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.

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

Table of Contents

1. Introduction

To a large degree there is agreement in the research community that rigid network control limits the flexibility of service creation. In [I-D.unify-nfvrg-challenges]

- we analyzed different compute domain abstractions to argue that joint compute and network virtualization and programming is needed for efficient combination of these resource domains;

- we described challenges associated with the combined handling of compute and network resources for a unified production environment.

Our goal here is to analyze different approaches to instantiate a service graph with transparent network functions into a commodity Data Center (DC). More specifically, we analyze

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

- a white box DC set-up, where the intra-DC network control is exposed directly to for a DC external control to coordinate forwarding configurations;

- a recursive approach, which illustrates potential benefits of a joint compute and network virtualization and control.

The discussion of the problems and the proposed solution is generic for any data center use case; however, we use NFV as an example.

2. Terms and Definitions

We use the terms compute and "compute and storage" interchangeably throughout the document. Moreover, we use the following definitions, as established in [ETSI-NFV-Arch]:

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

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

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

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

Further, we make use of the following terms:

NF: a network function, either software-based (VNF) or appliance-based.

SW: a (routing/switching) network element with a programmable control plane interface.

DC: a data center is an interconnection of Compute Nodes (see below) with a data center controller, which offers programmatic resource control interface to its clients.

CN: a server, which is controlled by a DC control plane and provides execution environment for virtual machine (VM) images such as VNFs.

3. Use Cases

Service Function Chaining (SFC) looks into the problem how to deliver end-to-end services through the chain of network functions (NFs). Many of such NFs are envisioned to be transparent to the client, i.e., they intercept the client connection for adding value to the services without the knowledge of the client. However, deploying network function chains in DCs with Virtualized Network Functions (VNFs) are far from trivial [I-D.ietf-sfc-dc-use-cases]. For example, different exposures of the internals of the DC will imply different dynamisms in operations, different orchestration complexities and may yield for different business cases with regards to infrastructure sharing.

We investigate different scenarios with a simple NF forwarding graph of three VNFs (o->VNF1->VNF2->VNF3->o), where all VNFs are deployed within the same DC. We assume that the DC is a multi-tier leaf and spine (CLOS) and that all VNFs of the forwarding graph are bump-in-the-wire NFs, i.e., the client cannot explicitly access them.

3.1. Black Box DC

In Black Bock DC set-ups, we assume that the compute domain is an autonomous domain with legacy (e.g., OpenStack) orchestration APIs. Due to the lack of direct forwarding control within the DC, no native L2 forwarding can be used to insert VNFs running in the DC into the forwarding graph. Instead, explicit tunnels (e.g., VxLAN) must be used, which need termination support within the deployed VNFs. Therefore, VNFs must be aware of the previous and the next hops of the forwarding graph to receive and forward packets accordingly.

3.1.1. Black Box DC with L3 tunnels

Figure 1 illustrates a set-up where an external VxLAN termination point in the SDN domain is used to forward packets to the first NF (VNF1) of the chain within the DC. VNF1, in turn, is configured to forward packets to the next SF (VNF2) in the chain and so forth with VNF2 and VNF3.

In this set-up VNFs must be capable of handling L3 tunnels (e.g., VxLAN) and must act as forwarders themselves. Additionally, an operational L3 underlay must be present so that VNFs can address each other.

Furthermore, VNFs holding chain forwarding information could be untrusted user plane functions from 3rd party developers. Enforcement of proper forwarding is problematic.

Additionally, compute only orchestration might result in sub-optimal allocation of the VNFs with regards to the forwarding overlay, for example, see back-forth use of a core switch in Figure 1.

In [I-D.unify-nfvrg-challenges] we also pointed out that within a single Compute Node (CN) similar VNF placement and overlay optimization problem may reappear in the context of network interface cards and CPU cores.

