TNOVA

NETWORK FUNCTIONS AS-A-SERVICE
OVER VIRTUALISED INFRASTRUCTURES

GRANT AGREEMENT NO. 619520

Deliverable D5.32

Network Functions Implementation and Testing – Final

Editor P. Paglierani (ITALTEL)

Contributors P. Comi, M. Arnaboldi (ITALTEL), Y. Rebahi (FOKUS), A. Abujoda (LUH), F. Pedersini, A. Petrini (UNIMI), N. Herbaut (VIOTECH), A. Kourtis, G. Xilouris (NCSRD), G. Dimosthenous (PTL), K. Karras (FINT)

Version 1.0

Date July 11rd, 2016

Distribution PUBLIC (PU)

Executive Summary

This deliverable reports the implementation of the seven Virtual Network Functions (VNFs) developed in T-NOVA. The VNF descriptions reported here must be considered the final ones, therefore they update and eventually correct the ones contained in the previous version of this Deliverable, namely [D5.31]. This document also discusses the final version of the VNF Descriptor (VNFD), thus updating the information reported in the previous Deliverable [5.31]. The T-NOVA VNFD was designed and adopted within the project in line with ETSI guidelines and is publicly released to be reused by the NFV community.

The VNFs developed in T-NOVA are the following:

- Security Appliance
- Session Border Controller
- Video Transcoding Unit
- Traffic Classifier
- Home gateway
- Proxy as a Service
- FPGA-based H.264 Decoder.

Such VNFs span a very wide area of the Network Function domain, and can thus represent, from a developer's perspective, a set of highly significant implementation use cases, in which many aspects related to network function virtualization have been effectively tackled. In the VNFs presented in this document different technologies have been adopted by developers. Most VNFs, in fact, take advantage of various contributions coming from the open source community, such as [SNORT], or exploit recent technological advances, such as [DPDK], SR-IOV [Walters], general purpose Graphical Processing Units [CUDA], or Field Programmable Gate Arrays (FPGAs).

In addition, this final deliverable provides practical information useful to function developers, most of it related to the most significant implementation issues encountered in the development phase. The practical lessons learnt during implementation by T-NOVA developers have been summarized in a specific section. Also, a discussion about the scaling mechanism adopted in T-NOVA is included. Finally, this document provides guidelines for the VNF characterization, as a result of the activities carried out in Task 5.4. The results obtained in the VNF performance tests are reported and discussed. A thorough characterization of two specific VNFs, i.e. the vSBC and the vTU, is reported in details in Annex A and B, as meaningful and complete examples of application of the guidelines developed in Task 5.4.

Table of Contents

Index of Figures

1. INTRODUCTION

This document contains the final description of the Virtual Network Functions (VNFs) developed in the T-NOVA project. In particular, seven different VNFs are discussed, covering a wide range of applications, which are:

- Virtual Security Appliance;
- Virtual Session Border Controller;
- Virtual Transcoding Unit;
- Traffic Classifier;
- Virtual CDN/Virtual Home Gateway;
- Proxy as a Service;
- FPGA-based H.264 decoder.

For each VNF, architectural and functional descriptions are provided, along with the technologies used and the internal/external interfaces. In addition, guidelines to characterize the VNF performance are given, as a result of the activities carried out in Task 5.4. The results obtained in the tests are summarized for every VNF.

Function providers who want to develop new VNFs compatible with the T-NOVA framework can use them as guideline examples. To offer practical guidelines to developers, a specific section about the practical lessons learnt during the T-NOVA development phase has been introduced. The final versions of the specific VNFs developed in T-NOVA are described.

The structure of the document is the following.

Section 2 presents the T-NOVA VNF Descriptor in its final version.

Section 3 presents the final version of the description of the seven VNFs developed in T-NOVA. Such descriptions should be considered the final versions, and therefore they update all the information contained in previous documents.

Section 4 is focused on the scaling approach implemented in T-NOVA. A general description is first given in sub-section 3.1; then, in 3.2 and 3.3 the scaling procedures implemented by two different VNFs, namely the vSBC and the vHG/vCDN are presented and discussed.

Section 5 reports the lessons learnt by developers during the various phases of the project, in particular related to the most innovative aspects considered in T-NOVA, such as the use of different types of HW accelerators and their impact on VNFs.

Section 6 presents the guidelines for the performance characterization of VNF, developed in Task 5.4 of T-NOVA WP5. A thorough application of such guidelines is described in Annex A and B, which report the results obtained in the testing phase of two VNFs, namely the vSBC and the vTU.

Finally, Section 7 draws the final conclusion and summarizes the main results reported in this document.

1.1. Dependency on previous documents

This report contains the research, design and implementation results and ideas developed in the WP5 "Network Functions" work-package of the T-NOVA project. This work-package has mainly taken inputs from WP2, "System Specification" and in particular from T2.5 "Specification of Network Function framework". The inputs to WP5 about Network Functions are summarized in the T-NOVA deliverable [D2.41]. The activities carried out within WP5 have been described in the previous deliverables [D5.01] and [D5.31]. Thus, the information reported in this deliverable updates and eventually corrects the one contained in the previous version of this deliverable, namely [D5.31]. This document also discusses the final version of the VNF Descriptor (VNFD), thus updating the information reported in [D5.31]. Other documents of interest for this deliverable are [D2.1], about T-NOVA use cases and requirements, [D2.21] for the information related to the overall T-NOVA architecture, [D4.01] and [D4.1], about infrastructure virtualization, and [D6.1], related to the T-NOVA Marketplace.

2. THE VNF DESCRIPTOR

This section describes the VNF Descriptor, or simply the VNFD, designed and adopted in T-NOVA, in line with ETSI guidelines.

2.1. VNF Descriptor (VNFD)

The VNFD plays an important role in the proper deployment of a particular VNF in the NFVI by the NFVO, as well as in the portability of the VNF to NFVI variants. A preliminary description of the VNFD was given in D5.31. This section presents the final version of the T-NOVA VNFD (also available in the T-NOVA repository, at https://github.com/T-NOVA/NFVdescriptors). The figure below (Figure 1) presents the generic structure of the information model for the description of the VNF properties as is specified by ETSI and the currently supported model by T-NOVA.

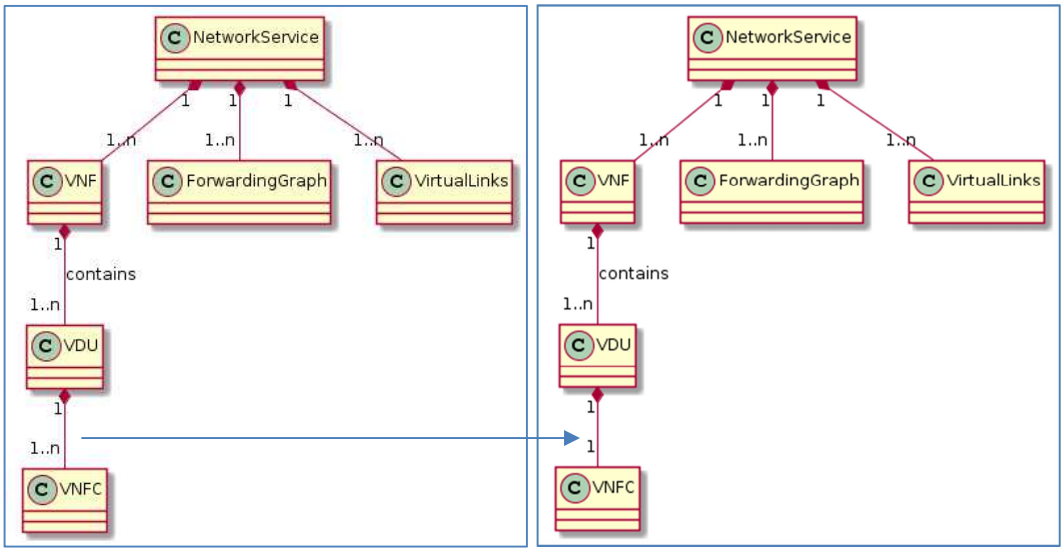


Figure 1 T-NOVA versus ETSI model

As it can be observed T-NOVA adapted ETSI model by simplifying the VDU – VNFC mapping. In this view, the assumption that every VNFC (VNF component) equals to one and only one Virtual Deployment Unit (VDU - as well-known as Virtual Machine (VM) in the cloud terminology). This decision is justified by the current implemented VNFs from T-NOVA that are based on VMs and not on Containers (i.e. Docker). However, the model and the relevant mechanisms can be easily extended to support many VNFC to VDU models.

The detailed structure of the VNFD is illustrated in the following (Figure 2). The figure presents the main classes used in the VNF descriptor information model. The classes are detailed furthermore in the following subsections.

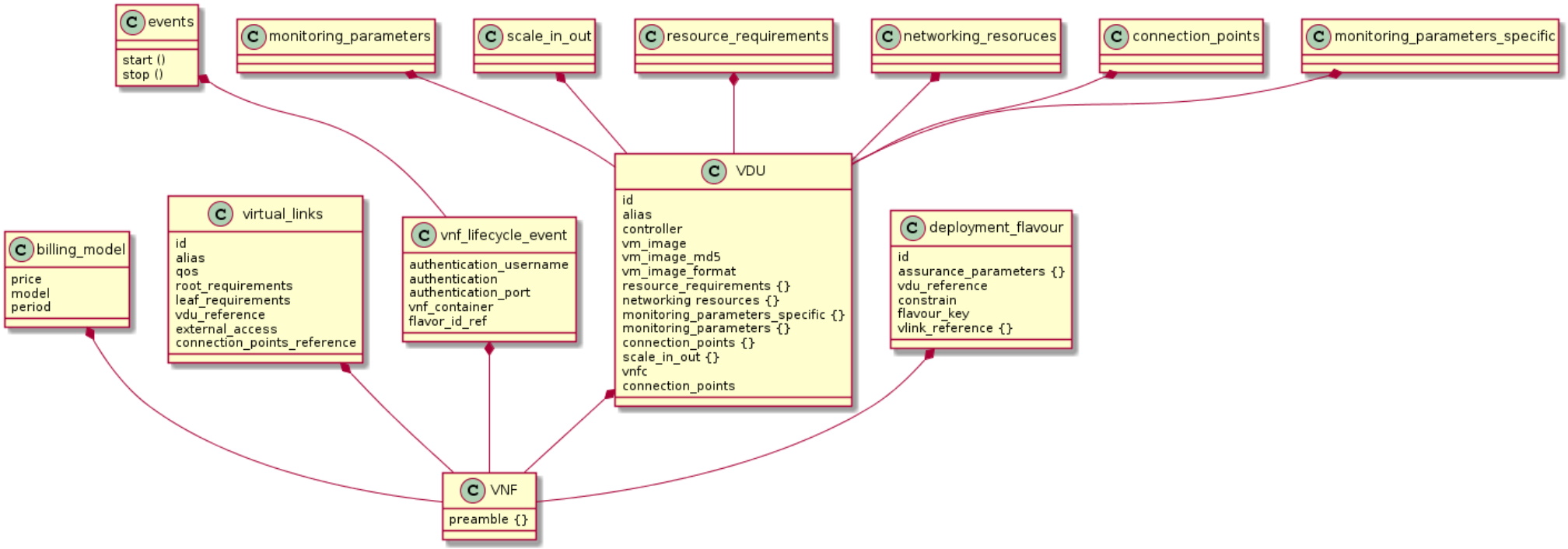


Figure 2 VNF Descriptor model

2.1.1. VNFD Preamble

The VNFD preamble provides the necessary information for the release, id, creation, provider etc. T-NOVA extents the information with Marketplace related information such as trading and billing.

Listing 1 VNF Preamble

VNFD Preamble		
field	**Description**	**Example**
release	Release Information, this field indicates the supported release scheme for the VNFD structure expected to be used from the parser	"0.2"
Id	UUID of the VNFD file appended to the VNFD during the upload at the NFStore. Provides a unique identification the VNFD file	624
provider	VNF Provider (FP) name	"PTL"
provider_id	ID of the FP as allocated by the T-NOVA Marketplace	21
name	VNF Name	"PXaaS"
description	VNF description	"The function identifies, classifies and forwards network traffic according to policies"
descriptor_version	VNFD version	0.1
version	VNF version	0.2
manifest_file_md5	Calculated by the NFStore during upload	fa8773350c4c236268f0bd7807c8a3b2
type	Signals the type of the VNF i.e. L2 or L3	L2
created_at	Date VNFD was created	2015-12-18T21:15:47Z

2.1.2. Virtual Deployment Unit (VDU)

The VDU segment of the descriptor provides information about the required resources that will be utilised in order to instantiate the VNFC. The configuration of

this part may be extremely detailed and complex depending on the platform specific options that are provided by the developer. However, it should be noted that the more specific are the requirements stated here the less portable the VNF might be, depending on the NFVO policies and the SLA specifications. It is assumed that each VDU describes effectively the resources required for the virtualisation of one VNFC.

The listing below (**Erreur ! Source du renvoi introuvable.**) presents the VDU section of the VNFD focusing on the IT resources and platform related information. Other fields may also be noted such as: i) Lifecycle events – where the drivers for interfacing with the VNF controller are defined as well as the appropriate commands allocated to each lifecycle event; ii) scaling – defining the thresholds for scaling in-out; and iii) VNFC related subsection where the networking and inter-VNFC Virtual Links are defined.

