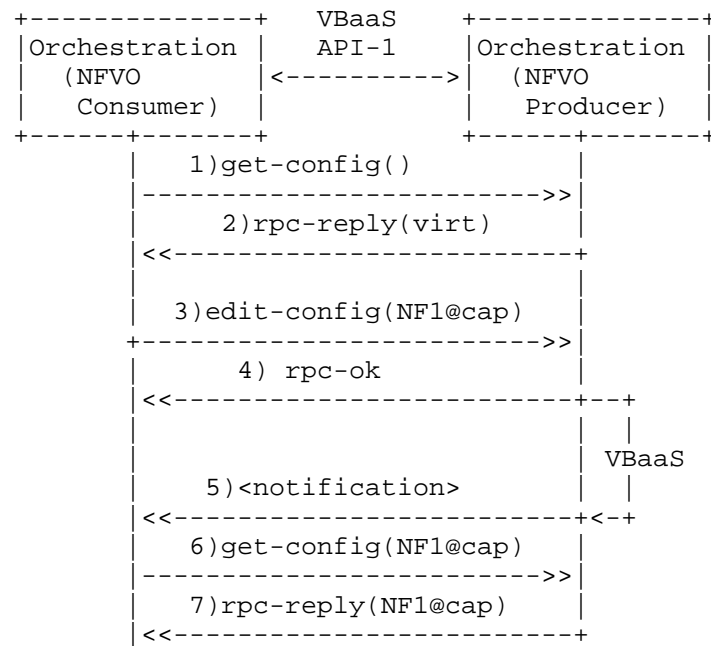


Figure 4: Sequence Diagram of VBaaS API (1) - Multi-Domain Interaction

```

<virtualizer xmlns="http://fp7-unify.eu/framework/virtualizer">
  <id>UUID001</id>
  <name>Single node simple VBaaS request</name>
  <nodes>
    <node>
      <id>UUID11</id>
      <name>single Bis-Bis node</name>
      <type>BisBis</type>
      <ports>
        ...
      </ports>
      <resources>
        <cpu>12</cpu>
        <mem>64 GB</mem>
        <storage>20 TB</storage>
      </resources>
      <capabilities>
        <supported_NFs>
          <node>
            <id>NF1</id>
            <name>vFirewall</name>
          </node>
        </supported_NFs>
      </capabilities>
    </node>
  </nodes>
</virtualizer>
  
```

```

    <type>Stateful virtual firewall C</type>
    <ports>
      <port>
        <id>1</id>
        <name>in</name>
        <port_type>port-abstract</port_type>
        <capability>...</capability>
      </port>
      <port>
        <id>2</id>
        <name>out</name>
        <port_type>port-abstract</port_type>
        <capability>...</capability>
      </port>
    </ports>
    <resources>
      <cpu> ? </cpu>
      <mem> ? </mem>
      <storage> ? </storage>
    </resources>
  <links>
    <link>
      <id>int0</id>
      <name>internal horizontal</name>
      <src>../../../../ports/port[id=1]</src>
      <dst>../../../../ports/port[id=2]</dst>
      <resources>
        <delay> 20 us </delay>
        <bandwidth> 10 GB </bandwidth>
      </resources>
    </link>
  </links>
</node>
</supported_NFs>
</capabilities>
</node>
</nodes>
</virtualizer>

```

Figure 5: VBaaS_Request(NF-FG-1)

```

<virtualizer xmlns="http://fp7-unify.eu/framework/virtualizer">
  <id>UUID001</id>
  <name>Single node simple VBaaS report</name>
  <nodes>
    <node>

```

```

<id>UUID11</id>
<name>single Bis-Bis node</name>
<type>BisBis</type>
<capabilities>
  <supported_NFs>
    <node>
      <id>NF1</id>
      <name>vFirewall</name>
      <type>Stateful virtual firewall C</type>
      <ports>
        <port>
          <id>1</id>
          <name>in</name>
          <port_type>port-abstract</port_type>
          <capability>...</capability>
        </port>
        <port>
          <id>2</id>
          <name>out</name>
          <port_type>port-abstract</port_type>
          <capability>...</capability>
        </port>
      </ports>
      <resources>
        <cpu>2</cpu>
        <mem>8 GB</mem>
        <storage>20 GB</storage>
      </resources>
    </node>
  </supported_NFs>
</capabilities>
</node>
</nodes>
</virtualizer>

```

Figure 6: VBaaS_Request(NF-FG-2)

Following the above example and the terminology of [ETS14t], the different tasks of performance verification, benchmarking and dimensioning can be mapped as follows:

Verification: Both {cpu, memory, storage} and {delay, bandwidth} parameters are provided and the VBaaS system verifies if the given association is correct or not. In the case of a failure the entry might be removed or updated with correct values.

Benchmarking: Where {cpu, memory, storage} parameters are provided and the VBaaS service fills-in the corresponding {delay, bandwidth} performance parameters. Note, such request might create more entries, like an entry with minimal delay and an entry with maximum bandwidth.

Dimensioning: Where {delay, bandwidth} performance parameters are provided and the VBaaS service fills-in the corresponding {cpu, memory, storage} resource parameters (as shown in the above example).

7. Related drafts and open source projects

Definetly, VBaaS intersects IETF Benchmarking Methodology Work Group (BMWG) goals. For example, in [Mor15], "Considerations for Benchmarking Virtual Network Functions and Their Infrastructure" presents relevant statements concerning, for example, % of utilization of NFVI PoPs resources; noisy behavior caused by VNFs operations and misbehavior in NFVI PoPs; long-term tests which represent service continuity guarantees for VNFs; and considerations on how shared resources dependencies may affect consistent benchmark results. All these NFV benchmarking aspects are relevant for VBaaS, as well other current [Tah15] and future recommendations from BMWG.

The main DevOps Principles for Software Defined Infrastructures (SDIs) (e.g., NFV and SDN), well explained in [Meir15], are associated with the main design concepts of VBaaS. By definition, DevOps principles look for deployment with repeatable and reliable processes (VNF-BPs), as well modular, e.g., VNF Profiles assisting orchestration decisions. Besides, it also stands for development and testing procedures to assist VNF Developers (VBaaS API) specify modular applications/components (VNF-BPs) against production-like systems. In addition, DevOps intends to define ways of monitor and validate operational quality of services before and after services deployment, e.g., respectively applying VNF-BPs in multiple NFVI PoPs and over instantiated VNFs. All these aspects illustrate how VBaaS can help in amplifying the cycle between VNFs continuous changes of

development and Operators desires for infrastructure stability and reliability, therefore, helping the consolidation of DevOps principles in SDIs.