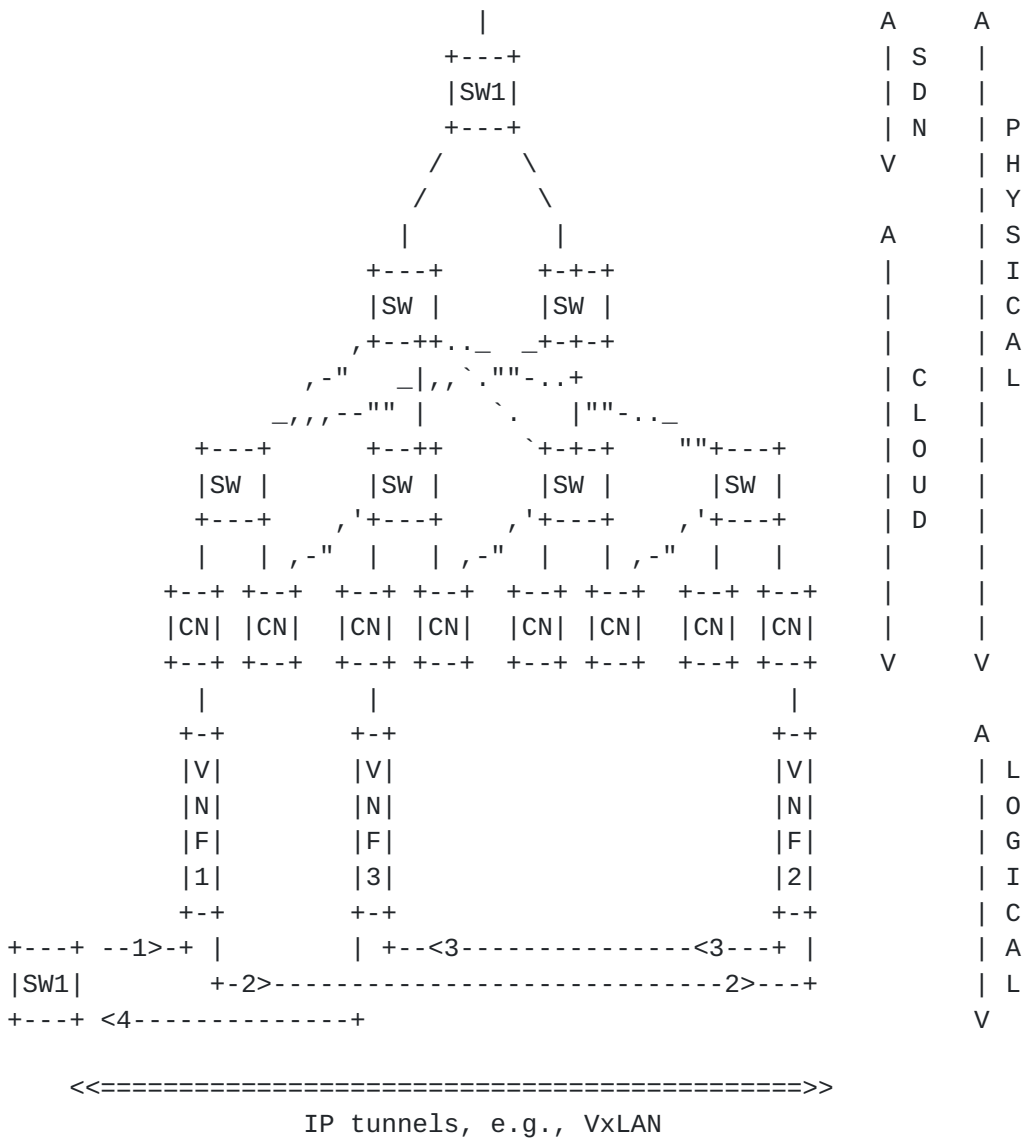

Figure 1: Black Box Data Center with VNF Overlay

3.1.2. Black Box DC with external steering

Figure 2 illustrates a set-up where an external VxLAN termination point in the SDN domain is used to forward packets among all the SFs (VNF1-VNF3) of the chain within the DC. VNFs in the DC need to be configured to receive and send packets between only the SDN endpoint, hence are not aware of the next hop VNF address. Shall any VNFs need to be relocated, e.g., due to scale in/out as described in [I-D.zu-nfvrg-elasticity-vnf], the forwarding overlay can be transparently re-configured at the SDN domain.

Note however, that traffic between the DC internal SFs (VNF1, VNF2, VNF3) need to exit and re-enter the DC through the external SDN switch. This, certainly, is sub-optimal an results in ping-pong traffic similar to the local and remote DC case discussed in [I-D.zu-nfvrg-elasticity-vnf].