Listing 2 VDU Descriptor section

vdu		
field	**Description**	**Example**
Id	VDU id	“vdu0”
alias	Alias used to refer to this vdu	“tc_core_vm”
vm_image	Location of the vm image	"http://store.t-nova.eu/NCSRDv/TC_ncsrd.v.022.qcow"
vm_image_md5	VM image MD5 hash	"a5e4533d63f71395bdc7debd0724f433"
vm_image_format	Image format. NFStore should store this (Allowed values: ami, ari, aki, vhd, vmdk, raw, qcow2, vdi, iso)	“qcow2”
resource_requirement {}	Array of vdu resource requirements	
Hypervisor_parameters {}	Hypervisor parameters array	
Version	Hypervisor version	10002|12001|2.6.32-358.el6.x86_64
Type	Hypervisor type	QEMU-KVM
network_interface_bandwidth_unit	Units used for the bandwidth	Mbps
network_interface_bandwidth	Network interface bandwidth (10/100/1000)	100

	check the QoS fields too	
network_interface_card_capabilities{}	Array of network interface card capabilities	
SR-IOV	True/False for SR-IOV usage	True
Mirroring	True/False if mirroring on multiple ports is requested	false
Device_pass_through	True/False if pass_through will be configures	True
Storage{}	Storage resources array	
Size_unit	Unit used to declare size	GB
Persistence	True/false if persistence storage is used	False
Size	Disk Size	32
Vcpus	Number of virtual CPU cores	1
Vswitch_capabilities{}	Vswitch capabilities array	
Version	Vswitch version	2.0
Type	Vswitch type (e.g. OVS or Linux Bridge etc)	Ovs
Overlay_tunnel	Tunneling used for the virtual networks	GRE
memory	Memory size in GB	1
Memory_parameters {}	Memory parameters array	
Large_pages_required	False/true for large page support	False

Numa_allocation_policy	NUMA allocation policy (VMA, Task/Process etc)	None
CPU_support_accelerator	CPU acceleration support	AES-NI
data_processing_acceleration_library	Acceleration library support	DPDK

2.1.3. VNFC section

This section of VNFD is related to the internal structure of the VNF and describes the connection points of the VDU (name id, type) and the virtual links where they are connected. The above information is illustrated in the provide VNFD listing below (**Erreur ! Source du renvoi introuvable.**).

Listing 3 VNFC section of the VDU section

VNFC		
id	VNF component id	Vdu0:vnfc0
alias	VNFC alias	Proxy
controller	Check if this is the controlling VNFC	True
connection_points {}	Array of connection points	
- vlink_ref	Reference for the vlink where the connection point belongs	vl0
- id	Connection point ID	CPzc4j

The above information is parsed and translated to HEAT template that the NFVI VIM based on Openstack (NOVA/Neutron Services) is able to parse and provide accordingly the required networks.

2.1.4. Virtual Links (vlinks) section.

This section is used to specify the internal to the VNF virtual links that are employed by the developer in order to create the internal networking topology among the components used to build the VNF. Although the Function Provider in T-NOVA is free to select his own internal structure to service the requirements of his VNF, T-NOVA additionally imposes a limited set of virtual segments mandatory for the proper deployment of the VNF in T-NOVA PoPs. Thus some of the vlinks that are specified here are actually connecting the VNFC to those mandatory networks. The virtual networks that should be defined by the FP are:

- Management network, also used for the transfer of monitoring data
- Data-in network (in case in/out is preferred to be done by different virtual interfaces)
- Data-out network, the network used for the exit traffic from the VNF
- Storage network, the network used for storage traffic (i.e. iscsi)

The vlink section is illustrated in the following listing (Listing 4)

Listing 4 vlink section of VNFD

Vlinks {}		
Id	Virtual Link ID	vl0
Connectivity_type	E-LINE/E-LAN/E-TREE	E-LINE
Vdu_reference []	Reference the VDU ids (comma separated). Those VDU are connected via this vlink	vdu0
External_access	Signal the access to public internet	True
Connection_points_reference	Array of connection points	"CPv41w","CPv41w"
Access	Access to the internel	false
Dhcp	DHCP allocated addresses	False
Qos {}	QoS specific values (not used)	
Leaf_requirement	Leaf rate (unlimited or a specific rate)	unlimited
Root_requirement	Root rate (unlimited or a specific rate)	unlimited

For full review of the VNFD JSON format file, reader is welcomed to run through the Annexes included in this document. An annotated example is included.

3. T-NOVA VNFs

In the following, the final description of the VNFs developed in T-NOVA is reported.

3.1. Virtual Security Appliance

3.1.1. Introduction

A Security Appliance (SA) is a "device" designed to protect computer networks from unwanted traffic. This device can be active and block unwanted traffic. This is the case for instance of firewalls and content filters. A security Appliance can also be passive. Here, its role is simply detection and reporting. Intrusion Detection Systems are a good example. A virtual Security Appliance (vSA) is a SA that runs in a virtual environment.

In the context of T-NOVA, we have suggested a virtual Security Appliance (vSA) composed of a firewall, an Intrusion Detection System (IDS) and a controller that links the activities of the firewall and the IDS. The vSA high level architecture was discussed in details in [D5.01].

3.1.2. Architecture

The idea behind the vSA is to let the IDS Analyze the traffic targeting the service and if some traffic looks suspicious, the controller takes a decision by, for instance, revising the rules in the firewall and block this traffic.

The architecture of this appliance is depicted in Figure **3** and includes the following main components.

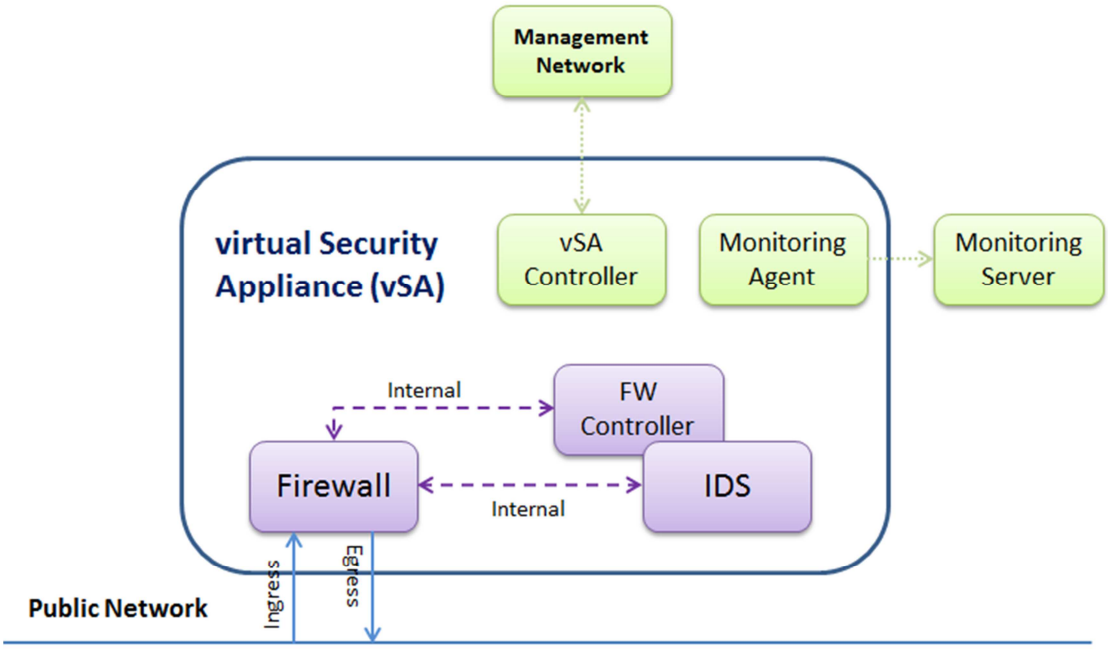


Figure 3. vSA high-level architecture

3.1.3. Functional description

The components of the architecture are the following:

- **Firewall:** this component is in charge of filtering the traffic towards the service.
- **Intrusion Detection System (IDS):** in order to improve attack detection, a combination of a packet filtering firewall and an intrusion detection system using both signatures and anomaly detection is considered. In fact, Anomaly detection IDS has the advantage over signature based IDS in detecting novel attacks for which signatures do not exist. Unfortunately, anomaly detection IDS suffer from high false-positive detection rate. It is expected that combining both arts of detection will improve detection and reduce the number of false alarms. In T-NOVA, the open source signature based IDS [SNORT] is being used and will be extended to support anomaly detection as well. The mode of operation of the IDS component was also discussed in deliverable [D5.01];
- **FW Controller***:* this application looks into the IDS "alerts repository" and based on the related information, the rules of the firewall are revised. Figure **4** depicts a part of the FW Controller code.

```
CONTROLLER_LOG_FILE = "/var/log/vsa-controller"
LOGGER_NAME = "vsa-controller"
LOG_FORMAT = "%(asctime)s | %(name)s | %(levelname)s: %(message)s"


class Controller(object):

    def __init__(self, logger, snort_log_dir="/var/log/snort", snort_classification_conf="/usr/local/etc/snort/classification.config", interface='wan'):
        """
        Creates a new controller
        :param logger: Logger to use
        :param snort_log_dir: Directory of the snort unified 2 log files
        :param snort_classification_conf: Classification mapping config file
        :param interface: Interface name to use for blocking sources
        :return: New controller object
        """
        self.logger = logger

        # get the most recent snort log file
        log_file = sorted(os.listdir(snort_log_dir), reverse=True)[0]
        logger.info('Using snort log file: %s' % log_file)
        self.event_reader = unified2.SpoolEventReader(snort_log_dir, log_file, follow=True)
        self.classification_map = maps.ClassificationMap(open(snort_classification_conf, 'r'))
        self.interface = interface

    def create_fw_rule(self, interface, ip_to_block):
        """
        Creates a firewall rule for 'interface' to block traffic from 'ip_to_block'
        :param interface: Interface name
        :type interface: str
        :param ip_to_block: IP address that should be blocked
        :type ip_to_block: str
        """
        self.logger.info("Blocking {0:s} on interface {1:s}".format(ip_to_block, interface))
        ret_code = subprocess.call(['easyrule', 'block', interface, ip_to_block])
        if ret_code == 1:
            self.logger.error('Failed to create blocking rule')

    def monitor_events(self):
        """
        Starts event monitoring.
        """
        for event in self.event_reader:
            if event['priority'] <= 2:
                self.create_fw_rule(self.interface, event['source-ip'])


def main():
    # set up logging
    # log to file
    logger = logging.getLogger(LOGGER_NAME)
    file_log = logging.FileHandler(CONTROLLER_LOG_FILE)
    log_level = logging.DEBUG
    logger.setLevel(log_level)
    file_log.setLevel(log_level)
    formatter = logging.Formatter(LOG_FORMAT)
    file_log.setFormatter(formatter)
    logger.addHandler(file_log)

    controller = Controller(logger)
    controller.monitor_events()


if __name__ == "__main__":
    main()
```

Figure 4. A sample of code of the FW Controller

- **Monitoring Agent:** this is a script that reports to the monitoring server the status of the VNF through some metrics such as (Number of errors coming in/ going out of the wan/lan interface of pfsense, Number of bytes coming in/ going out of the wan/lan interface of pfsense, CPU usage of snort, Percent of the dropped packets, generate by snort, etc);
- **vSA controller**: this is the application in charge of the vSA lifecycle.

3.1.4. Interfaces

The different components of the architecture interact in the following way,

1. data packets are first of all filtered by the firewall (ingress interface) before being forwarded to the service (egress interface);

2. filtered data packets are sniffed by the IDS for further inspection (internal interface). The IDS will monitor and analyze all the services passing through the network;
3. data packets go through a signature based procedure. This will help in detecting efficiently well know attacks such as port scan attacks and TCP SYN flood attacks;
4. If an attack is detected at this stage, an alarm is generated and the firewall is instructed to revise its rules (internal interface);
5. If no attack is detected, no further action is required.

In addition to that, there are two extra interfaces: the first one is in charge of the vSA lifecycle management, and the second one monitors the status of the vSA and sends the related information to the monitoring server.

3.1.5. Technologies

As performance is one of the main issues when deploying software versions of security appliances, we started by providing a short evaluation of firewalls software that could run in virtual environments. The idea was not to go through all the relevant existing software but just the most popular ones that could be extended to fulfill the use case requirements. This evaluation was described in [D5.01]. It turns out that the open source firewalls that are richer and more complete are Vyatta VyOS and pfSense (please refer to [D5.01] for more details). In addition to that, VyOS seems to support REST APIs for configuration which are important in the integration with the rest of the T-NOVA framework.

These two options were also evaluated from the performance point of view and the results are discussed in [D5.01]. Based on this assessment, the pfSense firewall seemed to be the best option to be used within the vSA.

3.1.6. Dimensioning and Performance

To study the performance of firewalls, appropriate metrics are needed. Although the activities in this area are very scarce, we described in D5.01, potential metrics that could be used. This includes, throughput, latency, jitter, and goodput.

3.1.6.1. Testbed setup

For simplicity reasons, we have used Iperf [IPERF] for generating IP traffic in our tests. In fact, other IP traffic generators such as [DITG], [Ostinato], and IPTraf [IPTR] could have also been utilized. Iperf mainly generates TCP and UDP traffic at different rates. Diverse loads (light, medium, heavy) and different packet sizes are also considered. For analyzing IP traffic, we used "tcpdump" for capturing it and "tcptrace" to analyze it and generate statistics. ***The main difference with respect to the tests performed in [D5.01] is the fact that in this paper, the tests are performed on a cloud computing platform (not simply in VirtualBox) namely, Openstack***. This also enables to test some networking functionalities of OpenStack as the latter does

not offer much freedom and flexibility on arbitrary traffic steering. Similarly, to [D5.01], the undertaken tests are based on three main scenarios,

- Scenario one (No firewall): Here, we configure and check the connectivity between the Iperf client and the virtual proxy without a firewall in between. This enables us to test the capacity of the communication channel
- Scenario two (TCP traffic with firewall and no rules): Here, we check whether the introduction of the vSA (in particular, the firewall in between) generates extra delay. We also test the capacity of the vSA in this context
- Scenario three (with firewall and increasing number of rules): the objective of this scenario is to study the effect of introducing rules into the firewall of the vSA. To achieve this scenario, a script for the firewall is implemented in order to generate rules in an automatic way. The script is a shell script using specific API commands and generate blocking rules for random source IP addresses (excluding those used in the test setup) and the WAN interface. Here, the easyrule function of pfsense is extended. In this scenario, some tests are also performed using UDP instead of TCP

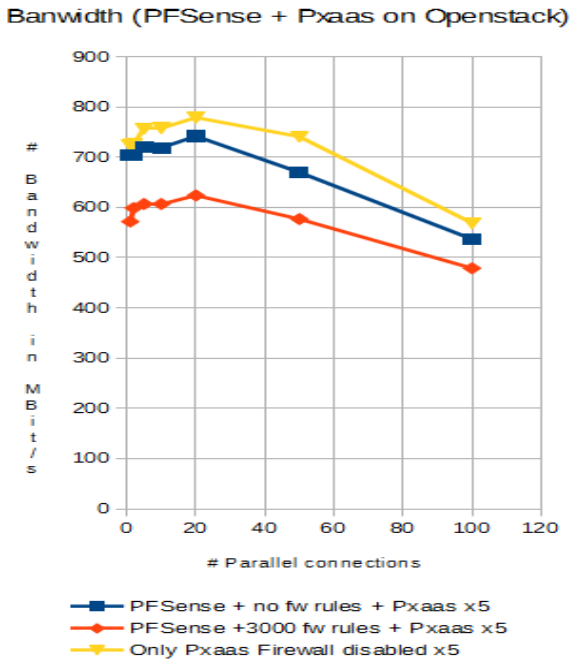


Figure 5. vSA throughput

When no firewall is used between the Iperf client and the virtual proxy, one can note that the throughput of the communication remains good (between 700 and 800 Mbit/s) as long as the number of 60 parallel connections is not exceeded. When the vSA (in particular the firewall) is in between, the throughput varies between 700 and 750 Mbit/s as long as the number of parallel connections does not exceed 20

connections. When the number of connections goes beyond the value 60, the throughput for the vSA without firewall rules decreases slowly to reach 580 Mbit/s when 100 connections are opened (Figure 5). This situation becomes worse when rules are configured on the firewall. Indeed, the throughput decreases to 480 Mbit/s when 3000 rules are configured and 100 connections are opened (Figure 5).