In OPNFV (Open Platform for NFV) community, projects dedicated to VIM and NFVI tests are being developed, such as vSwitchPerf, Pharos, Functest, among others. Specifically, Yardstick [Yard15] is closely related with VBaaS, because it stands for a common definition of tests to verify the infrastructure compliance when running VNF applications. While VBaaS is still building its concepts, Yardstick already defined requirements, methodologies (including metrics for compute, network and storage domains), besides functional code. The specification of yaml files to define structures like scenarios, contexts and runners, as a Yardstick nomenclature to compose tests, represents associations with VNF-BPs concepts. As Yardstick future work intends to apply tests in compute and storage domains, and store tests results, it is on its way the possible intersection with VBaaS concepts, like VNF Profiles.

T-NOVA [TNOVA15] pursues the definition of a marketplace for VNFs (and turn them instanced as a service). Regarding the description of capabilities for the allocation of VNFs, T-NOVA intends to allow network services and functions, by a variety of developers, to be published and brokered/traded, allowing customers to browse and select services that best match their needs (negotiating SLAs and billing models). One of the main marketplace functions is the publication of resources and network function advertisements in order to compose and package NFV services. VBaaS can enhance VNFs capabilities publication and certification models of their performance when constructing a diverse set of VNF-Profiles. Consequently, VNF flavors can be defined by an enhanced set of VBaaS capabilities for smarter placements, e.g., exposing VBaaS interfaces for developers evolve VNF-Profiles.

Unify [Szab15] presents a recursive compute and network joint virtualization and programming view of generic infrastructures (e.g., data centers), considering NFV as an example. Big Switch with Big Software (BiS-BiS) represents the abstraction of such virtualization. The draft presents an Yang model to describe a Network Function Forwarding Graph (NF-FG) representing NFV resources. Complementary, as part of main Unify goals, problem statements and challenges about the unification of carrier and cloud networks [Csas15] show the need for a high level of programmability beyond policy and service descriptions. As presented, VBaaS can contribute for such abstractions since it detaches from monolithic views of infrastructure resources allowing the recursive definitions of VBaaS Service Graphs and, consequently, the decomposition of benchmark tasks in multiple domains. Besides, as stated by Unify measurements

and analytics challenges, VBaaS poses as a solution of tools for planning, testing, and reliability assurance of NFVIs.

8. IANA Considerations

This memo includes no request to IANA.

9. Security Considerations

TBD

10. Acknowledgement

The authors would like to thank the support of Ericsson Research, Brazil.

This work is partially supported by FP7 UNIFY, a research project partially funded by the European Community under the Seventh Framework Program (grant agreement no. 619609). 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. Informative References

- [BVLS15] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations", Communications Magazine, IEEE, vol. 53, no. 2, pp. 90-97, February 2015.
- [Csas15] R. Szabo, A. Csaszar, K. Pentikousis, M. Kind, D. Daino, Z. Qiang, H. Woesner, "Unifying Carrier and Cloud Networks: Problem Statement and Challenges", March 2015, <<https://tools.ietf.org/html/draft-unify-nfvrg-challenges-01>>.
- [ETS14a] ETSI, "Architectural Framework - ETSI GS NFV 002 V1.2.1", Dec 2014, <http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01-_60/gs_NFV002v010201p.pdf>.
- [ETS14b] ETSI, "Terminology for Main Concepts in NFV - ETSI GS NFV 003 V1.2.1", Dec 2014, <http://www.etsi.org/deliver/etsi_gs/NFV/001_099-/003/01.02.01_60/gs_NFV003v010201p.pdf>.

- [ETSI14c] ETSI, "Management and Orchestration - ETSI GS NFV-MAN 001 V1.1.1", Dec 2014, <http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf>.
- [ETSI14d] ETSI, "Virtual Network Functions Architecture - ETSI GS NFV-SWA 001 V1.1.1", Dec 2014, <http://www.etsi.org/deliver/etsi_gs/NFV-SWA/001_099/001/01.01.01_60/gs_NFV-SWA001v010101p.pdf>.
- [ETSI14t] ETSI, "NFV Pre-deployment Testing - ETSI GS NFV TST001 V0.0.13", Sept 2015, <http://docbox.etsi.org/ISG/NFV/Open/DRAFTS/TST001_-_Pre-deployment_Validation/NFV-TST001v0013.zip>.
- [Meir15] C. Meirosu, A. Manzalini, J. Kim, R. Steinert, S. Sharma, G. Marchetto, I. Papafili, K. Pentikousis, S. Wright, "DevOps for Software-Defined Telecom Infrastructures", July 2015, <<http://www.ietf.org/id/draft-unify-nfvrg-devops-02.txt>>.
- [Mor15] A. Morton, "Considerations for Benchmarking Virtual Network Functions and Their Infrastructure", February 2015, <<http://tools.ietf.org/html/draft-morton-bmwg-virtual-net-03>>.
- [PSPL15] E. Paik and M-K. Shin and S. Pack and S. Lee, "Resource Management in Service Chaining", March 2015, <<http://tools.ietf.org/html/draft-lee-nfvrg-resource-management-service-chain-01>>.
- [Szab15] R. Szabo, Z. Qiang, M. Kind, "Towards recursive virtualization and programming for network and cloud resources", July 2015, <<https://tools.ietf.org/html/draft-unify-nfvrg-recursive-programming-01>>.
- [Tah15] M. Tahhan, B. O'Mahony, A. Morton, "Benchmarking Virtual Switches in OPNFV", July 2015, <<https://tools.ietf.org/html/draft-vsperf-bmwg-vswitch-opnfv-00>>.
- [TNOVA15] T-NOVA, "T-NOVA Results", July 2015, <<http://www.t-nova.eu/results/>>.
- [Yard15] OPNFV, "Project: Yardstick - Infrastructure Verification", July 2015, <<https://wiki.opnfv.org/yardstick>>.