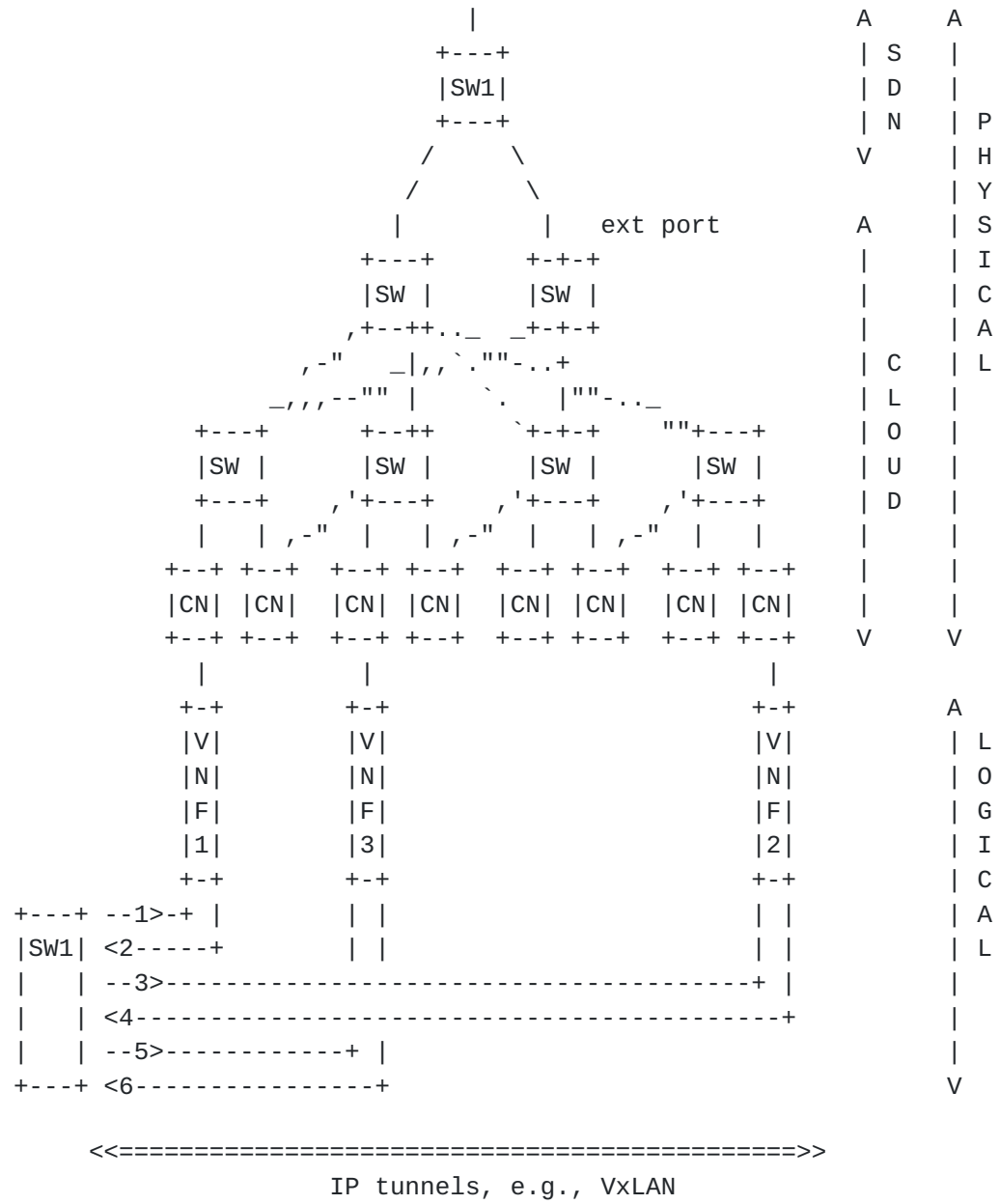

Figure 2: Black Box Data Center with ext Overlay

3.2. White Box DC

Figure 3 illustrates a set-up where the internal network of the DC is exposed in full details through an SDN Controller for steering control. We assume that native L2 forwarding can be applied all through the DC until the VNFs' port, hence IP tunneling and tunnel termination at the VNFs are not needed. Therefore, VNFs need not be forwarding graph aware but transparently receive and forward packets. However, the implications are that the network control of the DC must be handed over to an external forwarding controller (see that the SDN domain and the DC domain overlaps in Figure 3). This most probably prohibits clear operational separation or separate ownerships of the two domains.

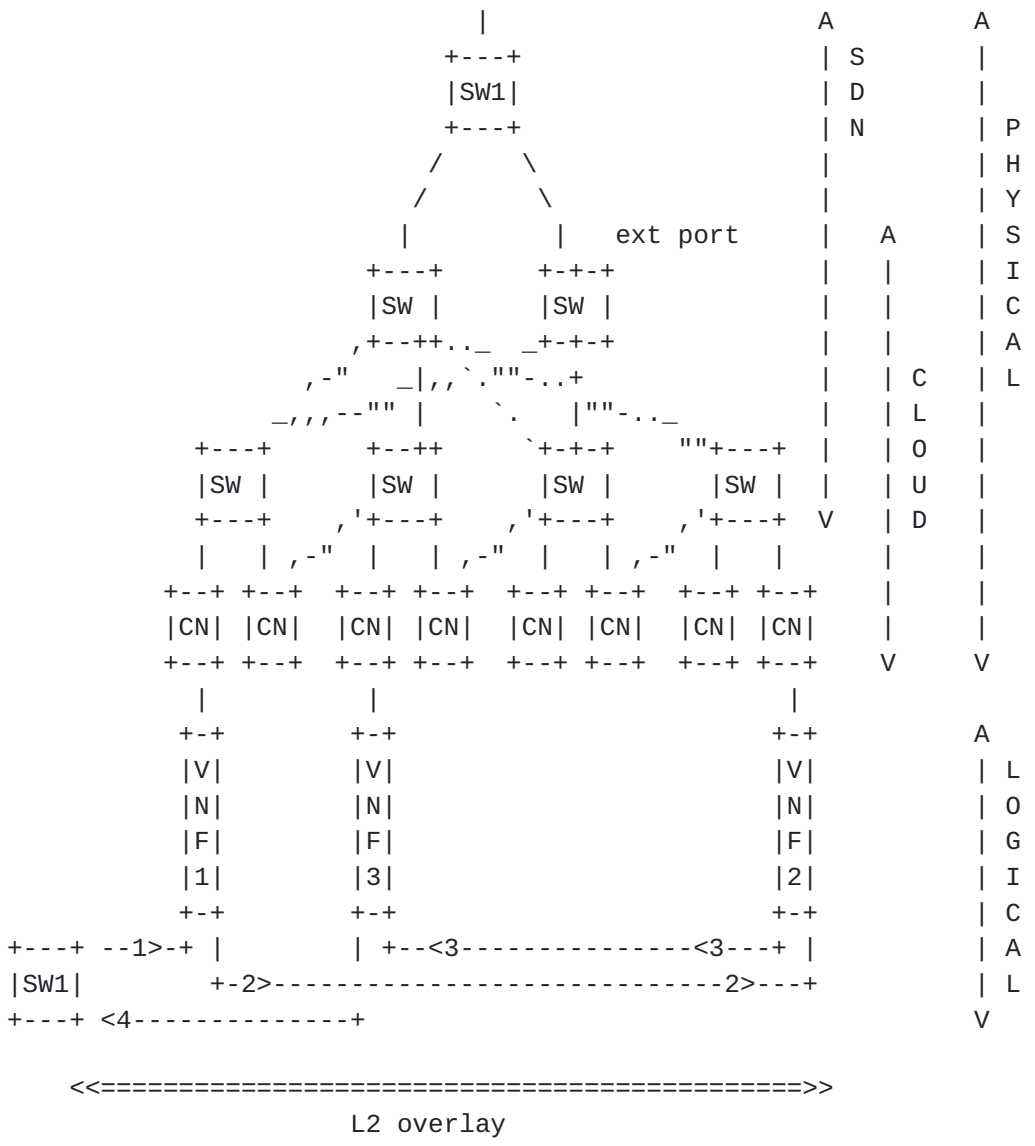

Figure 3: White Box Data Center with L2 Overlay

3.3. Conclusions

We have shown that the different solutions imply different operation and management actions. From network operations point of view, it is not desirable to run and manage similar functions several times (L3 blackbox DC case) - especially if the networking overlay can be easily managed upfront by using a programmatic interface, like with the external steering in black and whitebox DC scenarios.