These results were included in a paper that was recently submitted to the Wiley Security and Communication Networks Journal.

3.2. Session Border Controller (SBC)

3.2.1. Introduction

A Session Border Controller (SBC) is typically a standalone device providing network interconnection and security services between two IP networks. It operates at the edge of these networks and is used whenever a multimedia session involves two different IP domains. It performs:

- the session control on the "control" plane, adopting SIP as a signalling protocol;
- several functions on the "media" plane (i.e: transcoding, transrating, NAT, etc), adopting Real time Transport Protocol (RTP) for the multimedia content delivery.

The virtual SBC (vSBC) is the VNF implementing the SBC service in T-NOVA virtualized environment, and it is a prototyped version of the commercial product that Italtel is developing for the NFV market.

General requirements for vSBC comprise both essential features such as IP to IP network interconnection, SIP signalling proxy, Media flow NAT, RTP media support, and also advanced requirements such as SIP signalling manipulation, real-time audio and/or video transcoding, Topology hiding, Security gateway, IPv4-IPv6 gateway, generation of metrics, … etc.. Since our goal in the T-NOVA project was to study the structure and the lifecycle of the vSBC by means of a meaningful prototype, we mainly focused on its essential features and on a subset of its advanced requirements (i.e: IPv4-IPv6 gateway; real-time audio and/or video transcoding for mobile and fixed network; metrics generation; ….. etc).

3.2.2. Architecture

The basic architecture of the virtualized SBC is depicted in Figure 6.

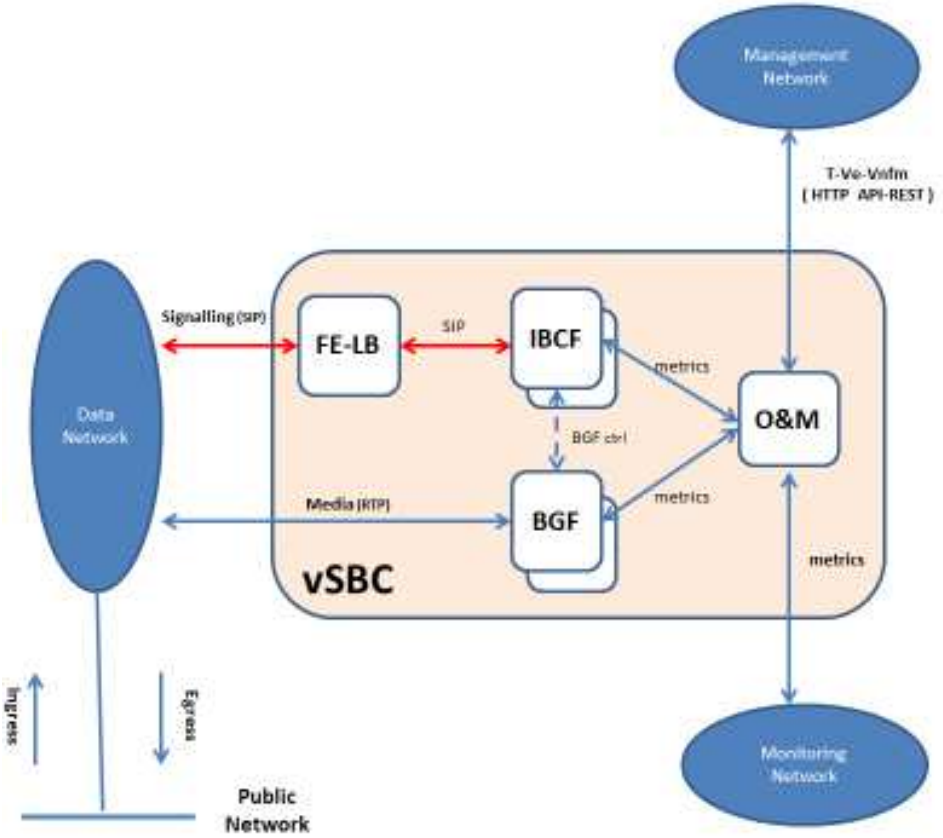


Figure 6 - Basic vSBC internal architecture

The basic vSBC consists mainly of:

- four Network Function: FE-LB, IBCF, BGF and O&M (described in detail below)
- one Management interface (T-Ve-Vnfm). It transports the HTTP commands of T-NOVA lifecycle from the VNFM to the O&M component
- one Monitoring Interface. The monitoring data produced by the internal VNFCs (i.e: IBCF and BGF), are collected by the O&M and are cyclically sent to the T-NOVA Monitoring Manager. See also Task 4.4 (Monitoring and Maintenance) for further details
- one Signaling interface (based on SIP protocol)
- one Media interface (based on RTP/RTCP protocols).

3.2.3. Functional description

1) *Front End-Load Balancer* (FE-LB): it is the front end of the vSBC and it balances the incoming SIP messages, forwarding them to the appropriate IBCF instance.
2) *Interconnection Border Control Function* (IBCF): it implements the control function of the SBC. It analyzes the incoming SIP messages, and handles the communication between disparate SIP end-point applications. The IBCF extracts from incoming SIP messages the information about media streams associated to the SIP dialog, and instructs media plane components (BGF) to process them. It can also provide:

- SIP message adaptation or modification, enabling in this way the interoperability between the interconnected domains
- topology hiding. The IBCF function hides all incoming topological information to the remote network
- monitoring data related to the SIP signalling (i.e: number of confirmed SIP sessions; … etc). The IBCF function can send this information to the T-NOVA Monitoring Agent
- other security features.

3) *Border Gateway Function* (BGF): it processes media streams, applying transcoding and transrating algorithms when needed. The transcoding transforms the algorithm used for coding the media stream, while the transrating changes the sending rate of IP packets carrying media content. This feature is used whenever the endpoints of the media connection support different codecs, and it is an ancillary function for an SBC because, in common network deployments, only a limited subset of media streams processed by the SBC need to be transcoded. The BGF is controlled by the IBCF using the internal *BG ctrl* interface (see Figure 6). The BGF component can also provide metrics to the T-NOVA Monitoring Agent related to the media flow, for example: number of incoming/outgoing RTP packets or octects; latency (maximum and average value); jitter (maximum and average value); RTP frame loss; number of incoming/outgoing transcoding and transrating procedures; ….. etc
4) *Operating and Maintenance* (O&M): it supervises the operating and maintenance functions of the vSBC and interacts (via HTTP) with the VNF manager, using the T-Ve-Vnfm interface depicted in Figure 6, for applying the T-NOVA lifecycle.

3.2.4. Interfaces

The most relevant internal and external interfaces depicted in Figure 6 are:

1) *Management Interface (T-Ve-Vnfm):* it is used to transport the HTTP commands of the T-NOVA lifecycle (i.e: start, stop, destroy, scale in/out, etc). It's supported by the O&M component;
2) *Monitoring Interface:* the monitoring data produced by the internal VNFCs (i.e: IBCF and BGF) are collected by the O&M and cyclically sent to the T-NOVA Monitoring Manager;
3) *Signalling interface*: it is based on SIP protocol. It carries also the SDP protocol containing the media information (i.e: Ip addresses/port, codecs, … etc) exchanged during the *Offer/Answer* negotiation;
4) *Media interface*: it is based on RTP/RTCP protocol and it carries the audio/video packets of SIP sessions handled by the vSBC;
5) *BG ctrl***:** this internal interface instructs the *Border Gateway Function* (BGF) to perform media transcoding/transrating procedures.

3.2.5. Technologies

The vSBC utilizes various technologies in order to offer a stable and high performance VNF, compliant to the high standards of legacy physical networks.

The development of the vSBC is based on the use of:

- Linux operating system
- KVM hypervisor
- C and Java language for its internal functions (i.e: IBCF, BGF, O&M, ….)
- Collectd (it periodically collects generic performance statistics of the Virtual Machine, such as CPU and memory utilization)
- FFMPEG libraries (for G.711 a/u, G.722 audio codecs)
- FFMPEG + VISUALON libraries (for G722.2 audio codec)
- INTEL-IPP libraries (for G.729 audio codec)
- OPUS libraries (for OPUS audio codec)
- FFMPEG + X.264 libraries (for H.264 video codec)
- LIBVPX libraries (for VP8 video codec)

These various technologies/libraries used generate a great variety of test case scenarios, as described in par. 10.5.1.

3.2.6. Dimensioning and Performance

The vSBC performances can be monitored using the metrics generated either by its internal components (i.e: IBCF or BGF) or by the Collectd daemon of each Virtual Machine. See also [D4.41] for further information. For example:

1. monitoring data related to the control plane: total number of SIP sessions/transactions
2. monitoring data related to the media plane: incoming/outgoing RTP data throughput, RTP frame loss, latency, inter-arrival jitter, number of transcoding/transrating procedures, … etc
3. base monitoring data: percentage of memory consumption, percentage of CPU utilization, … etc

These monitoring data are strongly affected by:

- packet sizes;
- kinds of call (i.e: audio or video calls);
- audio/video codecs (i.e: H.264, VP8, …);
- transport protocols (i.e: UDP or TCP).

The vSBC performance are described in par. 10.6. The IBCF and BGF components provide market sensitive performances; for example the IBCF can support up to 1000 simultaneous SIP sessions, while the BGF up to 20 simultaneous

transcoding/transrating operations. In the commercial product each component size will be associated to a license fee.

3.2.7. vSBC testing

These kind of tests have been carried out:

- Creation of a VNF Descriptor (VNFD) for the vSBC using the Marketplace GUI
- Creation of a HEAT template (Hot) for the vSBC
- Instantiation of the vSBC
- Support of the T-NOVA lifecycle, using a HTTP REST-based interface and a basic access authentication. These events were tested:

  - *Start* (via http POST command)
  - *Stop* (via http PUT command) [1]
  - *Destroy* (via http DELETE command)
  - *Scale-in* (via http PUT command)
  - *Scale-out* (via http PUT command).

- Generation of basic audio sessions (without transcoding), using the most common audio codecs (i.e: G711, G729, …);
- Generation of basic video sessions (without transcoding), using the most common video codecs (i.e: H248, VP8, …);
- Generation of audio sessions with transcoding (for example G711 <-> G729);
- Generation of video sessions with transcoding (for example H248 <-> VP8). The requested transcoding may be mono-directional (i.e: audio/video stream distribution) or bidirectional (i.e.: videoconferencing applications). The encoding/decoding procedures must be handled in a real-time way (at least 30 fps) and with a fixed frame-rate during the whole video session.

- Scale-in scenario;
- Scale-out scenario.

Some general guidelines to describe how to test the vSBC are described in par. 6.1.

The methodology used to execute the Performance testing and to evaluate the obtained measurements by means of load curves are described in par. 10.5 and in par. 10.6.

These vSBC performances may be further improved in a commercial product by means of :

- the usage of HW accelerators (GPU). In fact the pure software implementation, in some scenarios such as the video transcoding, may be complemented by acceleration technologies able to guarantee a real time transcoding and a fixed

[1] Note: this lifecycle event will be handled in an immediate or graceful mode according to a specific vSBC internal data.

video-rate during the whole video session. Graphic Processing Units (GPU's) represent a very appealing solution owing to their high computation performance (one or two order of magnitude faster than a general purpose CPU) and relatively low cost. Different types of commercial GPU board, hosted in a PCIe bay, are already available for this scope (i.e: Nvidia GPU). Further information about GPUs and the gain offered by them can be found in in par. **Erreur ! Source du renvoi introuvable.** and in par. 3.3

- the implementation of a new front-end function based on DPDK acceleration technology (available in Intel x86 architectures), whose goal is to provide high speed in processing the addressing information in the header of the IP packet, leaving untouched the payload. This new function is instructed by the IBCF function, acting as an internal controller. Using a new internal interface it can:
  - provide the packet forwarding towards the BGF function (in case of transcoding)
  - apply a local Network Address Translation (NAT)/port translation.

3.3. Video Transcoding Unit

3.3.1. Introduction

The vTU provides the transcoding function for the benefit of many other VNFs for creating enhanced services.

3.3.2. Architecture

3.3.2.1. Functional description

The core task of the Video Transcoding Unit is to convert video streams from one video format to another. Depending on the applications, the source video stream could originate from a file within a storage facility, as well as coming in from of a packetized network stream from another VNF. Moreover, the requested transcoding could be mono-directional, as in applications like video stream distribution, or bi-directional, like in videoconferencing applications.

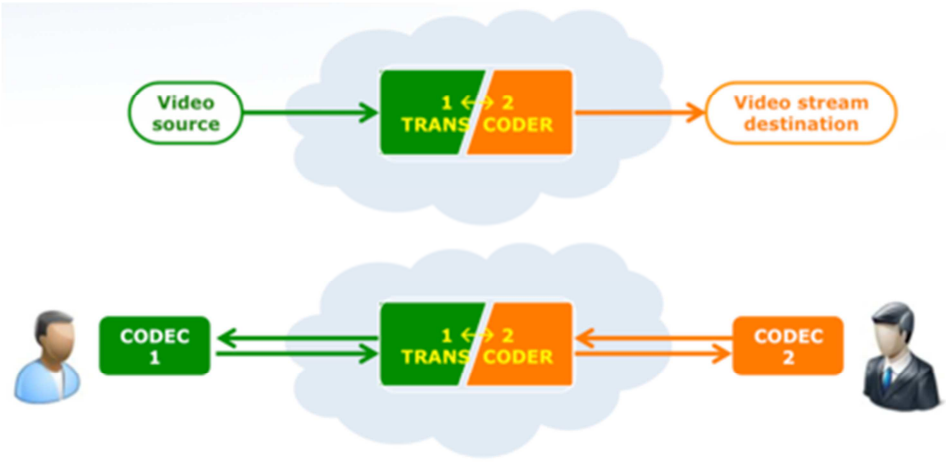


Figure 7. Functional description of the vTU

Having this kind of applications in mind, it is clear that the most convenient overall architecture for this Unit is a modular architecture, in which the necessary encoding and decoding functionalities are deployed as plug-in within a "container" Unit taking care for all the communication, synchronization and interfacing aspects. In order to find a convenient approach for the development of the vTU architecture, an investigation has been carried out, about the state of the art of any available software framework that could be usefully employed as starting point for this architecture. This investigation has identified **avconv**, the open-source audio-video library under Linux environments (https://libav.org/avconv.html), as the best choice for the basic platform for the vTU, as it is open-source, it is modular and customizable, and it contains most of the encoding/decoding plug-ins that this VNF could need.