Authors' Addresses

Raphael Vicente Rosa
University of Campinas
Irinnyi Jozsef u. 4-20
Budapest 1117
Hungary

Email: raphaelvrosa@dca.fee.unicamp.br
URI: <http://www.intrig.dca.fee.unicamp.br/>

Christian Esteve Rothenberg
University of Campinas
Av. Albert Einstein, 400
Campinas 13083-852
Brasil

Email: chesteve@dca.fee.unicamp.br
URI: <http://www.dca.fee.unicamp.br/~chesteve/>

Robert Szabo
Ericsson Research, Hungary
Irinnyi Jozsef u. 4-20
Budapest 1117
Hungary

Email: robert.szabo@ericsson.com
URI: <http://www.ericsson.com/>

NFVRG
Internet-Draft
Intended status: Informational
Expires: May 5, 2016

M-K. Shin
ETRI
K. Nam
Friesty
S. Pack
KU
S. Lee
ETRI
R. Krishnan
Dell
T. Kim
LG U+

October 5, 2015

Verification of NFV Services : Problem Statement and Challenges
draft-shin-nfvrg-service-verification-04

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 (VNF) 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-enabled 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 May 2, 2016.

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. Real-time verification	9
5.5. Languages and their semantics	9
6. Gap analysis - open source projects	9
6.1. OPNFV	9
6.2. ODL	11
6.3. Summary	12
7. Security Considerations	12
8. Acknowledgements	13
9. References	13
Author's Address	15

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 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]. This 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 of VNFs can be composed together to reduce 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 be 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 locate 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 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 locate 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 with network congestion, the end-to-end performance (e.g., latency and throughput) in the VNF FG can be degraded. 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 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 inconsistency with each other when they have contradictory resource constraints on the shared NFVI resources. Examples of the policy conflicts are as follows:

<Example conflict case #1>

- o 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

A verification framework for NFV-based services 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 the entire network states 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 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 project 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 high-level language is newly proposed and discussed in open source project. 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 that 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 of the 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. Live traffic verification

It is 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, configurations and states of VNFs need to be precisely expressed using formal semantics. There are many languages and models, and it is impractical for the verification frameworks to support all of the existing languages and models. Languages and semantic models optimized to the verification framework need to be selected or newly developed.

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. Prediction

The Prediction project provides a data collection for failure prediction framework. The failure prediction framework diagnoses or verifies which entity is suspected to be progressing towards a failure and which VNFs might be affected due to the predicted anomaly.

While the Prediction project focuses only on fault prediction in NFVI compute, network, and storage resources, the document includes broader fault management and prediction issues such as faults in the NFV service deployment and operation.

6.1.3. Resource Scheduler

The Resource Scheduler project provides an enhanced scheduler for optimizing the performance of the VNFs. In particular, this project supports resource isolation. For example, when a VNF strictly requires low latency, strongly isolated compute resources can be allocated to the VNF.

The Resource Scheduler project only focuses on optimizing the performance of individual VNFs without considering the end-to-end performance (e.g., latency and throughput) in NFV services.

6.1.4. 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.

6.1.5 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.

6.2. 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.

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 Virtualization (NFV) Management and Orchestration," 2014.

[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.

[RAFT] <https://raftconsensus.github.io/>.

[ODL] "OpenDaylight SDN Controller, "<http://www.opendaylight.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

Tae-wan Kim
LG U+

Phone: +82 10 8080 6603
Email: dm24ks@lguplus.co.kr

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/lid-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.

* Holistic/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 ($\sim 10^4..10^6$). 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 hand 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 of 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) 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

NFVRG
Internet-Draft
Intended status: Informational
Expires: April 21, 2016

R. Szabo
Z. Qiang
Ericsson
M. Kind
Deutsche Telekom AG
October 19, 2015

Towards recursive virtualization and programming for network and cloud
resources
draft-unify-nfvrg-recursive-programming-02

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 April 21, 2016.

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	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	20
6. Examples	23
6.1. Infrastructure reports	24
6.2. Simple requests	29
7. IANA Considerations	31
8. Security Considerations	31
9. Acknowledgement	32
10. Informative References	32
Authors' Addresses	32

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]

- o 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;
- o 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

- o two black box DC set-ups, where the intra-DC network control is limited to some generic compute only control programming interface;
- o 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;
- o 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.

3.1.1.1. 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

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 and results in ping-pong traffic similar to the local and remote DC case discussed in [I-D.zu-nfvrg-elasticity-vnf].