4. Recursive approach

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

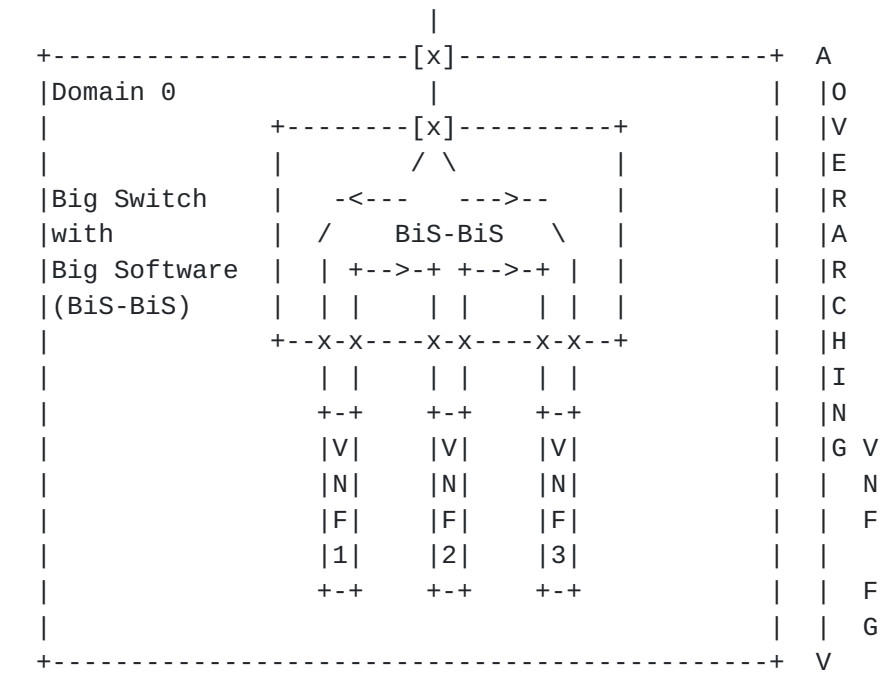

Figure 4: Recursive Domain Virtualization and Joint VNF FG programming: Overarching View

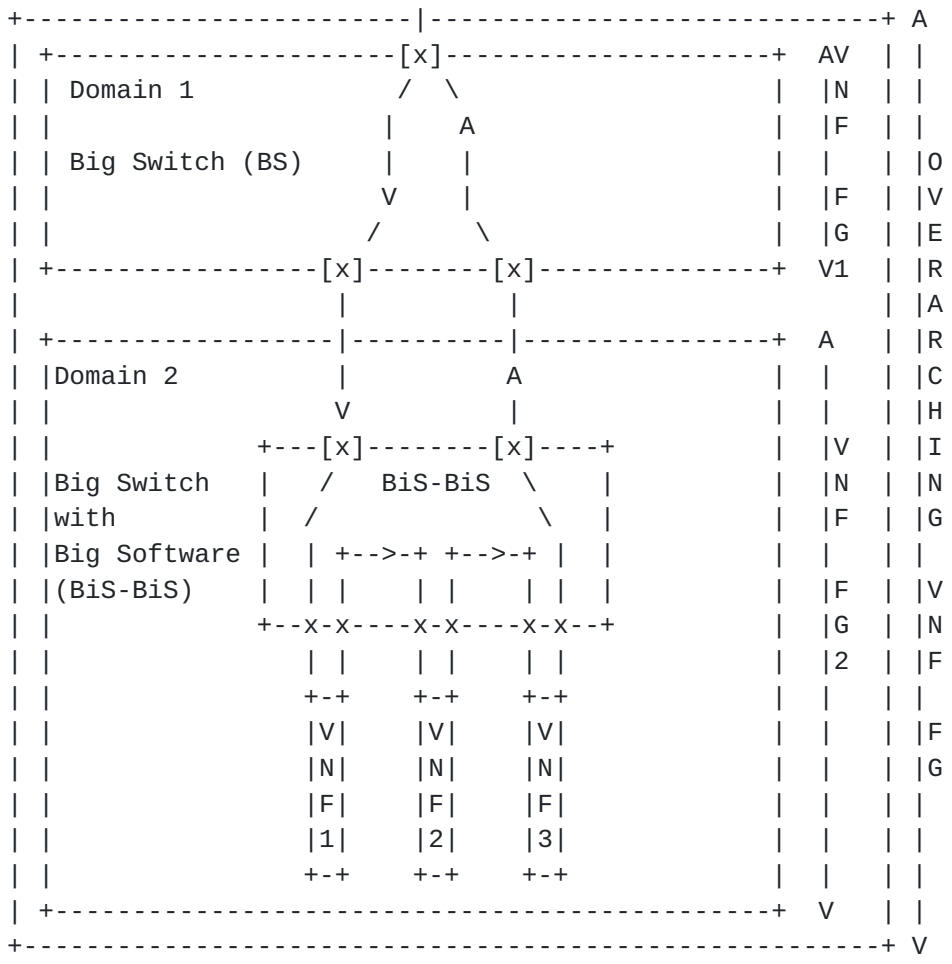

Figure 5: Recursive Domain Virtualization and Joint VNF FG programming: Domain Views

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).