In order to define the functionalities that best fit to the needs of the target applications for the vTU, a survey has been carried out, searching for the most diffused video formats that should therefore be present as encoding/decoding facilities in the vTU. This analysis has shown that the following video formats should be primarily considered:

- ITU-T **H.264** (aka AVC)
- Google's **VP8**

and the following ones would be also highly desirable, especially in the future:

- ITU-T **H.265** (aka HEVC)
- Google's **VP9**.

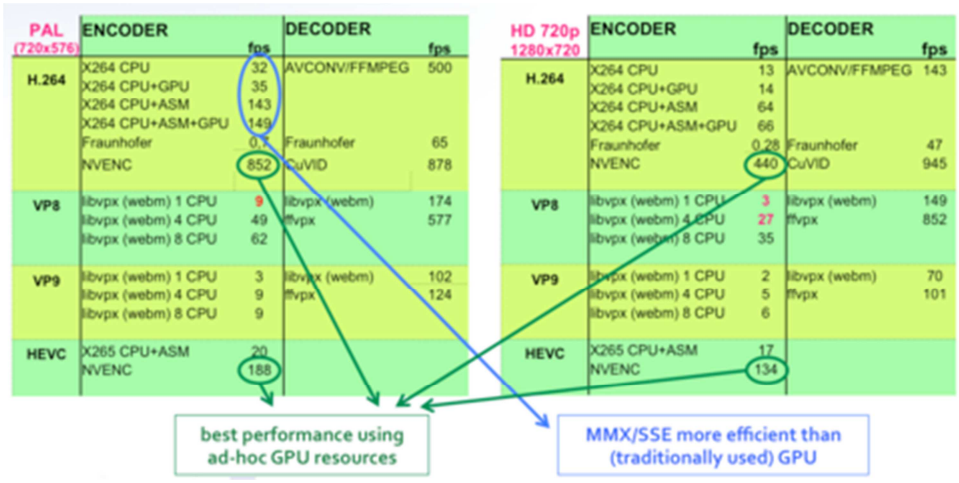
Once the video formats of interest are defined, the whole panorama of the available codecs have been considered and evaluated, in order to identify tools which could be successfully employed in the Unit and those which could be possibly used as development basis. A synthesis of the available panorama is shown in the following table:

codec library/package	capabilities (Encode/Decode)				uses HW acceleration
	H264	VP8	H265	VP9	
avconv (libav.org)	E D	E D	E D	E D	–
openh264 (CISCO)	E D				–
X264 / X265 (videolan.org)	E		E		GPU/ MMX / SSEx
JM (Fraunhofer – HHI)	E D		E D		–
NVenc (NVIDIA)	E		E		GPU + ad-hoc HW
cuVid (NVIDIA)	D				GPU

The analysis evidenced that the choice of avconv as the starting development framework is most appropriate in terms of already-available codecs.

From the point of view of performance, however, avconv could be unable to fulfil the needs of the vTU, as all the endoders/decoders provided therein do not make any use of HW acceleration facilities. Performance, in terms of encoding/decoding speed, is actually of crucial importance in the vTU, as in all online applications, a fixed video frame-rate must be guaranteed during the whole video session.

For this reason, a performance analysis of all available codecs has been carried out. Several of them, in fact, are able to exploit hardware acceleration facilities, like GPU's or MMX/SSEx instructions, whenever they are available. The tests have been carried out considering a typical scenario for the underlying hardware infrastructure: a Xeon server with 8 cores (2 x 4-cores) Xeon E5-2620v3, equipped with GPU facility (1 NVIDIA GeForce GTX 980). Several video test sequences have been considered, at the most common resolutions. The obtained results, in terms of achieved encoded/decoded frames per second, are summarized in the following tables, for PAL (576x720 pixel), 720p (1280x720), HD (1920x1080) and 4k (3840x2160) resolutions:



Based on the obtained results, the following observations can be drawn:

- As expected, encoding is much more time-consuming than decoding. On average, decoding is approximately 20 times faster than encoding, for the same format. The consequence is that encoders represent the bottleneck to performance in a vTU. For what decoding concerns, the tested tools have

performed faster than 30 fps in all situations, at least for H264 and VP8 standards, which are those currently use;

- Hardware-accelerated tools are not only providing much better performance than CPU-based ones, as visible in the tables for all resolutions, but is necessary in some cases, e.g. for 4k resolutions, where standard algorithms are not able to reach 30 fps encoding speed and therefore could not support a real-time transcoding session;

- As highlighted in the first table, different hardware accelerators can be successfully exploited to speed-up the encoding process – not only GPU's. In particular, X264 performs significantly better using Assembly-level optimizations which exploit SIMD instructions (MMX/SSE), than delegating computation to GPU cores.

This is due to the fact that encoding/decoding algorithms cannot be massively parallelized, for two main reasons: a) there are strong sequential correlations and many spatial/temporal dependencies within the computation, and b) the limited extent of parallelism needed in all situations where data parallelism could be applied (e.g. computing DCT/IDCT for a macroblock). Nevertheless, the huge computing power of modern GPU's makes it reasonable to focus the research effort towards the development of GPU-accelerated encoding algorithms, able to efficiently exploit the potential of all available cores. Therefore, as shown in the table above, the first goal on which we focus is the development of a GPU-accelerated encoder for the VP8 standard video format.

3.3.3. Interfaces

As described in Section 2.4.2.1, the Virtual Transcoding Unit (vTU) is a VNF that, during its normal operation, receives an input video stream, transcodes it and generates an output video stream in the new format. For each transcoding job, the vTU also needs to receive a proper job description, in which all necessary information is provided, like, for instance, information on the video format of the input stream and on the desired video format for the output stream, the identification and definition of the input/output data channels (e.g. IP addresses and ports, in case of network streams, or file ID within a storage facility, for file-generated streams).

For these reasons, the vTU needs, at its inner level, to communicate through three interfaces, as Figure **8** shows:

- **Input** port, receiving the video stream to transcode;
- **Output** port, producing the transcoded video stream;
- **Control** port, receiving the job descriptor and, implicitly, the command to start the job.

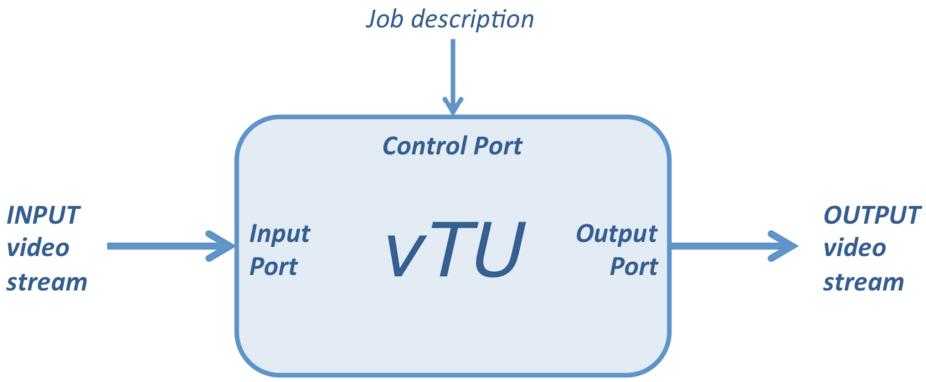


Figure 8. vTU low level interfaces

Through the Control interface, the vTU receives the job descriptor, which contains all necessary information to start the requested transcoding task. The starting command is implicit in the reception of the job description message: when such a message is received on the Control port, the vTU starts listening at the Input port and begins the transcoding task, according to the received description, as soon as the first stream packets are received.

The format of the job description message is XML based. The general structure of the message is shown in Figure **9**. This format allows to define all necessary parameters, such as the desired input and output video formats and the I/O stream channels (files, in this case, but they could as well identify network channels sending/receiving RTP packets).

```
<vTU>
  <in>
    <local>
      <stream>test.y4m</stream>
    </local>
    <rstp>
      <ip/>
      <port/>
      <stream/>
      <timeout/>
    </rstp>
    <codec>
      <vcodec/>
      <acodec/>
    </codec>
  </in>
  <out>
    <local>
      <overwrite>y</overwrite>
      <stream>out_test.h264</stream>
    </local>
    <rstp>
      <ip/>
      <port/>
      <stream/>
      <timeout/>
    </rstp>
    <codec>
      <vcodec>h264</vcodec>
      <acodec/>
    </codec>
  </out>
</vTU>
```

Figure 9. XML structure of the vTU job description message

3.3.4. Technologies

In the virtualization context, the problem of virtualizing a GPU is now well-known, and can be stated as follows: a guest Virtual Machine (VM), running on a hardware platform provided with GPU-based accelerators, must be able to concurrently and independently access the GPU's, without incurring in security issues [Walters],[Maurice]. Many techniques to achieve GPU virtualization have been presented. However, all the proposed methods can be divided in two main categories, which are usually referred to as API remoting [Walters] (also known as split driver model or driver paravirtualization) and PCI pass-through [Walters] (also known as direct device assignment [Maurice]), respectively. In the vTU, the passthrough approach has been adopted. For the sake of clarity, a brief review of this technology is shortly given in the next paragraph.

Pass-through techniques are based on the pass-through mode made available by the PCI-Express channel [Walters], [Maurice]. To perform PCI pass-through, an Input/Output Memory Management Unit (IOMMU) is used. The IOMMU acts like a traditional Memory Management Unit, i.e. it maps the I/O address space into the CPU virtual memory space, so enabling the access of the CPU to peripheral devices through Direct Memory Access channels. The IOMMU is a hardware device which provides, besides I/O address translations, also device isolation functionalities, thus guaranteeing secure access to the external devices [Walters]. Currently, two IOMMU implementations exist, one by Intel (VT-d) and one by AMD (AMD-Vi). To adopt the pass-through approach, this technology must also be supported by the adopted hypervisor. Nonetheless, Xenserver, open source Xen, VMWare ESXi, KVM and also the Linux containers can support pass-through, thus allowing VMs accessing external devices such as accelerators in a secure way [Walter]. The performance that can be achieved by the pass-through approach are usually higher than the one offered by API-remoting [Walter], [Maurice]. Also, the pass-through method gives immediate access to the latest GPU drivers and development tools [Walter]. A comparison between the performance achievable using different hypervisors (including also Linux Containers) is given in [Walter], where it is shown that pass-through virtualization of GPU's can be achieved at low overhead, with the performance of KVM and of Linux containers very closed to the one achievable without virtualization. One major drawback of pass-through is that it can only assign the entire physical GPU accelerator to one single VM. Thus, the only way to share the GPU is to assign it to the different VMs one after the other, in a sort of "time sharing" approach [Walters]. This limitation can be overcome by a technique also known as Direct Device Assignment with SR-IOV (Single Root I/O Virtualization) [Walters]. A single SR-IOV capable device can expose itself as multiple, independent devices, thus allowing concurrent hardware multiplexing of the physical resources. This way, the hypervisor can assign an isolated portion of the physical device to a VM; thus, the physical GPU resources can be concurrently shared among different tenants. However, to the best of the author's knowledge, the only GPU enabled to this functionality belongs to the recently launched NVIDIA Grid family [Maurice], [Walters]; also, the only hypervisors which can currently support this type of hardware virtualization are VMWare Sphere and Citrix XenServer 6.2. However, since also KVM can now support SR-IOV, there is a

path towards the use of GPU hardware virtualization also with this hypervisor [Walters].

3.3.5. Dimensioning and Performance

In order to obtain a realistic assessment of the performance of the vTU, as if it was embedded as a VNF in the T-NOVA framework, it is necessary to perform the tests on a virtualized platform resembling as much as possible the T-NOVA infrastructure.

The performance results presented in the table of Section 3.3.2.1. were obtained by the vTU running natively on physical computation resources. For a VNF like the vTU, however, the actual performance achievable in the T-NOVA environment could be quite different from those obtained running on the physical infrastructure. This is mainly due to the following reasons:

- CPU virtualization overheads (vCPU's switching over physical CPU's, at the hypervisor level);
- GPU virtualization strategies (e.g. multiple vGPU's associated to the same physical GPU);
- vCPU-vGPU communication overheads (switching overheads in managing time-sharing policies on the PCI-Express bus).

These reasons let one expect a possible performance loss, when running on a virtualized environment, especially in case of vTU running tasks which exploit the GPU resources.

For this reason, in order to obtain a realistic evaluation of the encoding/decoding computation speeds in the actual T-NOVA environment, the performance tests presented in Section 3.3.2.1. have been carried out on a virtualized environment. In order to get a significant comparison, the VM running the vTU has been equipped with the same amount of CPU and GPU cores as in the native tests. The following table presents the obtained results, in terms of computation speed (frames/sec), compared to the speed obtained on physical resources, for the same task.