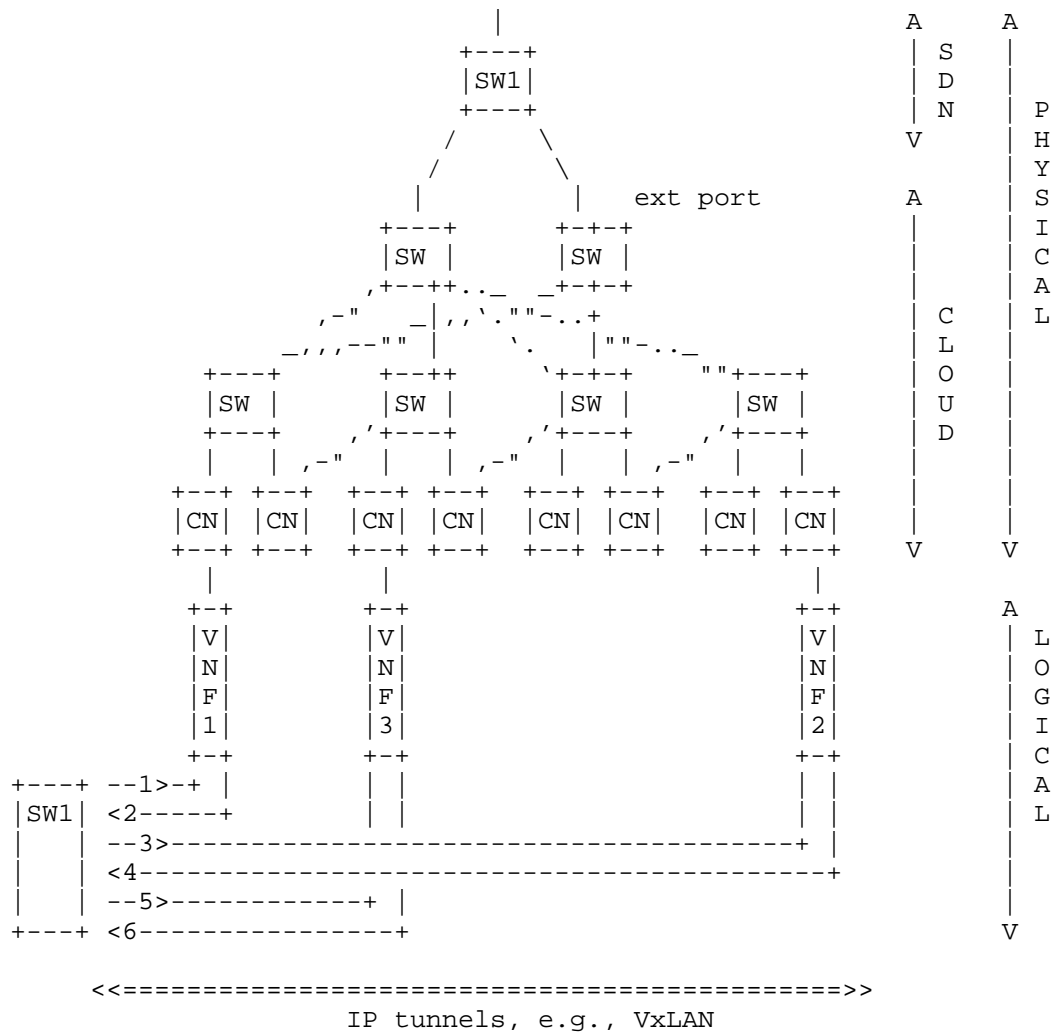


Figure 2: Black Box Data Center with ext Overlay

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

4. Recursive approach

We argued in [I-D.unify-nfvrg-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.

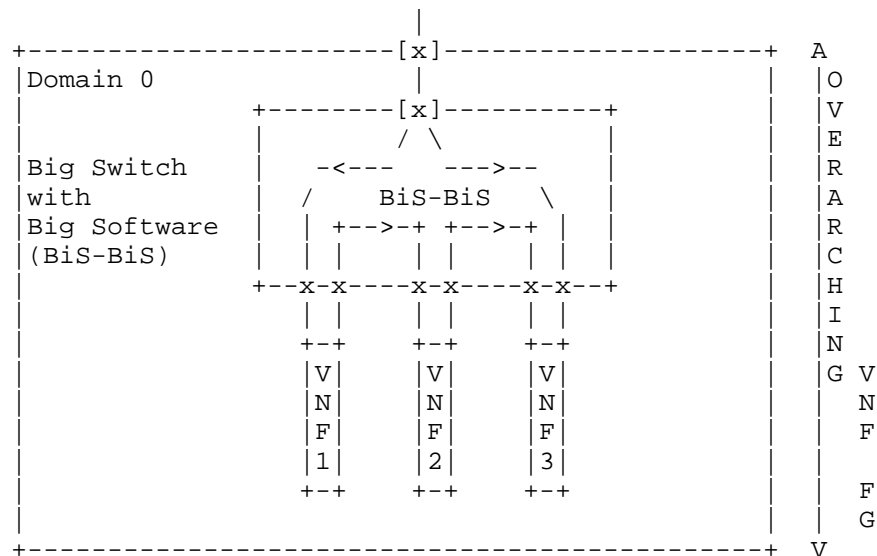


Figure 4: Recursive Domain Virtualization and Joint VNF FG programming: Overarching View

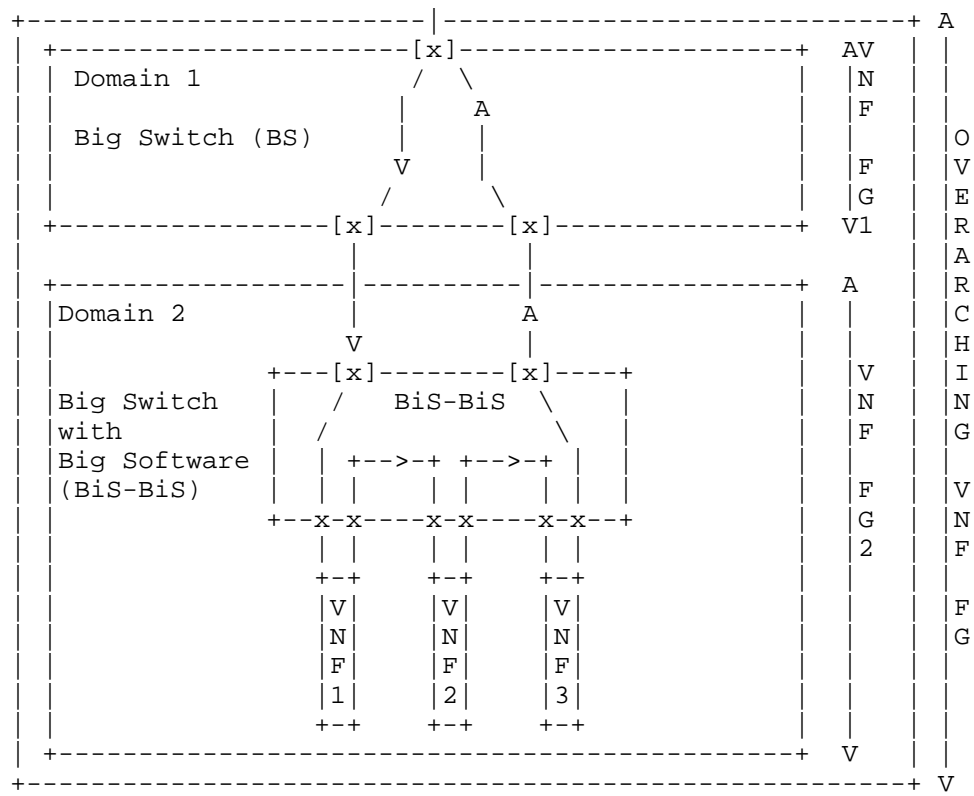


Figure 5: Recursive Domain Virtualization and Joint VNF FG programming: Domain Views

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).