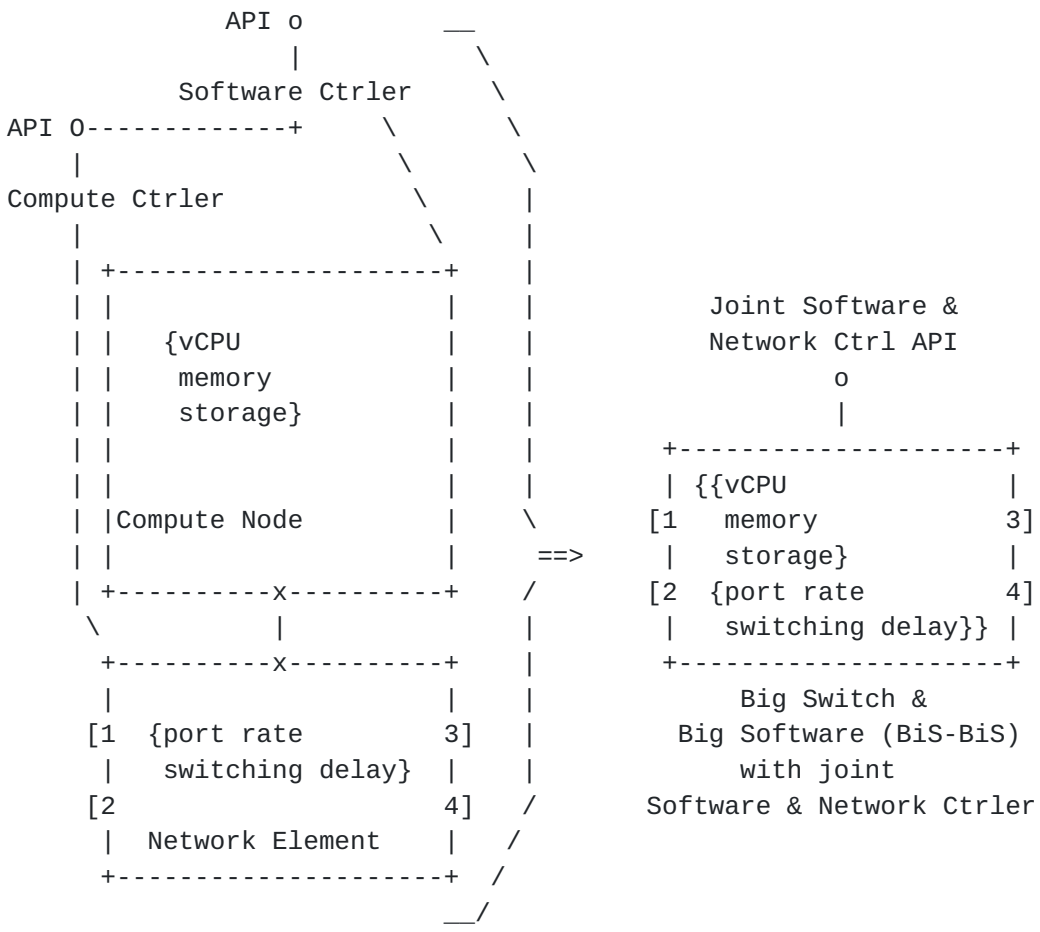

Figure 6: Big Switch with Big Software definition

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.

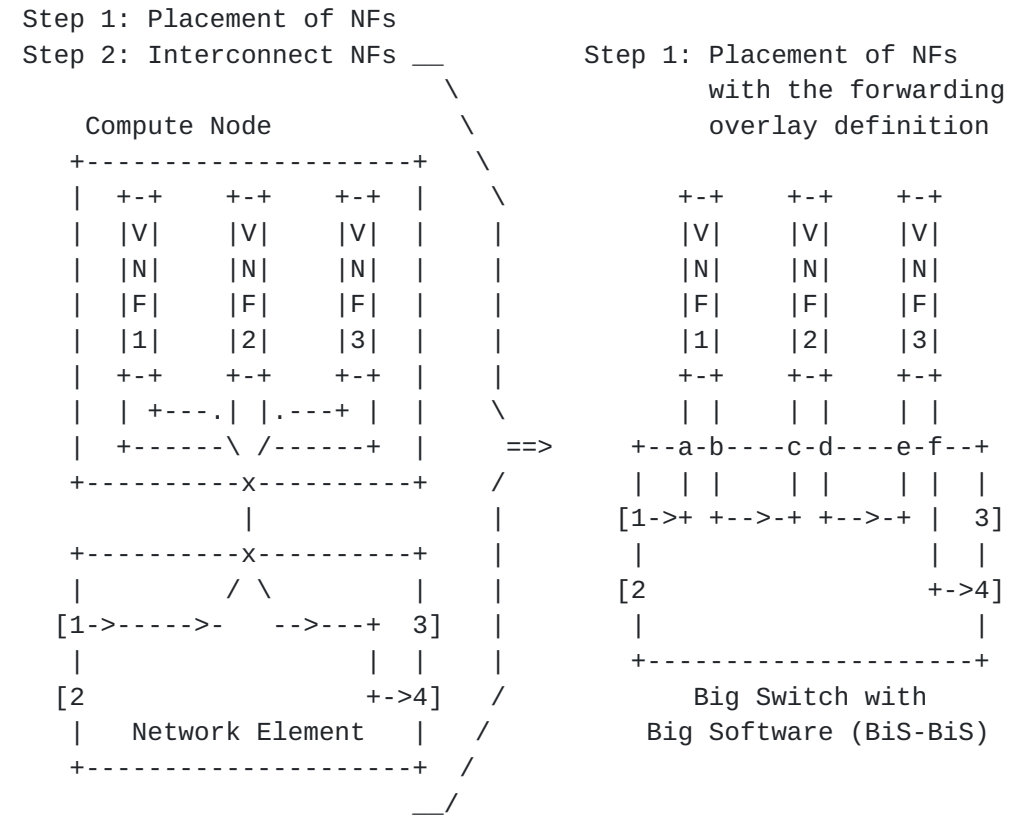

Figure 7: Big Switch with Big Software definition with a Network Function - Forwarding Graph (NF-FG)

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 defintion";
        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:

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

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

Similarly, a VIM is split into two functions:

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

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

The functional split is shown in Figure 13.