Full HD (1920x1080)	ENCODER	Native time (s)	Virtualized time (s)	DECODER	Native time (s)	Virtualized time (s)
H.264	X264 CPU	6,97	7,26	AVCONV/FFMPEG	0,39	0,6
	X264 CPU+GPU	7,78	10,08			
	X264 CPU+ASM	1,55	1,85			
	X264 CPU+ASM+GPU	5,81	4,23			
	Fraunhofer	846	869	Fraunhofer	6,85	6,7
	NVENC	1,64	2,57	CuVID	1,53	2,24
	NVENC (AvConv)	2,4	2,64			
VP8	libvpx (webm) 1 thread	26,64	27,79	libvpx (webm)	0,37	4,21
	libvpx (webm) 4 thread	10,14	13,56	ffvpx	1,15	0,42
	libvpx (webm) 8 thread	8,33	7,2			
VP9	libvpx (webm) 1 thread	46,23	50,77	libvpx (webm)	2,23	1,94
	libvpx (webm) 4 thread	23,35	26,64	ffvpx	1,75	2,22
	libvpx (webm) 8 thread	18,87	26,21			
HEVC	X265 CPU+ASM	6,29	6,52	AVCONV/FFMPEG		1,19
	NVENC	2,9	1,71			

4k (3840x2160)	ENCODER	Native time (s)	Virtualized time (s)	DECODER	Native time (s)	Virtualized time (s)
H.264	X264 CPU	25,91	30,72	AVCONV/FFMPEG	1,38	1,74
	X264 CPU+GPU	22,21	24,36			
	X264 CPU+ASM	5,18	6,71			
	X264 CPU+ASM+GPU	5,89	9,81			
	Fraunhofer	3117	4714	Fraunhofer	22,37	24,49
	NVENC	2,72	8,63			
	NVENC (AvConv)	5,24	7,03			
VP8	libvpx (webm) 1 thread	78,63	76,4	libvpx (webm)	1,21	4,26
	libvpx (webm) 4 thread	32,44	36,78	ffvpx	4,37	1,54
	libvpx (webm) 8 thread	25,83	22,78			
VP9	libvpx (webm) 1 thread	146,19	155,2	libvpx (webm)	7,31	9,54
	libvpx (webm) 4 thread	60,7	75,23	ffvpx	5,43	9,16
	libvpx (webm) 8 thread	63,21	53,94			
HEVC	X265 CPU+ASM	6,29	17,88	AVCONV/FFMPEG	20,94	3,03
	NVENC	8,11	5,16			

As the table shows, the obtained results show that, for almost all the considered tasks, there is no significant performance loss with respect to the same task running on physical resources, even in the tasks running mainly on GPU (like H264 encoding using NVENC). This encouraging result is mainly due to the high efficiency of the adopted GPU virtualization strategy – GPU pass-through – which assigns a virtual GPU exclusively to a physical GPU, thus allowing to bypass any overhead in the GPU-CPU communication. The cost for this efficiency, however, is paid in terms of difficulty to share a physical GPU resource among multiple VMs.

3.3.6. Future Work

Two main steps are foreseen for the vTU. A first activity will focus on scaling mechanism for this VNF. Also, the vTU will be combined with other VNFs developed within T-NOVA in order to create new service with a wider scope.

3.4. Traffic Classifier

3.4.1. Introduction

The Traffic Classifier (TC) VNF used comprises of two Virtual Network Function Components (VNFCs), namely the Traffic Inspection engine and Classification and Forwarding function. The two VNFCs are implemented in respective VMs. The proposed Traffic Classification solution is based upon a Deep Packet Inspection (DPI) approach, which is used to analyze a small number of initial packets from a flow in order to identify the flow type. After the flow identification step no further packets are inspected. The Traffic Classifier follows the Packet Based per Flow State (PBFS) in order to track the respective flows. This method uses a table to track each session based on the 5-tuples (source address, destination address, source port, destination port, and the transport protocol) that is maintained for each flow.

3.4.2. Architecture

Both VNFCs can run independently from one another, but in order for the VNF to have the expected behaviour and outcome, the 2 VNFCs are required to operate in a parallel manner.

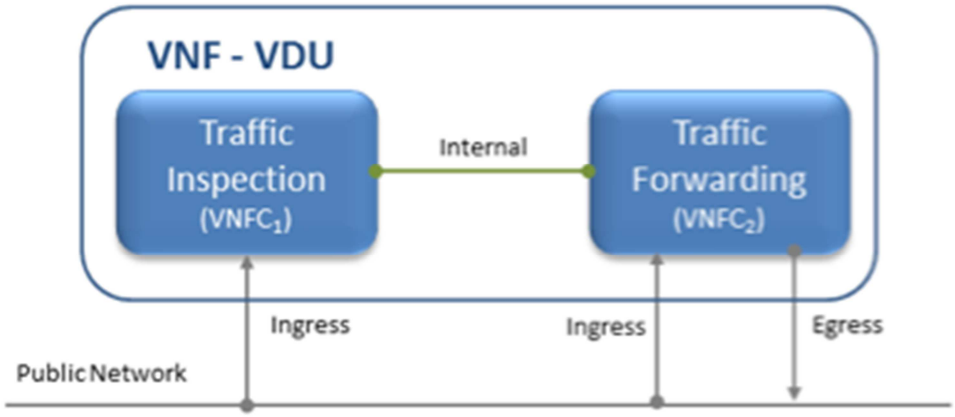


Figure 10. Virtual Traffic Classifier VNF internal VNFC topology

Furthermore, in order to achieve the parallel processing of the 2 VNFCs it is required for the traffic to be mirrored towards the 2 VNFCs, so the 2 VNFCs receive identical traffic. The 2 VNFCs are inter-connected internally with an internal virtual link, which transfers the information extracted by the Traffic Inspection VNFC, and transmits it to the Traffic Forwarding VNFC in order to apply the pre-defined rules.

3.4.3. Functional Description

The Traffic Inspection VNFC is the most processing intense component of the VNF. It implements the filtering and packet matching algorithms in order to support the enhanced traffic forwarding capability of the VNF. The component supports a flow

table (exploiting hashing algorithms for fast indexing of flows) and an inspection engine for traffic classification.

The Traffic Forwarding VNFC component is responsible for routing and packet forwarding. It accepts incoming network traffic, consults the Flow Table for classification information for each incoming flow and then applies pre-defined policies (i.e. TOS/DSCP (Type of Service/Differentiated Services Code Point) marking for prioritizing multimedia traffic) on the forwarded traffic. It is assumed that the traffic is forwarded using the default policy until it is identified and new policies are enforced. The expected response delay is considered to be negligible, as only a small number of packets are required to achieve the identification. In a scenario where the VNFCs are not deployed on the same compute node, traffic mirroring may introduce additional overhead.

3.4.4. Interfaces

The virtual Traffic classifier VNF is based upon the T-NOVA network architecture but from the advised set of network interfaces (management, datapath, monitoring and storage) uses the management, datapath and the monitoring. The storage interface is not particularly essential to the vTC, as all the computational and packet processing utilize mostly CPU and memory resources. The VNF requires intensive CPU tasks and a large number in memory I/Os for the traffic analysis, manipulation and forwarding. The storage interface would add an unnecessary overhead to the already intensive process, and it was decided to be excluded in favour of an optimal performance.

3.4.5. Technologies

The vTC utilizes various technologies in order to offer a stable and high performance VNF compliant to the high standards of legacy physical network functions. The implementation for the traffic inspection used for these experiments is based upon the open source nDPI library [REFnDPI]. The packet capturing mechanism is implemented using various technologies in order to investigate the trade-off between performance and modularity. The various packet handling/forwarding technologies are:

- **PF_RING**: PF_RING is a set of library drivers and kernel modules, which enable high-throughput, packet capture and sampling. For the needs of the vTC the PF_RING kernel module library is used, which is polling the packets through the LINUX NAPI. The packets are copied from the kernel to the PF_RING buffer and then they are analyzed using the nDPI library.
- **Docker**: Docker is a platform using container virtualization technology to run applications. In order to investigate the pros and cons of the container technology, the vTC is developed also as an independent container application. The forwarding and the inspecting of the traffic are also using PF_RING and nDPI as technologies, but they are modified to fit and function in a container environment.
- **DPDK**: DPDK comprises of a set of libraries that support efficient implementations of network functions through access to the system's network interface card (NIC). DPDK offers to network function developers a set of tools

to build high speed data plane applications. DPDK operates in polling mode for packet processing, instead of the default interrupt mode. The polling mode operation adopts the busy-wait technique, continuously checking for state changes in the network interface and libraries for packet manipulation across different cores. A novel DPDK-enabled vTC has been implemented in this test case in order to optimize the packet-handling and processing for the inspected and forwarded traffic, by bypassing the kernel space. The analyzing and forwarding functions are performed entirely on user-space which enhances the vTC performance.

The various technologies used generate a great variety of test case scenarios and exhibit a rich VNF test case. The PF_RING and Docker cases have the capability of keeping the NIC driver, and so the VNFC maintains connectivity with the OpenStack network connected. On the contrary, in the case of DPDK the NIC is unloaded of the Linux-kernel driver and loaded the DPDK one. However, the DPDK driver causes the VNFC to lose network connectivity with the network attached, the compensation is the significantly higher performance as shown in the next section.

3.4.6. Dimensioning and Performance

Results include comparison of the traffic inspection and forwarding performance of the vTC using PF_RING, Docker and DPDK.

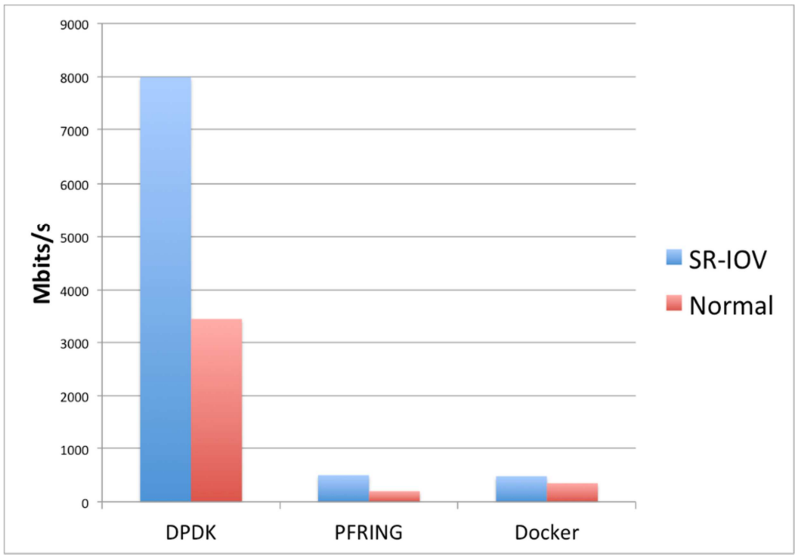


Figure 11. vTC Performance comparison between DPDK, PF_RING and Docker

As it can be seen from the evaluation results among the various approaches used for the vTC, the DPDK approach performs significantly better from the other 2 options. Especially in the case it is combined with SR-IOV connectivity it can achieve nearly 8Gbps/s of throughput. However, the DPDK version as already mentioned has an impact on connectivity with the OpenStack network, as the kernel stack is removed

from the NIC. Although the PF_RING and Docker versions maintain connectivity with the network, their performance is clearly degraded compared to DPDK's.

The Dimensioning of the vTC due to its architecture is based on the infrastructure aspect. The vTC performance is dependent on whether there is SR-IOV available on the running infrastructure.

3.4.7. Deployment Details

The vTC was developed in order to be deployed and run in an OpenStack environment, the OS of the virtualized environment was Ubuntu 14.04 LTS. The selection of the OS version assures the maintenance and continuous development of the VNF. In order to conform to the T-NOVA framework a Rundeck job-oriented service functionality was implemented.

The vTC lifecycle is performed via the Rundeck framework in order to facilitate the seamless functionality of the VNF. In Rundeck, we have created different Jobs to describe the different lifecycle events. Each event has a description and is part of a Workflow.

An example workflow:

```
If a step fails: Stop at the failed step.
Strategy: Step-oriented
We add a step of type "Command". The command differs according to the
operation we want to implement. The operations we implemented are
described below:
* 1. VM Configuration – Command: "~/rundeck_jobs/build.sh"
* 2. Start Service – Command: "~/rundeck_jobs/start.sh"
* 3. Stop Service – Command: "~/rundeck_jobs/stop.sh"
```

In terms of the data traffic required to test the vTC, several changes and modifications had to be made in order to fit the desired traffic mirroring scenario it was tested. Detailed information about this subject is further discussed in the section below.

3.4.7.1. Traffic Mirroring – Normal Networking

In order to support direct traffic forwarding, meaning the virtual network interface of one Virtual Network Function Component (VNFC) be directly connected to another VNFC's virtual network interface, a modification on Neutron's OVS needs to be applied. Each virtual network interface of a VNFC is reflected upon one TAP-virtual network kernel device, a virtual port on Neutron's OVS, and a virtual bridge connecting them. This way, packets travel from the VNFC to Neutron's OVS through the Linux kernel. The virtual kernel bridges of the two VNFCs need to be shut down and removed, and then an OVSDB rule needs to be applied at the Neutron OVS, applying an all-forwarding policy between the OVS ports of the corresponding VNFCs. The OpenStack network detailed topology is shown in Fig. 15.

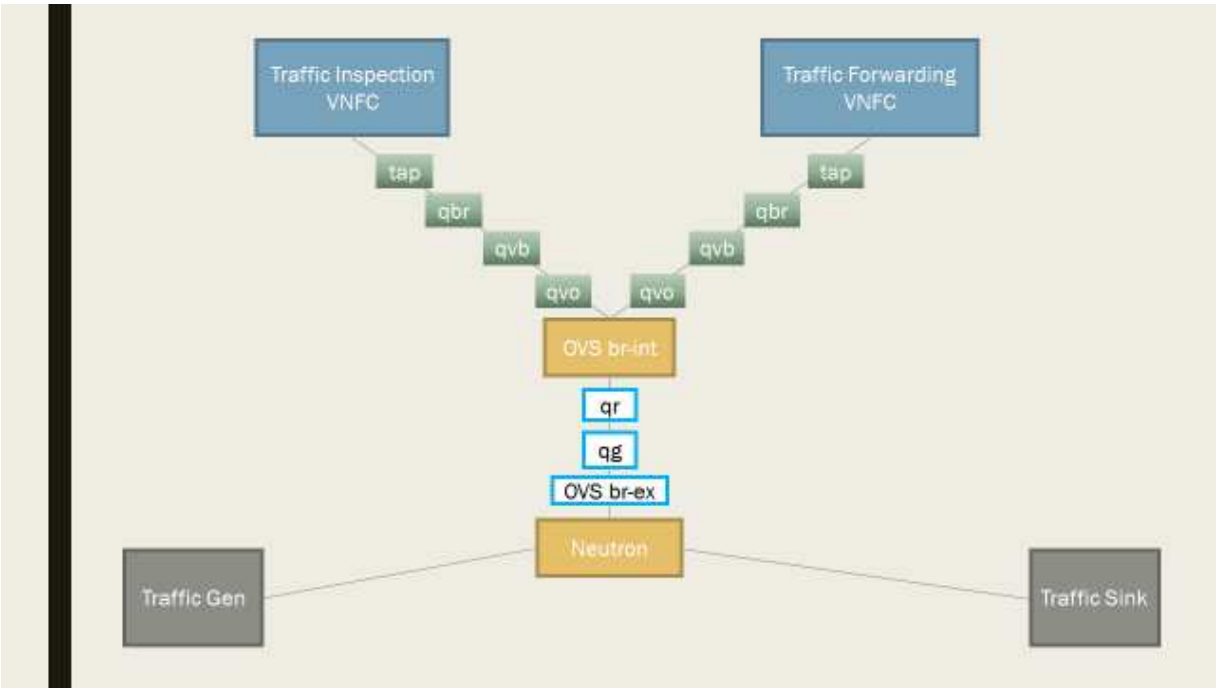


Figure 12. Example overview of the vTC OpenStack network topology.