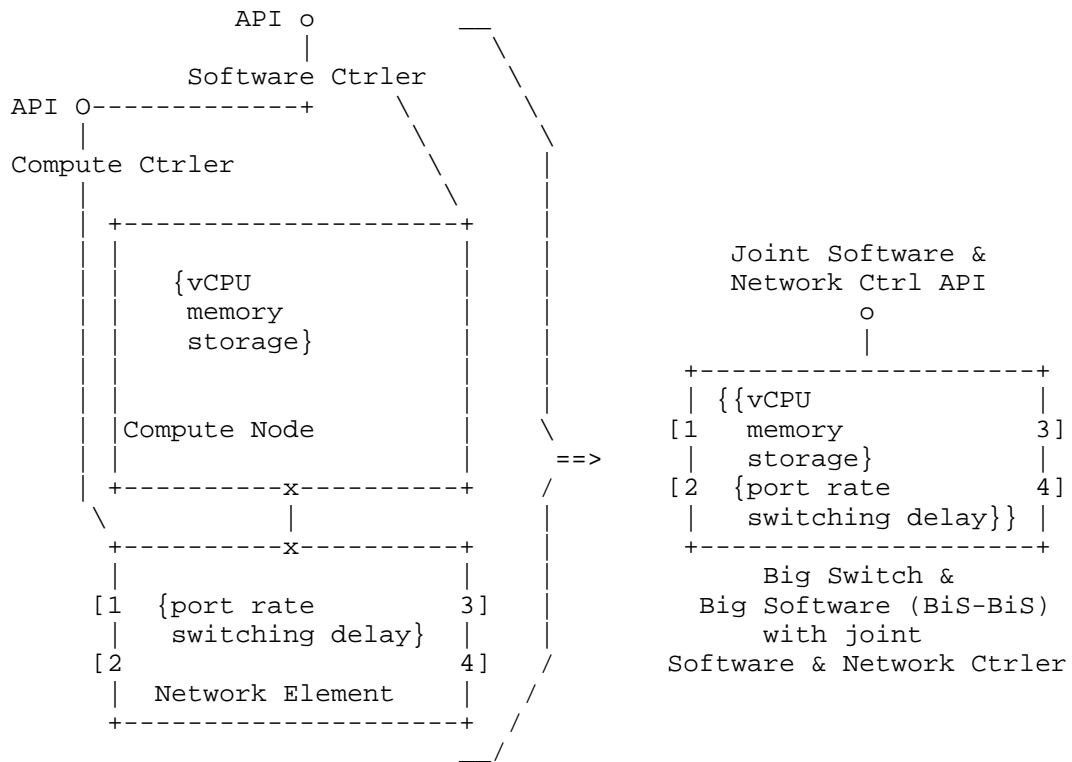


Figure 6: Big Switch with Big Software definition

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.

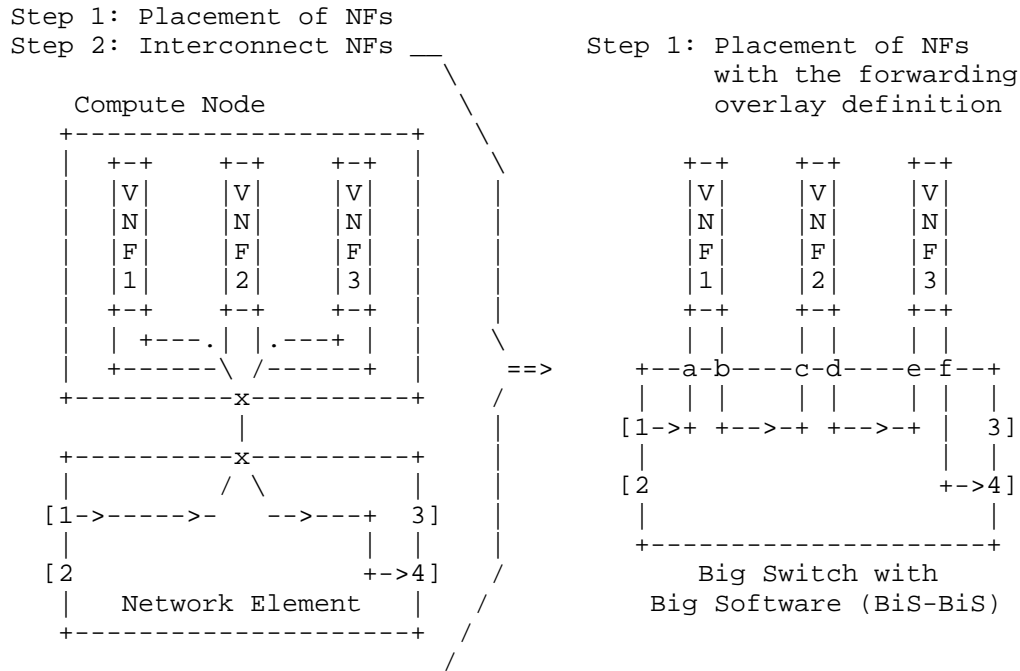


Figure 7: Big Switch with Big Software definition with a Network Function - Forwarding Graph (NF-FG)

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 port_data?  string
        +--rw links
          +--rw link* [src dst]

```

```

    |--rw id?          string
    |--rw name?       string
    |--rw src         port-ref
    |--rw dst         port-ref
    |--rw resources
        |--rw delay?   string
        |--rw bandwidth? string
+--rw resources
    |--rw cpu         string
    |--rw mem         string
    |--rw storage     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 port_data? string
        |--rw links
            |--rw link* [src dst]
                |--rw id?     string
                |--rw name?   string
                |--rw src     port-ref
                |--rw dst     port-ref
                |--rw resources
                    |--rw delay?   string
                    |--rw bandwidth? string
        |--rw resources
            |--rw cpu         string
            |--rw mem         string
            |--rw storage     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 port_data? string
        |--rw links

```