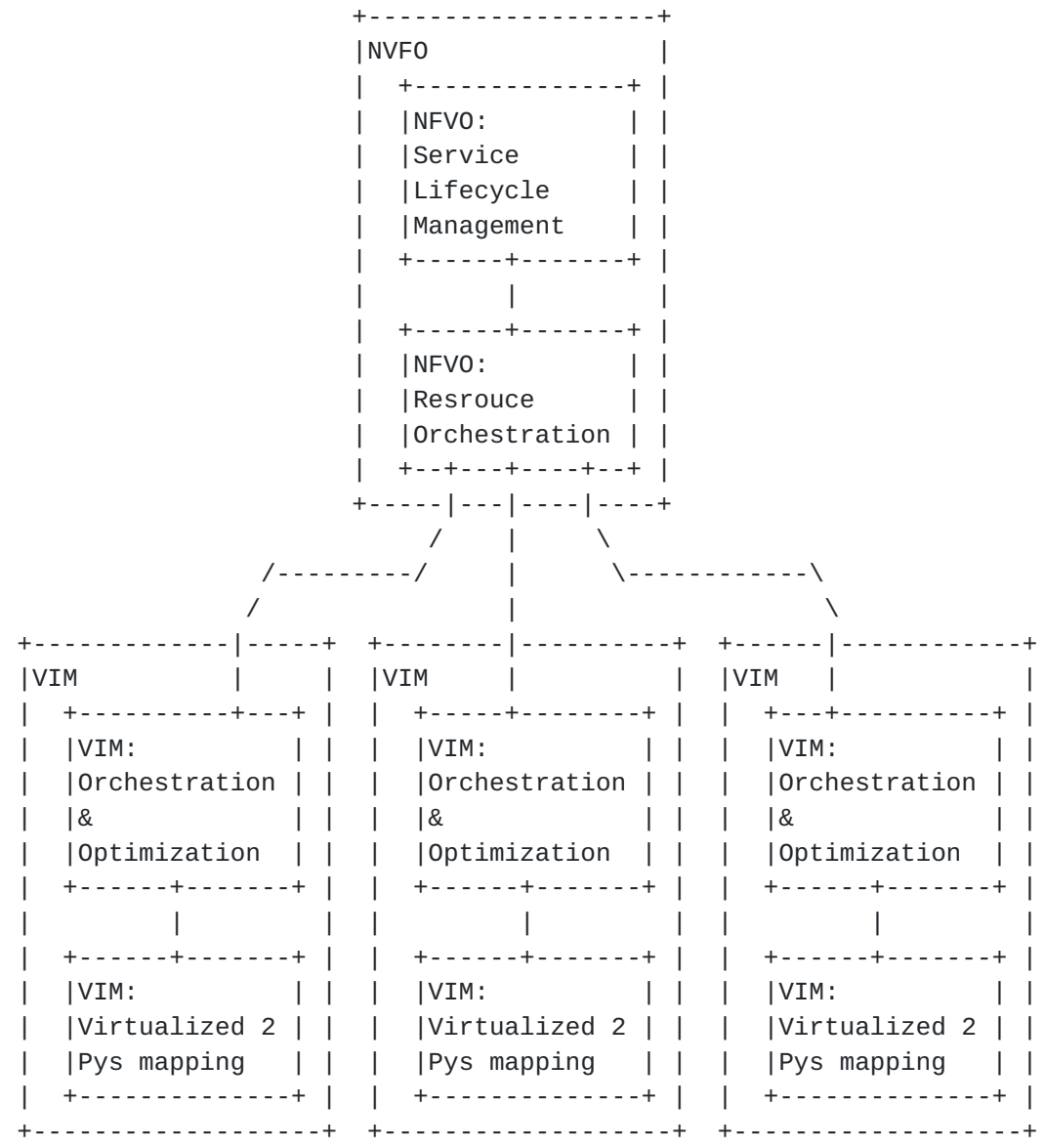

Figure 10: Functional decomposition of the NFVO and the VIM according to the ETSI MANO

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.

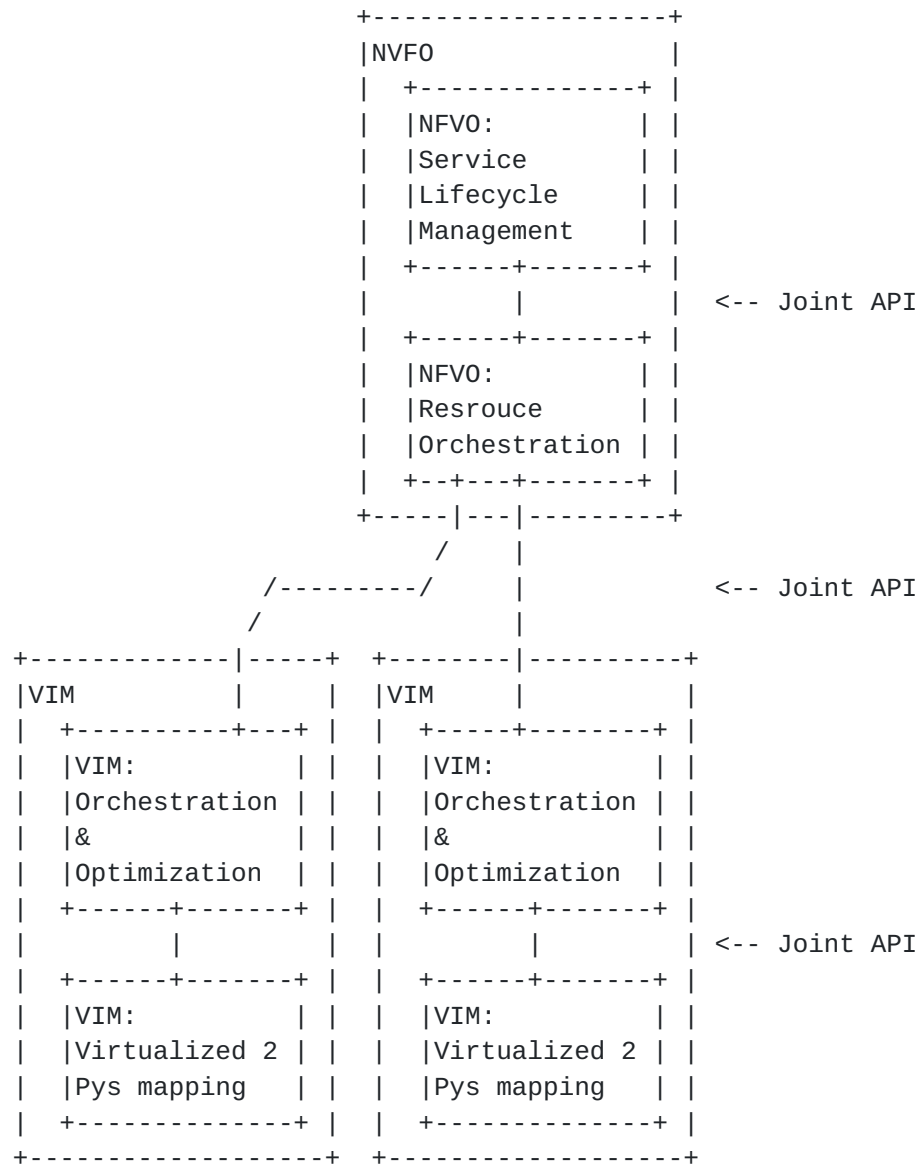

Figure 11: Functional decomposition of the NFVO and the VIM with the Joint Software and Network control API