First Option unbind interfaces from the Openstack networking and connect them directly via OVS

```
 * Remove from br-ex, the qvo virtual interfaces
 * Remove from the qbr linux bridge, the qvb and the tap virtual
interfaces
 * Add the tap-interfaces on the OVS directly and add a flow
forwarding the traffic to them.
```

This option has been tested and as shown in the results section for the cases of normal network setup.

3.4.7.2. Traffic Mirroring – SR-IOV

Single Root I/O virtualization (SR-IOV) in networking is a very useful and strong feature for virtualized network deployments. SRIOV is a specification that allows a PCI device, for example a NIC or a Graphic Card, to share access to its resources among various PCI hardware functions:

Physical Function (PF) (meaning the real physical device), from it a number of one or more Virtual Functions (VF) are generated. Supposedly we have one NIC and we want to share its resources among various Virtual Machines, or in terms of NFV various VNFCs of a VNF. We can split the PF into numerous VFs and distribute each one to a different VM. The routing and forwarding of the packets is done through L2 routing where the packets are forwarded to the matching MAC VF. In order to perform our mirroring and send all traffic both ways we need to change the MAC address both on the VM and on the VF and disable the spoof check.

3.4.8. Future Steps

Future steps include the implementation of automated way to apply the direct connection of the VNFCs. This step will be included in a HEAT deployment

- Benchmarking all options and comparing them.

Other options to be tested, is to add a TCP/IP stack on the DPDK and maintain the connectivity of the VNFCs. These alternatives include:

- A kernel with tcp/ip stack on the userspace DPDK rump kernel – https://github.com/rumpkernel/drv-netif-dpdk
- DPDK FreeBSD TCP/IP Stack porting https://github.com/opendp/dpdk-odp

3.5. Virtual CDN / Virtual Home Gateway (VIO)

3.5.1. vHG

Another VNF that T-NOVA aims to produce is currently known in the research and the industry world under various names, notably Virtual Home Gateway (VHG), Virtual Residential Gateway, Virtual Set-Top Box or Virtual Customer Premise Equipment.

We will see how the initial need has been expanded to cover some aspects of the Content Delivery Network virtualization as well.

The following sections aim to provide a brief description of the proposed virtual function along with the requirements, the architecture design, functional description, and technology.

In T-NOVA, we will focus on the bottleneck points usually found in resource constrained physical gateway like media delivery, streaming and caching, media adaptation and context-awareness. In fact, some previous research proposals like [Nafaa2008] or [Chellouche2012] include the Home Gateways to assist the content distribution. By using a Peer-to-Peer approach, the idea in those approaches is to offload the main networks and provide an "Assisted Content Delivery" by using a mix of Server Delivery and Peer delivery.

When virtualizing the Home Gateway, this approach can lead in some extent to the creation of a Virtual CDN or vCDN as a VNF.

Particular attention will be given to real world deployment issues, like coexistence with legacy hardware and infrastructure, compatibility with existing user premise equipment and security aspects.

3.5.2. vCDN

Content delivery networks (CDN) have been created to cope with the challenges encountered by of Content Providers to delivery huge amounts of static data through

best effort internet. Like traditional CDN operators and recently even Content Providers, ISP are interested in building these solution as they bring an interesting growth driver.

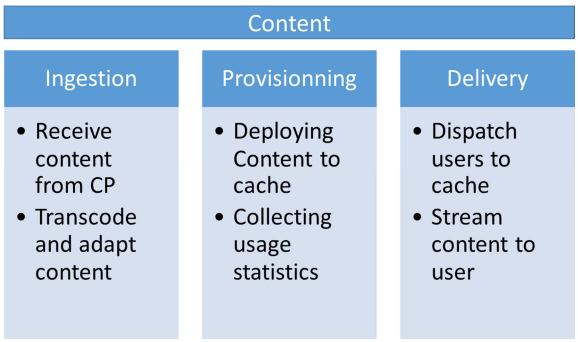


Figure 13 The three main components of a CDN System

In a traditional deployment show in Figure 13, CDN services are an aggregation of network function allowing the ingestion of the content, the provisioning in cache and finally the delivery to End Uses.

More precisely, ingestors carry out transcoding operation allowing to decrease the size of the video as well as re-segmentation to optimize diffusion over IP. Provisioning modules deploy ingested content in caches, taking into account the local popularity of content. Finally the delivery modules perform End User – Server assignation and content delivery.

Also described by ETSI as a virtualization use case, the vCDN complements our work on the VHG. Indeed, the VHG's routing function is used in a vCDN use case to make the user-server assignation easier and more fine-grained.

3.5.2.1. High Level

Our proposal is developed around 4 main modules:

Virtual Home Gateway: is a transit Network Function inspecting high-level HTTP traffic that can influence the IP routing decisions, based on the presence of the content in a nearby POP. Its configuration is provided by the caching orchestrator which has a complete vision on the system.

Content Streamers: we integrated a distributed object storage engine that provides resiliency, horizontal scalability and geographical redundancy amongst POPs.

Content Ingestors: are scalable workers that perform software transcoding to H264 and H265 video compression standards as well as re-segmentation of videos using both DASH and HLS technology to provide adaptive HTTP Streaming capability. Ingestors receive content from the CP Servers (push model) or can be automatically provisioned from the most popular contents (pull model). Ingestor have been demonstrated being able to rely on hardware accelerators (Virtual Transcoding Units) for computer intensive tasks when available.

Caching Orchestrator: is the module is charge of controlling the ingestion (by scheduling the job of the workers), the provisionning (by selecting which content is

cached in which streamer) and the delivery (by deploying configuration on the VHG) of content

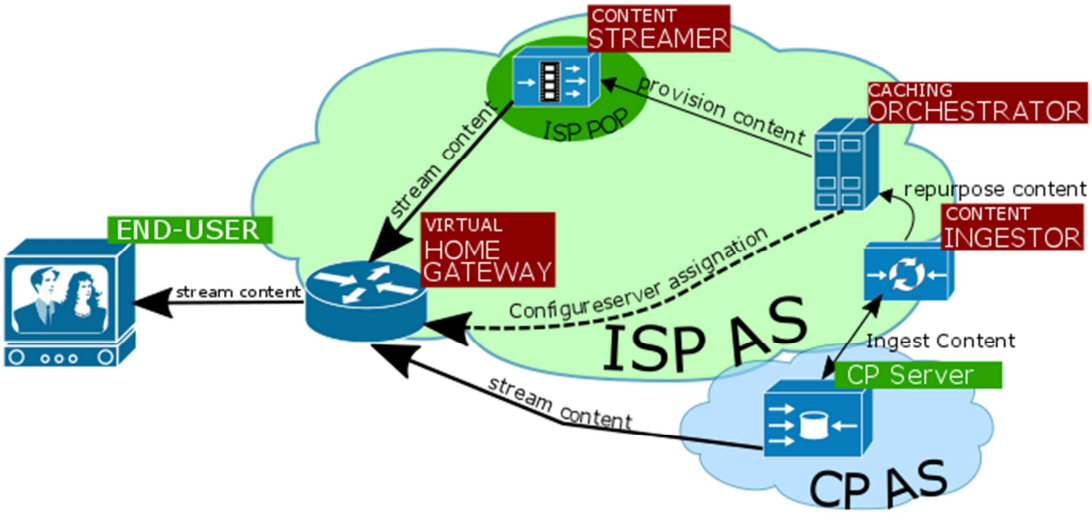


Figure 14 HIgh level architecture diagram for transcode/stream VNF

Figure 14 shows a Virtual Home Gateway which acts as an HTTP proxy, notifying the content fronted when a video is consumed by the end user. Having this information allows the content frontend to trigger the download from the content provider's network to the VNF. Once the video is entered on the VNF, it is transcoded and moved to a streamers.

Once the video resource is available to the end user, the gateway routes the user's request to the streamer. As the streamer is located within the ISP Network near the end user, hops and latency are limited, which increases QoE.

3.5.3. Sequence diagrams

The sequence diagram presented in Figure 15 is associated with the level architecture presented in the previous section.

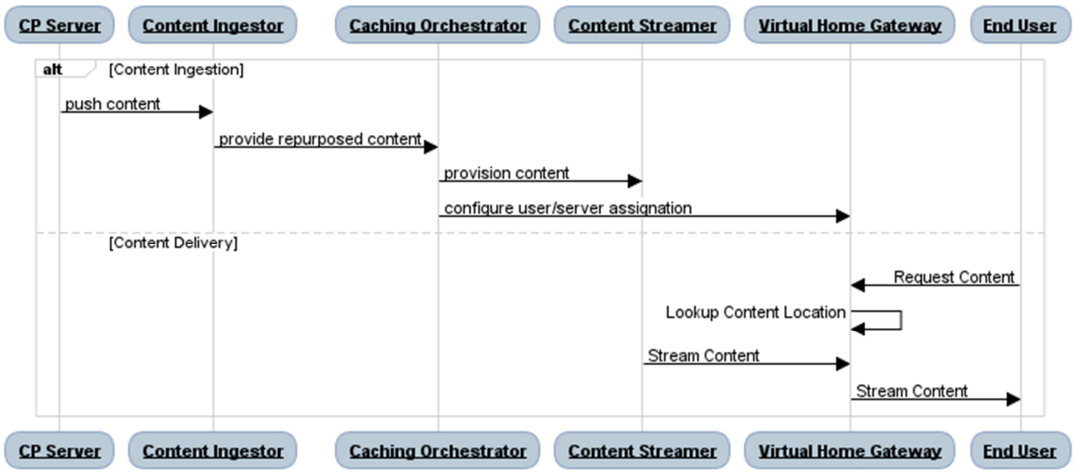


Figure 15 Sequence diagram for the transcode/stream VNF example.

3.5.4. Technology

3.5.4.1. Netty: a Java Non-Blocking Network Framework

Netty is *an asynchronous event-driven network application framework* [Netty] for rapid development of maintainable high performance protocol servers and clients.

One of the most striking features of Netty is that it can access resources in a non-blocking approach, meaning that some data is available as soon as it gets in the program. This avoids wasting system resources while waiting for the content to become available; instead a callback is triggered whenever data is available. This also saves system resources by having only 1 thread for resource monitoring.

Netty is one of the building blocks used to implement the vHG network capabilities.

3.5.4.2. Restful architecture

End user applications, Gateways and Front-end need to interact though secured connection on the internet.

A Java Restful architecture can be implemented for those reasons:

- Architecture is stateless, which means that the servers that expose their resources do not need to store any session for the client. This greatly eases scaling up, since no real time session replication needs to be performed, therefore a new server will be deployed for load balancing purposes.
- Architecture is standard and well supported by the industry, allowing us to leverage tools for service discovery and reconfiguration.
- Authentication methods are well documented and widespread among web browsers and servers.

Regarding the technical details, we will consider the standards of the Java SDK, by using JAX-RS and its reference implementation, Jersey. This framework can be

integrated on any servlet container, JEE container or lightweight NIO HTTP server like Grizzly which is used on the vHG.

3.5.4.3. Transcoding workers

One of the key features of cloud computing is its ability to produce on-demand compute power at a small cost. To take advantage of this feature, we plan to implement the most computing intensive tasks as a network of workers using a Python framework called Celery. Celery is an asynchronous task queue/job queue based on distributed message passing.

Every Celery worker is a stand-alone application being able to perform one or more tasks in a parallelized manner. To achieve this goal, a general transcoding workflow has been designed to be applied on a remote video file.

Having a network of workers allows us to scale-up or scale-down the overall compute power simply by turning a virtual machine up or down. Once the worker is up, it connects to the message broker, and picks up the first task available on the queue. Frequent feedback messages are pushed to the message broker, allowing us to present the results on the gateway as soon as they are available on the storage.

If the compute capacity is above the required level, active workers are decommissioned, leaving the pool as their host virtual machine turns off.

Note that workers only carry out software transcoding, leaving room for optimization through the use of hardware. The virtual Transcoding Unit (vTU) is an excellent drop-in replacement for the transcoding vNF. However, as hardware transcoding may not be available everywhere, we keep the slow software transcoding as a fall-back option.

3.5.4.4. Scalable Storage

We need to have caches able to store the massive amount of data needed by a CDN. These caches can be spread among several datacentres and must be tolerant to failure. They also need to scale, and must support adding or removing storage node as defined by the scaling policy.

To implement that, we decided to deploy [Swiftstack] which proposes to create a cluster of storage node to support Scalable Object storage with High availability, Partition Tolerance and eventual consistency.