```

|         |         |         |--rw link* [src dst]
|         |         |         |--rw id?          string
|         |         |         |--rw name?         string
|         |         |         |--rw src           port-ref
|         |         |         |--rw dst           port-ref
|         |         |         |--rw resources
|         |         |         |         |--rw delay?      string
|         |         |         |         |--rw bandwidth?  string
|         |         |--rw resources
|         |         |         |--rw cpu           string
|         |         |         |--rw mem           string
|         |         |         |--rw storage        string
|--rw flowtable
|         |--rw flowentry* [port match action]
|         |--rw port           port-ref
|         |--rw match          string
|         |--rw action          string
|         |--rw resources
|         |         |--rw delay?      string
|         |         |--rw bandwidth?  string
|--rw links
|         |--rw link* [src dst]
|         |--rw id?          string
|         |--rw name?         string
|         |--rw src           port-ref
|         |--rw dst           port-ref
|         |--rw resources
|         |         |--rw delay?      string
|         |         |--rw bandwidth?  string

```

Figure 8: Virtualizer's YANG data model: tree view

4.1.1.2. YANG Module

<CODE BEGINS> file "virtualizer.yang"

```

module virtualizer {
  namespace "http://fp7-unify.eu/framework/virtualizer";
  prefix virt;
  organization "EU-FP7-UNIFY";
  contact "Robert Szabo <robert.szabo@ericsson.com>";
  description "data model for joint software and network
  virtualization and resource control";

  revision 2015-06-27 {
    reference "Initial version";
  }
}

```

```
// REUSABLE GROUPS
grouping id-name {
  description "used for key (id) and naming";
  leaf id {
    type string;
    description "For unique key id";}
  leaf name {
    type string;
    description "Descriptive name";}
}

grouping node-type {
  description "For node type definition";
  leaf type{
    type string;
    mandatory true;
    description "to identify nodes (infrastructure or NFs)";
  }
}

// PORTS
typedef port-ref {
  type string;
  description "path to a port; can refer to ports at multiple
  levels in the hierarchy";
}

grouping port {
  description "Port definition: used for infrastructure and NF
  ports";
  uses id-name;
  leaf port_type {
    type string;
    mandatory true;
    description "Port type identification: abstract is for
    technology independent ports and SAPs for technology specific
    ports";}
  leaf port_data{
    type string;
    description "Opaque data for port specific types";
  }
}

grouping ports {
  description "Collection of ports";
  container ports {
    description "see above";
    list port{
```

```
        key "id";
        uses port;
        description "see above";
    }
}
// FORWARDING BEHAVIOR
grouping flowentry {
    leaf port {
        type port-ref;
        mandatory true;
        description "path to the port";
    }
    leaf match {
        type string;
        mandatory true;
        description "matching rule";
    }
    leaf action {
        type string;
        mandatory true;
        description "forwarding action";
    }
    container resources{
        uses link-resource;
        description "network resources assigned to forwarding entry";
    }
    description "SDN forwarding entry";
}

grouping flowtable {
    container flowtable {
        description "Collection of flowentries";
        list flowentry {
            key "port match action";
            description "Index list of flowentries";
            uses flowentry;
        }
    }
    description "See container description";
}

// LINKS
grouping link-resource {
    description "Core networking characteristics / resources
(bandwidth, delay)";
    leaf delay {
        type string;
    }
}
```

```
        description "Delay value with unit; e.g. 5ms";
    }
    leaf bandwidth {
        type string;
        description "Bandwithd value with unit; e.g. 10Mbps";
    }
}

grouping link {
    description "Link between src and dst ports with attributes";
    uses id-name;
    leaf src {
        type port-ref;
        description "relative path to the source port";
    }
    leaf dst {
        type port-ref;
        description "relative path to the destination port";
    }
    container resources{
        uses link-resource;
        description "Link resources (attributes)";
    }
}

grouping links {
    description "Collection of links in a virtualizer or a node";
    container links {
        description "See above";
        list link {
            key "src dst";
            description "Indexed list of links";
            uses link;
        }
    }
}

// CAPABILITIES
grouping capabilities {
    description "For capability reporting: currently supported NF
types";
    container supported_NFs { // supported NFs are enumerated
        description "Collecction of nodes as supported NFs";
        list node{
            key "id";
            description "see above";
            uses node;
        }
    }
}
```

```
    }
    // TODO: add other capabilities
}

// NODE

grouping software-resource {
  description "Core software resources";
  leaf cpu {
    type string;
    mandatory true;
    description "In virtual CPU (vCPU) units";
  }
  leaf mem {
    type string;
    mandatory true;
    description "Memory with units, e.g., 1Gbyte";
  }
  leaf storage {
    type string;
    mandatory true;
    description "Storage with units, e.g., 10Gbyte";
  }
}

grouping node {
  description "Any node: infrastructure or NFs";
  uses id-name;
  uses node-type;
  uses ports;
  uses links;
  container resources{
    description "Software resources offer/request of the node";
    uses software-resource;
  }
}

grouping infra-node {
  description "Infrastructure nodes wich can contain other nodes
as NFs";
  uses node;
  container NF_instances {
    description "Hosted NFs";
    list node{
      key "id";
      uses node;
      description "see above";
    }
  }
}
```

```

    }
    container capabilities {
        description "Supported NFs as capability reports";
        uses capabilities;
    }
    uses flowtable;
}

//===== Virtualizer =====

container virtualizer {
    description "Definition of a virtualizer instance";
    uses id-name;

    container nodes{
        description "infra nodes, which embeds NFs and report
        capabilities";
        list node{
            key "id";
            uses infra-node;
            description "see above";
        }
    }
    uses links;
}
}
<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:

- o the orchestration of NFVI resources across multiple VIMs, fulfilling the Resource Orchestration functions;
- o 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

Similarly, a VIM is split into two functions:

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

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

The functional split is shown in Figure 13.