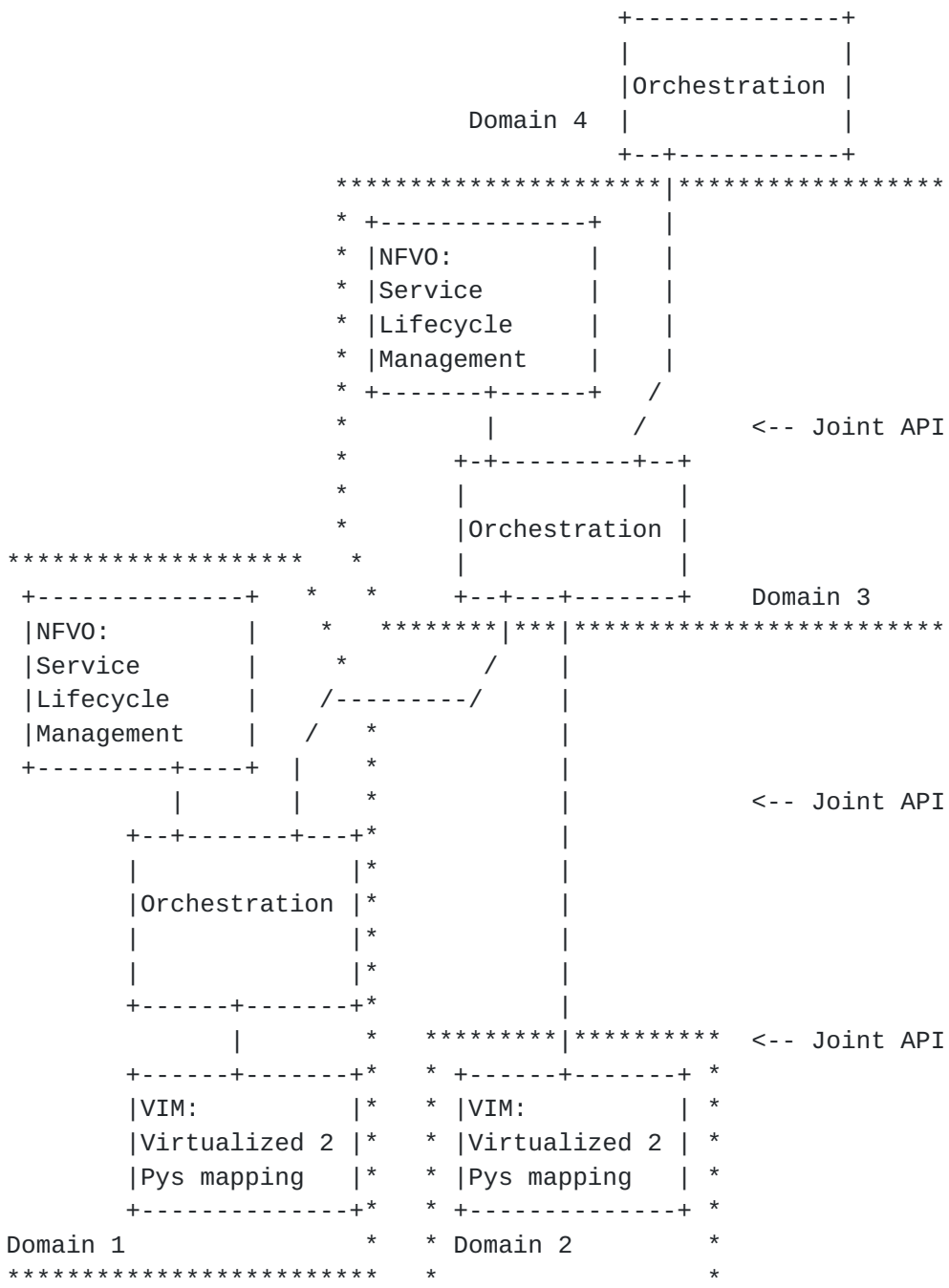

Figure 12: Joint Software and Network Control API: Recurring Flexible Architecture

6. Examples

6.1. Infrastructure reports

Figure 13 and Figure 14 show a single node infrastructure report. The example shows a BiS-BiS with two ports, out of which Port 0 is also a Service Access Point 0 (SAP0).