Storage Nodes are accessed by external users using a Swift Proxy that handles the read and write operations. Swift has abstractions where nodes are stored inside zones and regions. We detail the mapping between swift abstraction and T-NOVA in **Erreur ! Source du renvoi introuvable.**

Swift	T-NOVA	Meaning
Region	NFVI-POP	Parts of the cluster that are physically separated

Zone	Compute Node	Zones to be configured to isolate failure
Node	VNFC	Server that run one or more swift process

Table 3-A Mapping between Swift and T-NOVA abstraction

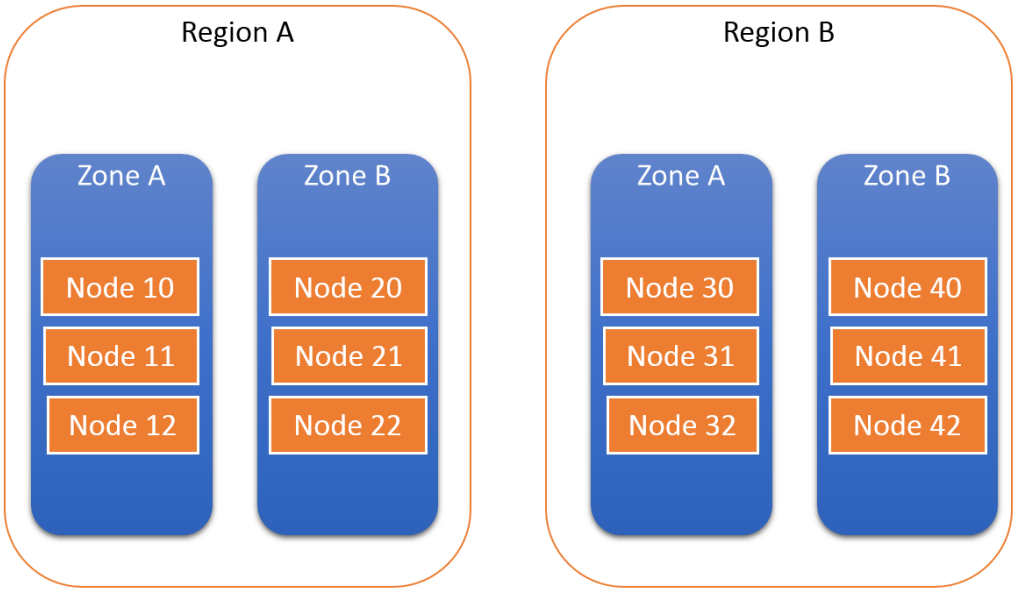


Figure 16 swift stack Region/Zone/Node.

We use swift abstraction to provide a reliable storage solution. For example, our vCDN spans over multiple datacentres to provide good connectivity. Each pop is associated to a region. With the same approach, we can have several compute nodes hosting our VNFC. For reliability reasons, we don't want all our nodes hosted on the same compute node, so that if the compute node goes down, part of the service will be still available. Finally, each VNFC hosts a swift Node.

Even if swift is an object storage, it allows users to access and push data over a standard HTTP API. It means that the streamer vNF feature can be implemented using swift as well.

3.5.4.5. Using Docker to provide safe, reliable and powerful application deployment

We decided to use [Docker] to support the implementation. Docker is an OS Virtualization technology that runs segregated applications and libraries on a common Linux kernel.

Docker can be run on major Linux Distribution like Debian or Fedora, but it can also run on smaller, custom distribution that provide an execution environment for container. CoreOS produces, maintains and utilizes open source software for Linux

containers and distributed systems. Projects are designed to be composable and complementing each other in order to run container-ready infrastructure.[2]

The applications we build are based on vendor technologies (for example, the Java Docker image maintained by Oracle) that are kept updated on a regular basis. We implemented continuous deployment, meaning that whenever an upstream dependency gets updated, we re-package our software with the new image and run test to discover potential regression.

Our approach is safer. The traditional installation of a package on an OS since every container is walled from the other ones and the OS has the only responsibility of maintaining the container execution environment. Vendors usually provide a shorter delay to update their Docker images that the Linux Distribution.

Our approach is reliable in the sense that if a T-NOVA virtual machine goes down (except the VNF Controller which is not highly available for the moment) we are able to redeploy containers on the cluster on another available machine.

We also don't have to upload a new vnfd + vnf images every time we have a security update. All we need to do is to push the new release on our Docker registry and the new image will be picked up automatically when configuring the VMs.

3.5.4.6. Orchestration and scaling

In order to ease the deployment of our vNFs, we use a configuration management tool named Salt Stack [Salt]. The necessity to use such a tool is developed in the next paragraphs; we then explain why we choose salt and finally conclude with an overview of the mechanisms we implemented.

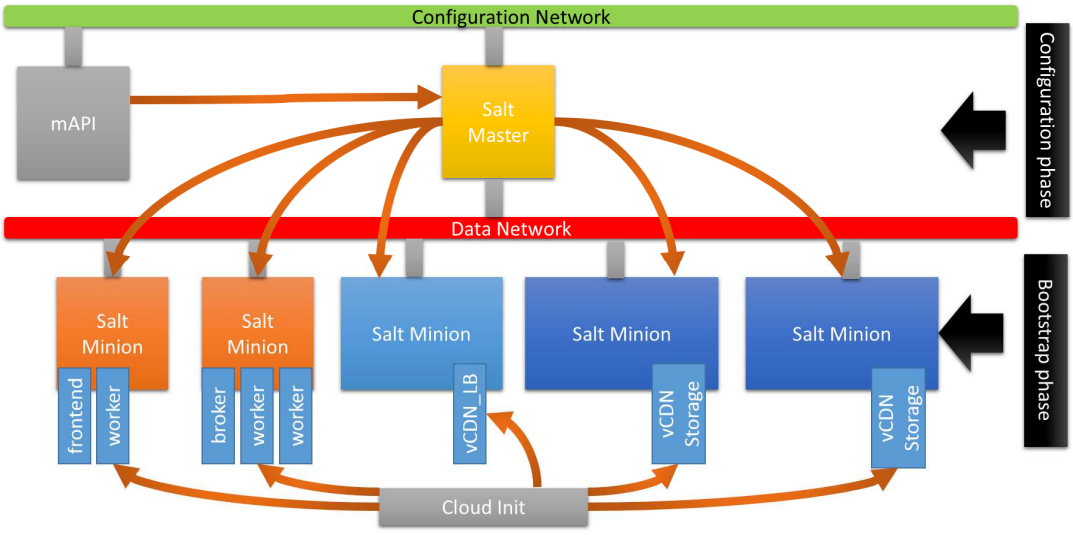


Figure 17 Software configuration Management for vHG+vCDN

[2] https://coreos.com/docs/

3.5.4.7. Why configuration management tool?

As mentioned in the VNF Controller section of Deliverable D5.31, only one VNFC is able to receive configuration commands from the Orchestration to support the whole VNF life cycle. This means that the information received on the configuration interface must be propagated to the other VNFCs.

When the VNF starts, some configuration need to be carried out to initialize the software components. For example, the storage nodes must be initialized with the DHT from the proxy, some block storage must be allocated to the node and so one. This non trivial configuration tasks must be carried out after the VM has booted, but also when scaling out or in. These tasks may fail, but the consistency of the whole system should be kept intact.

For those reasons, we decided to use an orchestration tool that create an abstraction level over the system to manage the software deployment, system configuration, middleware installation and service configuration with ease.

3.5.4.8. Why Salt?

SaltStack platform or Salt is a Python-based open source configuration management software and remote execution engine. Supporting the "infrastructure-as-code" approach to deployment and cloud management, it competes primarily with Puppet, Chef, and Ansible.[3]

Salt Stack was preferred over other alternatives due to its scalability, ease of deployment, good support for Docker and python source code. We don't claim that what we designed would not have been possible with other alternative, but Salt was the solution we felt the more comfortable with at the end.

3.5.4.9. Implementation of our configuration management

We implemented the configuration management as a two-phase process. It is illustrated in Figure 17.

First during the bootstrap phase, each virtual machine is injected with cloud-init with the following data and programs.

- IPaddress of the salt master
- Certificates to assure a secure connection with the salt master
- Its role in the system.
- Salt-master or salt-minion service installed and launched.
- The "recipes" or desired infrastructure code deployed on the salt master.

Once the bootstrapping phase is over, we have a system comprised of VMs securely connected on the data network ready to take order from the master. Note that the OS could be pre-bundled with software in order to fasten the next phase, but this is not mandatory.

[3] https://en.wikipedia.org/wiki/Salt_%28software%29

The second phase is launched when the start lifecycle event from TeNOR is received through the middleware API. This processes the infrastructure code and verifies the compliance of each minion with the desired infrastructure.

As we can see in **Erreur ! Source du renvoi introuvable.**, the yaml DSL used with Salt describes how the infrastructure should be configured. Salt allows us to "synchronize" the code infrastructure in yaml with the real infrastructure simply by calling the Salt API. This synchronization process installs, copies, configures, downloads the required missing software components and can even configure more low level aspects.

Providing the possibility for the system the scale-in is straightforward when having the infrastructure described as code. Installing, configuring and ramping up new VM is just a matter of "synching" the infrastructure state with the new resources available.

Our implementation use the Debian Jessie for applications and containers.

```
#here we make sure that the latest worker docker image is present on the system
nherbaut/worker:
  #this command is equivalent to docker pull
  docker.pulled:
            #always use the latest version from our continus build system
    - tag: latest
    - require:
              #make sure that docker is installed before pulling the image
      - sls: docker
              #make sure that docker daemon is running
      - service.running: docker

# this set of jinja2 template file is here to provide the broker's IP address
{%- set minealias = salt['pillar.get']('hostsfile:alias', 'network.ip_addrs')
%}
{%- set addrs = salt['mine.get']('roles:broker', minealias,"grain") %}
{%- set broker_ip= addrs.items()[0][1][0] %}

# this set of instruction is there to provide the the swift proxy ip address
{%- set addrs = salt['mine.get']('roles:swift_proxy', minealias,"grain") %}
{%- set swift_proxy_ip= addrs.items()[0][1][0] %}


# now we are ready to cook our docker image
core-worker-container:
  docker.installed:
    - name: core-worker-container
    - image: nherbaut/worker:latest
     # now we are ready to cook our docker image
    - environment:
      - "CELERY_BROKER_URL" : "amqp://guest@{{ broker_ip }}"
      - "ST_AUTH" : "http://{{ swift_proxy_ip }}:8080/auth/v1.0"
      - "ST_USER" : "admin:admin"
      - "ST_KEY" : "admin"
    - watch:
      # trigger this event whenever the image is done being pulled
      - docker: nherbaut/worker
```

Code listing 1 an example of infrastructure code

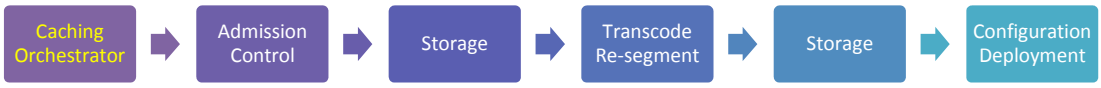
3.5.5. Dimensioning and Performances

For testing purposes, the vHG and vCDN can be seen as a chain of microservices working together to implement the function. Having several components interacting together complexifies the task of characterizing the bottlenecks of the solution. We also need to take into account the fact that absolute performance is not really meaningful for scalable applications, since adding additional resources increase the processing capacity and the state of the cloud environment hosting the solution can vary over time along with the performances.

We carried out our experiments in a full-fledged NFV Infrastructure deployed within the T-NOVA project for a baseline configuration of 5 Virtual Machines. We only present high level performance results corresponding the 2 end-to-end scenarios: Ingestion-Provisionning and Delivery.

3.5.5.1. Testing vCDN Ingestion-Provisionning

Caching Orchestrator



The first element that we need to test is the caching orchestrator. It receives requests from the VHG and from the CP to create message for Admission Control.

We can see from Figure 18 that we didn't manage to saturate this module, even at 40 connections per minute, meaning that the admission control module will handle the request in real time.

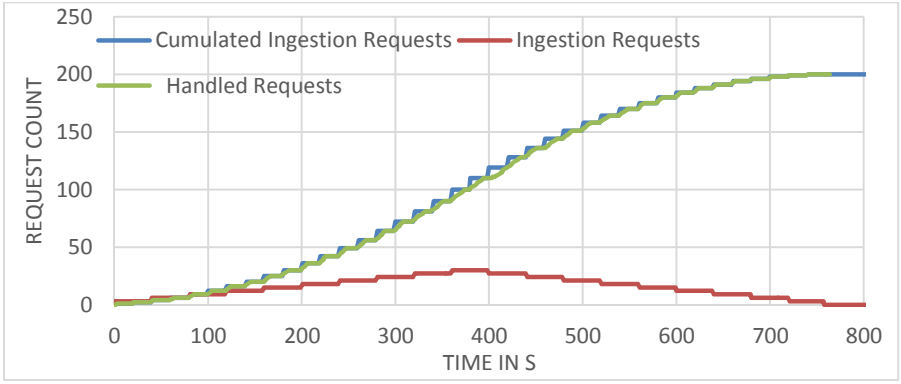
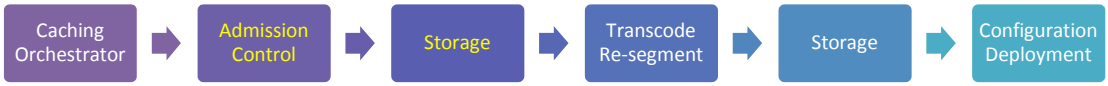


Figure 18 Caching Orchestrator performances

Admission Control



Here the admission control download the video and analyze it before sending message the transcoder and resegmenter to treatments. We can see from **Erreur ! Source du renvoi introuvable.** that the module saturate at around 28 videos per minutes.

For test purposes, we used a 6.6 MB video corresponding to 10s of playback.

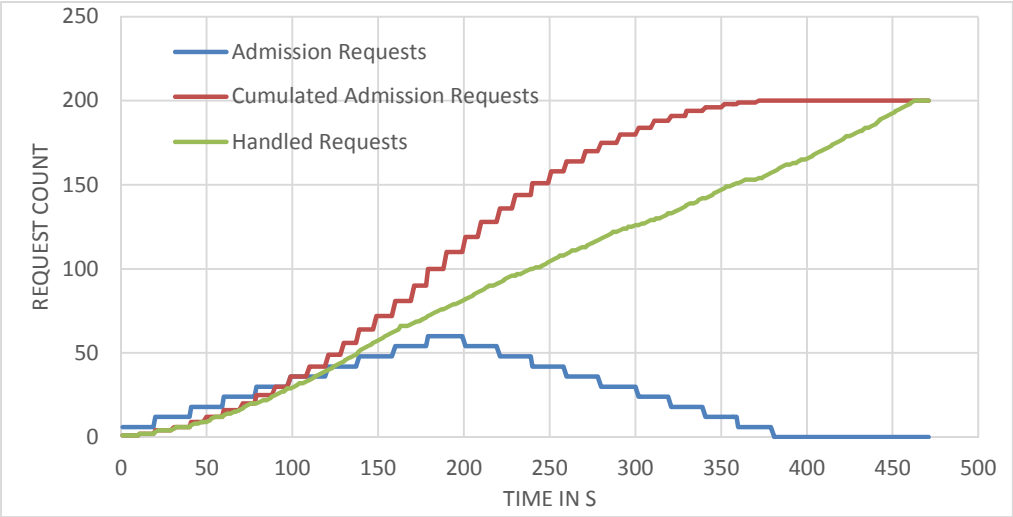
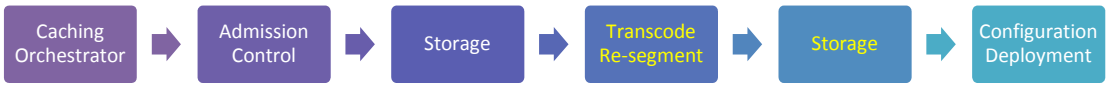


Figure 19 Admission Control performances

Transcode and Re-Segment



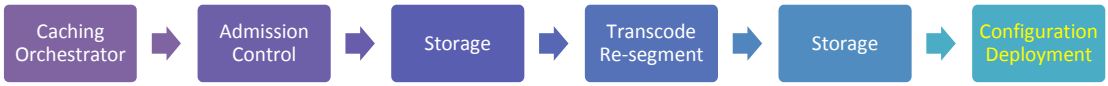
For the vCDN, ingestion means deploying the original content in the object store, analyzing this content, deciding which the optimal format for the content is and producing the adapted content. It is a very CPU and memory intensive task that can be easily scaled with the adjunction of a "worker" VM. Figure 19 shows a setting where we let the system ingest 200 videos of 20 MB at an average arrival rate of 30 videos per minutes. We compare the number of "pending" video jobs that are queued by the system for several settings. We scaled our VNF out and allowed the number of ingestion VM to vary.