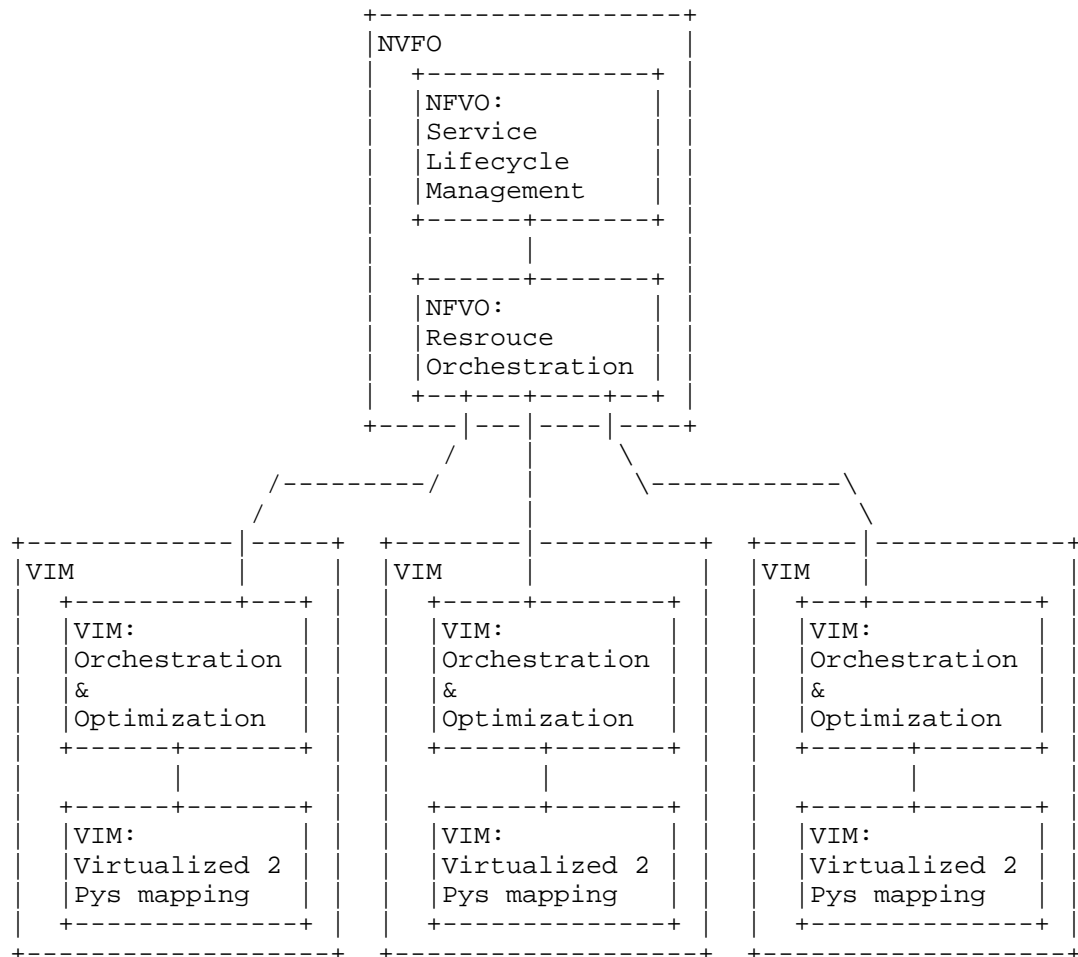


Figure 10: Functional decomposition of the NFVO and the VIM according to the ETSI MANO

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.

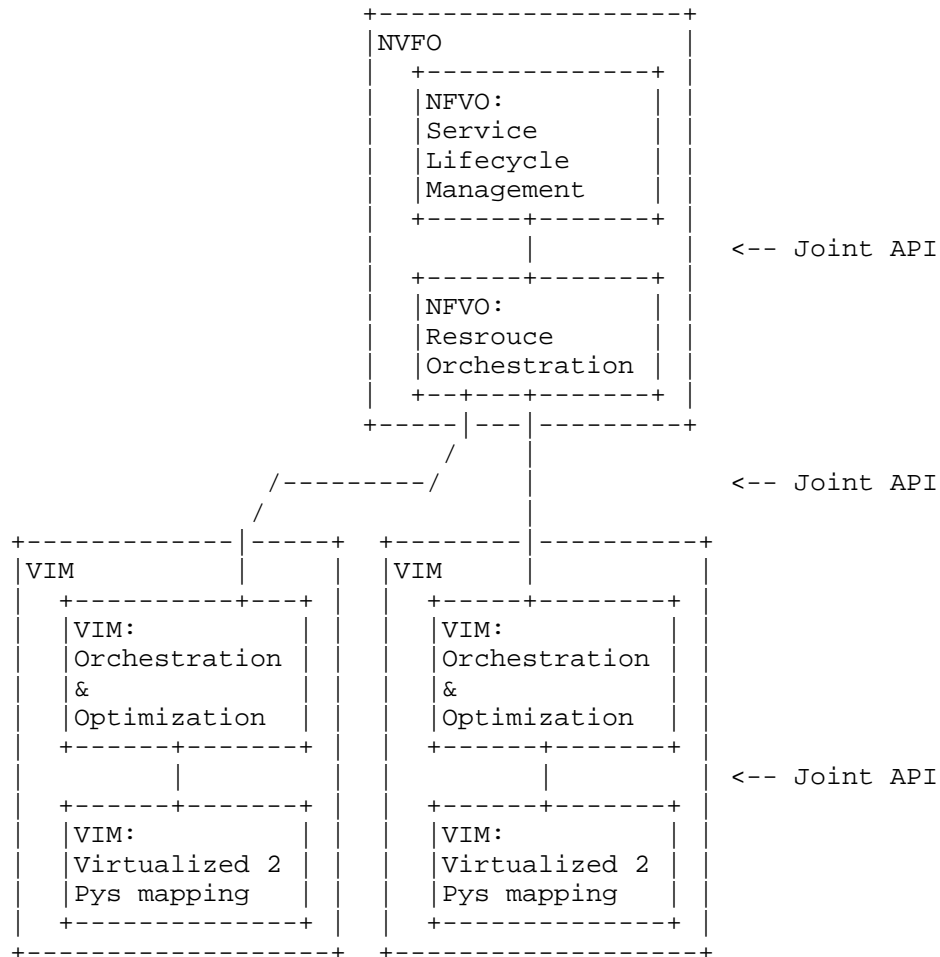


Figure 11: Functional decomposition of the NFVO and the VIM with the Joint Software and Network control API