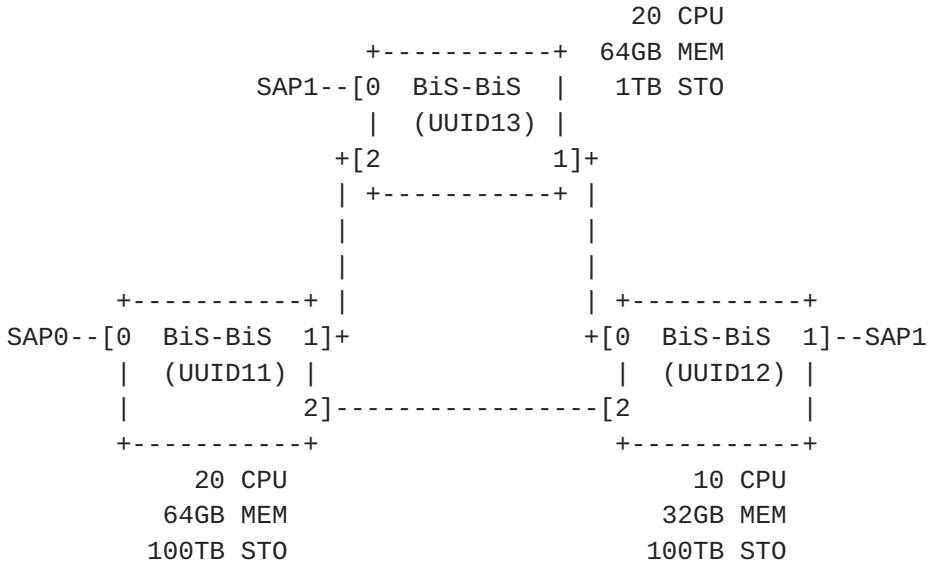

Figure 13: Single node infrastructure report example: Virtualization view

```
<virtualizer xmlns="http://fp7-unify.eu/framework/virtualizer">
    <id>UUID001</id>
    <name>Single node simple infrastructure report</name>
    <nodes>
        <node>
            <id>UUID11</id>
            <name>single Bis-Bis node</name>
            <type>BisBis</type>
            <ports>
                <port>
                    <id>0</id>
                    <name>SAP0 port</name>
                    <port_type>port-sap</port_type>
                    <vxlan>...</vxlan>
                </port>
                <port>
                    <id>1</id>
                    <name>North port</name>
                    <port_type>port-abstract</port_type>
                    <capability>...</capability>
                </port>
                <port>
                    <id>2</id>
                    <name>East port</name>
                    <port_type>port-abstract</port_type>
                    <capability>...</capability>
                </port>
            </ports>
            <resources>
                <cpu>20</cpu>
                <mem>64 GB</mem>
                <storage>100 TB</storage>
            </resources>
        </node>
    </nodes>
</virtualizer>
```

Figure 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.

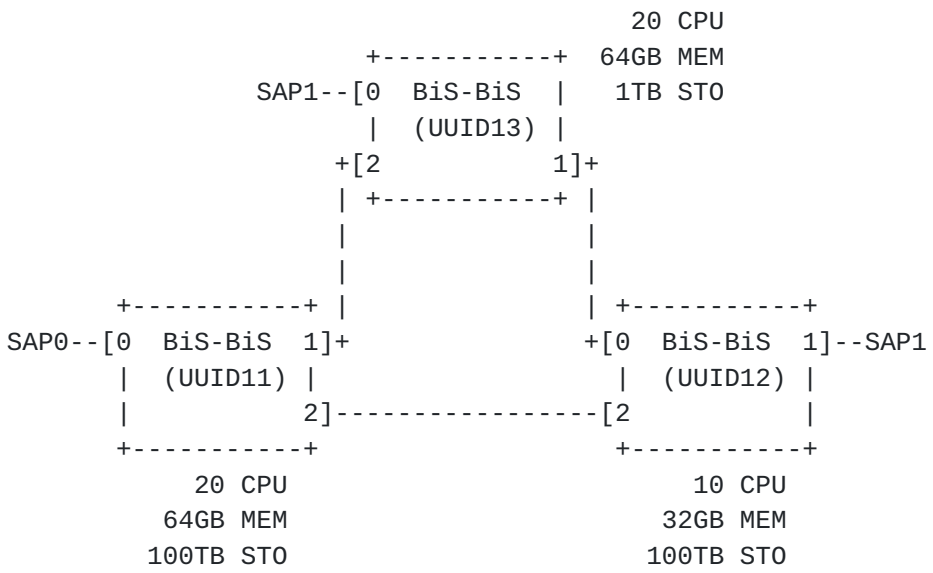

Figure 15: 3-node infrastructure report example: Virtualization view

```
 <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>
    </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.

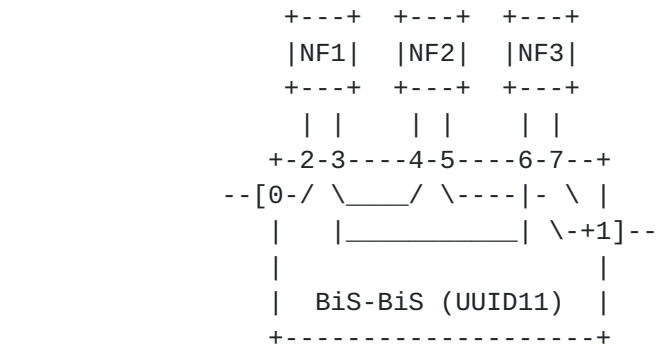

Figure 17: Simple request of 3 NFs on a single BiS-BiS: Virtualization view

```
<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>
    rpcre                </flowentry>
                    <flowentry>
                        <port>../../NF_instances/node[id=NF1]
                          /ports/port[id=3]</port>
                        <match>fr-b</match>
                        <action>output:../../NF_instances/node[id=NF3]
                          /ports/port[id=6]</action>
                    </flowentry>
                    <flowentry>
                        <port>../../NF_instances/node[id=NF2]
                          /ports/port[id=5]</port>
                        <match>*</match>
                        <action>output:../../ports/port[id=1]</action>
                    </flowentry>
                    <flowentry>
                        <port>../../NF_instances/node[id=NF3]
                          /ports/port[id=7]</port>
                        <match>*</match>
                        <action>output:../../ports/port[id=1]</action>
                    </flowentry>
                </flowtable>
            </node>
        </nodes>
    </virtualizer>
```

Figure 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>.

[ETSI-NFV-MANO]
ETSI, "Network Function Virtualization (NFV) Management and Orchestration V0.6.1 (draft)", Jul. 2014, <http://docbox.etsi.org/ISG/NFV/Open/Latest_Drafts/NFV-MAN001v061-%20management%20and%20orchestration.pdf>.

[I-D.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