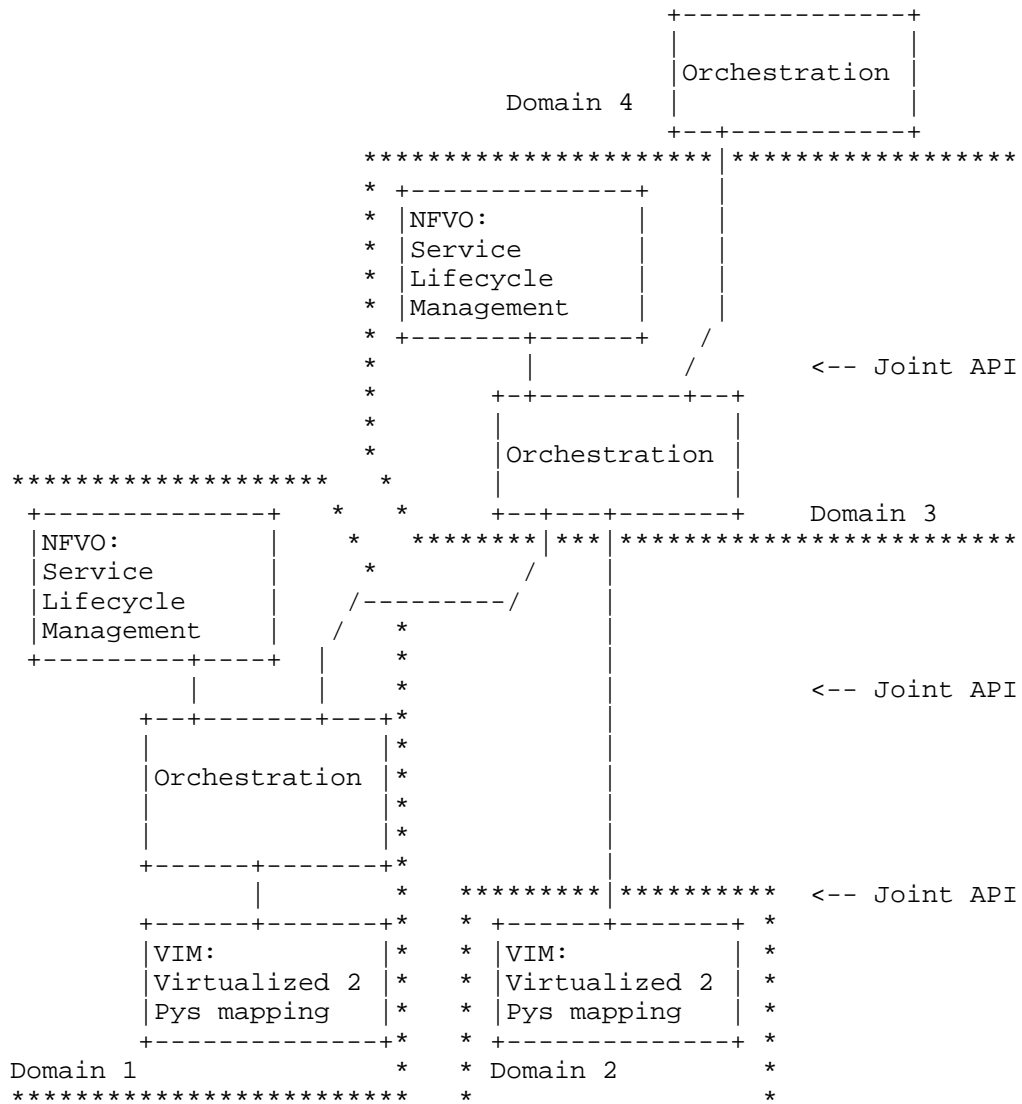


Figure 12: Joint Software and Network Control API: Recurring Flexible Architecture

6. Examples


```

<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 14: Single node infrastructure report example: xml view

Figure 15 and Figure 16 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.

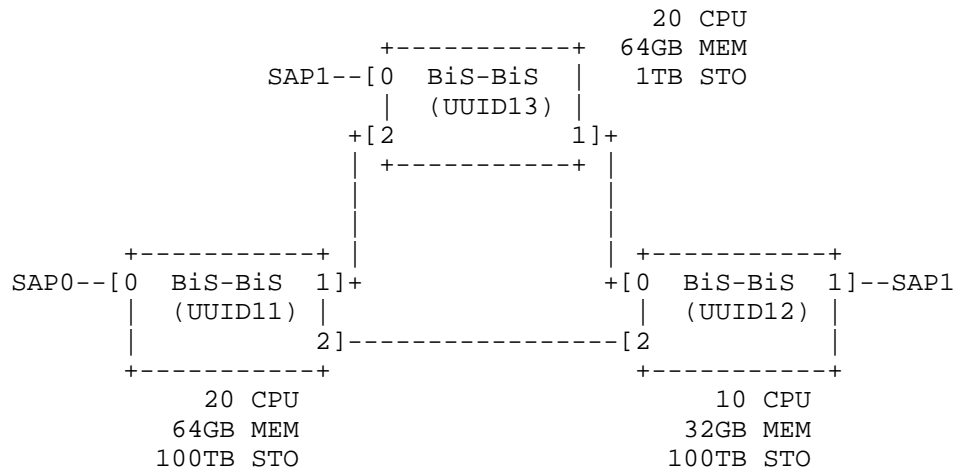


Figure 15: 3-node infrastructure report example: Virtualization view

```

<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>

```



```

    </link>
  </links>
</virtualizer>

```

Figure 16: 3-node infrastructure report example: xml view

6.2. Simple requests

Figure 17 and Figure 18 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.

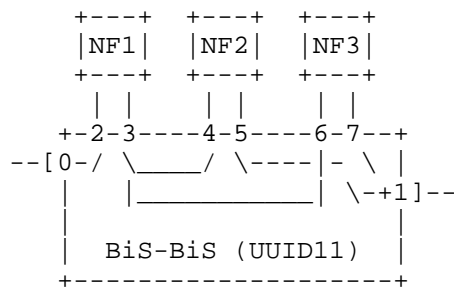


Figure 17: Simple request of 3 NFs on a single BiS-BiS: Virtualization view

```

<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>
    </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>

```

rpcre

Figure 18: Simple request of 3 NFs on a single BiS-BiS: xml view

7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

TBD

9. 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.

10. Informative References

[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](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](http://docbox.etsi.org/ISG/NFV/Open/Latest_Drafts/NFV-MAN001v061-%20management%20and%20orchestration.pdf)>.

[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-03 (work in progress), July 2015.

[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-02 (work in progress), July 2015.

[I-D.zu-nfvrg-elasticity-vnf]

Qiang, Z. and R. Szabo, "Elasticity VNF", draft-zu-nfvrg-elasticity-vnf-01 (work in progress), March 2015.

Authors' Addresses

Robert Szabo
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