We can see that the configuration with only 1 VM doesn't cope with the load, as it accumulates more than 120 pending videos and it depletes its video stock in more than 900 s. On the contrary, the 3 VM setting manage to finish nearly on time (420s).

Thanks to this design, the number of ingestion-Provisionning VM can be adjusted based on the characteristics of the videos, and on the tolerance to delay of the customer.

Figure 20 Transcode and Re-Segment Performance

Configuration Deployment



The User-Server POP assignment configuration is fetched by the VHG periodically. The Caching orchestrator offers a read API for "configuration deployment" that has the following performances for 1000 content.

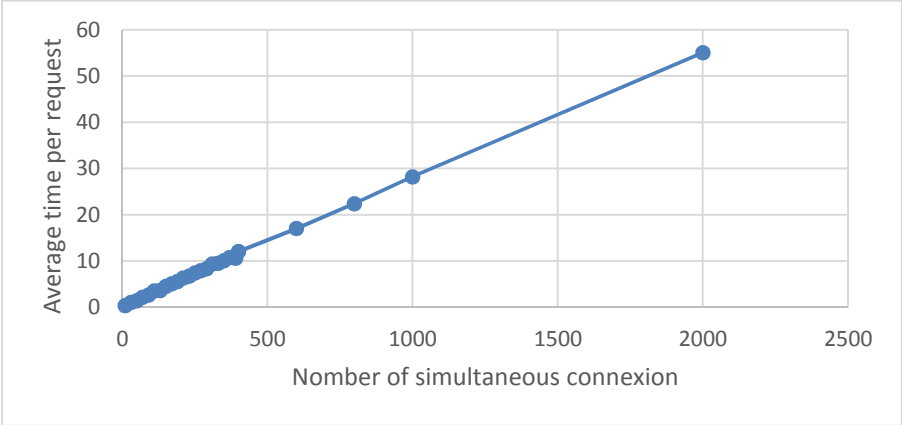
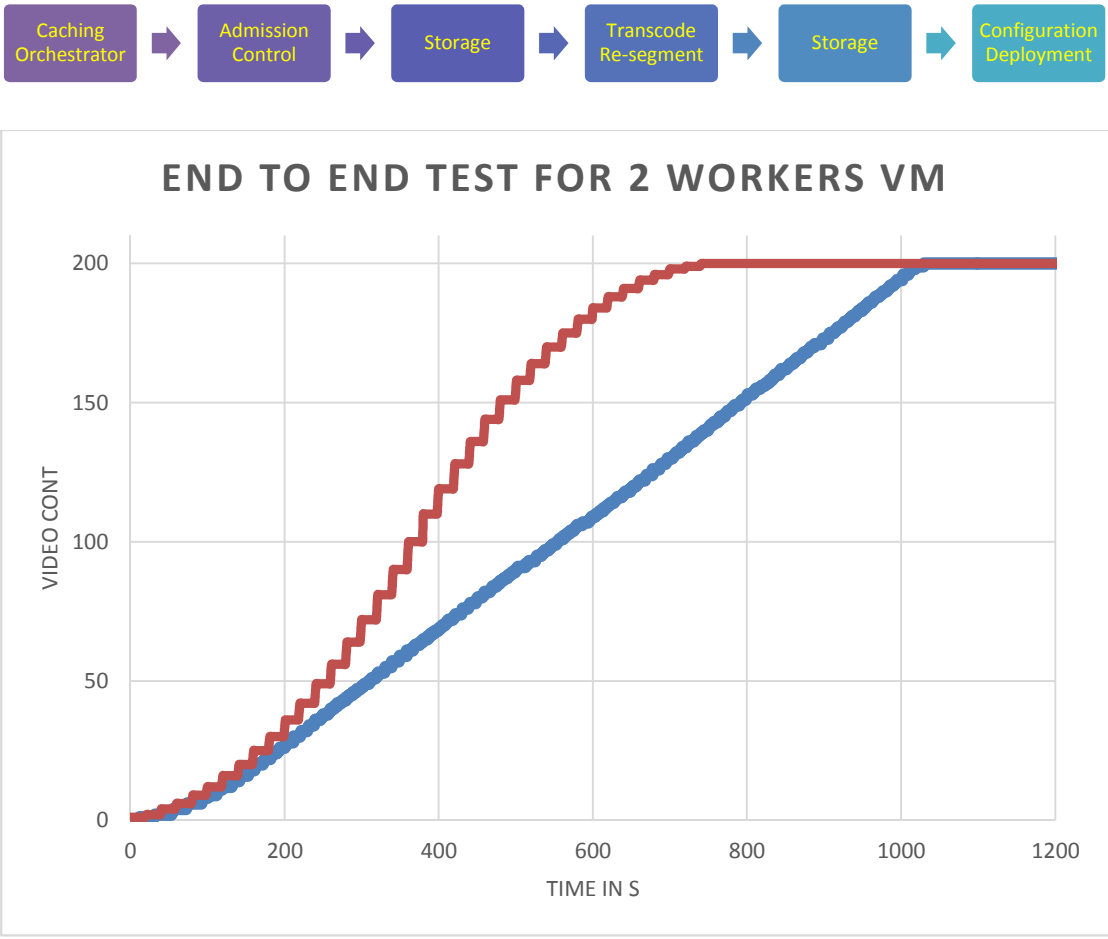


Figure 21 Configuration Deployment

From this figure we can see that the module can achieve 350 simultaneous query in less than 10s per query. Performances could be enhanced by using a caching mechanism instead of pure database access.

End to End Test



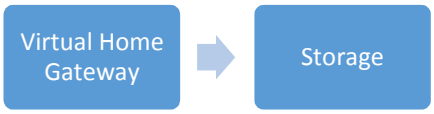
This end to end test has been carried out 2 worker VMS. We launched the ingestion of 200 videos (of 6.6MB) on 2 worker nodes and we pushed the results on 2 storage nodes.

With those settings, the system can absorb up to 12 video/minutes. Increasing the number of worker or using hardware acceleration could dramatically increase performances.

3.5.5.2. Testing Delivery with the vHG

The other side of the vCDN/vHG vNF is the actual delivery of the content to the end users. In this section we present the corresponding results.

End to End test



The overall performance of the delivery part of the vHG/vCDN depends in a large extent on the network performances between the object storage nodes. Indeed, each content is chunked and spread on several nodes to provide redundancy and increase performance. Furthermore the Virtual Home Gateway is used to inspect HTTP Packets, which may also cause delay and reduced throughput.

In Figure 18 we used apache2's ab tool to compute the 95 percentile maximum time taken to download a 10s, 6 MB video file encoded as 600 KBps. We increased the number of concurrent connection to establish the threshold above which the video cannot be stream at its nominal bitrate for the 5VM baseline configuration.

We can see two important results from the graph, first of all, there's no significant difference between the performance of Storage with or without the VHG. It means that the storage is the bottleneck in this case, and the VHG need not to be scaled-up to increase performances. Next, the video can be stream by 250 simultaneous users. This value is strongly correlated to the underlying state of the network on our infrastructure and also on the storage technology used in the platform. For example, our object storage engine is designed to use SSD disks to boost the delivery of the most used files. This feature wasn't available on our infrastructure, and could have dramatically increased performances, especially for internet content where only a small number of items is popular while the rest remain unknown.

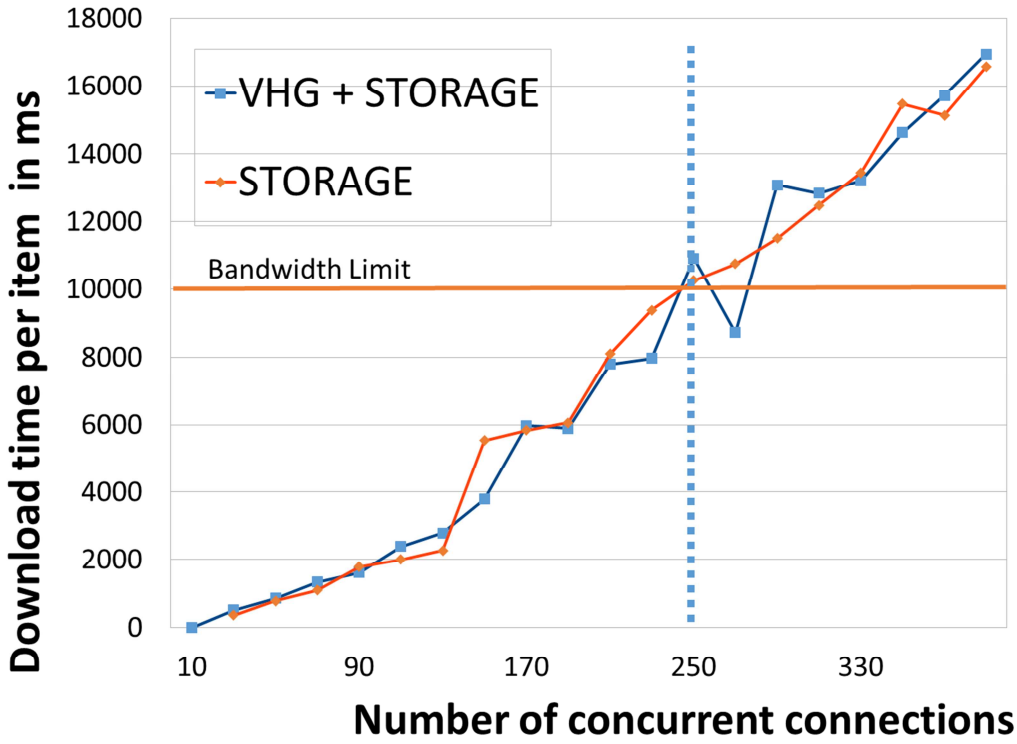


Figure 22 End to end Results for the vHG / vCDN delivery

3.5.6. Future Work

The next step for vHG+vCDN in WP5 is to Integrate the scalability feature of T-NOVA to automatically create/destroy new VDU as demand vary. We also plane to fine tune performances accordingly.

3.6. ProXy as a Service VNF (PXaaS)

3.6.1. Introduction

A Proxy server is a middleware between clients and servers. It handles requests, such as connecting to a website or service and fetching a file, sent from a client to a server. In the most cases a proxy acts as a web proxy allowing or restricting access to content on the World Wide Web. In addition, it allows clients to surf the Web anonymously by changing their IP address to the Proxy's IP address.

A proxy server can protect a network by filtering traffic. For instance, a company's policies require that its employees are restricted to access some specific web sites, such as Facebook, during working hours but they are allowed to access them during break times or are restricted to access adult-content sites at all times. Furthermore, a proxy server can improve response times by caching frequently used web content and introduce bandwidth limitations to a group of users or individuals. Traditionally, proxy software resides inside users' LANs (behind NAT or Gateway). It is deployed on a physical machine and all local devices can connect to the Internet through the proxy by changing their browser's settings accordingly. However, a device can bypass the proxy. A stronger alternative deployment is to configure the proxy to act as a transparent proxy server so that all web requests are forced to go through the proxy. In this scenario the gateway/router should be configured to forward all web requests to the proxy server.

The Proxy as a Service VNF (PXaaS VNF) aims to provide proxy services on demand to a Service Provider's subscribers (either home users e.g. ADSL subscribers or corporate users such as company subscribers). The idea behind the PXaaS VNF is to move the proxy from the LAN to the cloud in order to be used "as a service". Therefore, a subscriber (e.g. LAN administrator) will be able to configure the proxy from a web-based user friendly dashboard and according to their needs so that it can be applied to the devices within the LAN.

3.6.2. Requirements

The table below provides the major requirements that the VNF will need to fulfill.

Table 3-B: PxaaS VNF requirements

Requirement ID	Requirement name	Description	Priority level
1	Web caching	The PXaaS VNF should be able to	High

		cache web content.	
2	User anonymity	The PXaaS VNF should allow for hiding the user's IP address when accessing web pages. The proxy VNF's IP should be shown instead of the user's real IP.	High
3	Bandwidth rate limitation per user	The PXaaS VNF should allow for setting bandwidth rate limitations on a group of users or individual users by creating ACLs based on their account.	High
4	Bandwidth rate limitation per service	The PXaaS VNF should allow for setting bandwidth rate limitations on a group of services or individual services. For example, the PXaaS VNF should limit the bandwidth used for torrents.	Low
5	Bandwidth throttling on huge downloads	The PXaaS VNF should allow for reducing the bandwidth rate when huge downloads are detected. It could be applied to all users or a group of users or individuals.	High
6	Web access control	The PXaaS VNF should allow for blocking specific websites by the users.	High
7	Web access control (time)	The PXaaS VNF should allow for blocking or accessing specific websites by the users based on the current time.	Medium
8	User Control and Management	The user should be able to configure the PXaaS VNF using a dashboard. The dashboard should be responsive in order to be accessible from multiple devices and easy to use.	High
9	Service availability	The Proxy VNF should be available as soon as the user sets the configuration parameters on the dashboard. Each time a user changes configuration, the service should be available immediately.	High
10	Service accessibly	The connection with the proxy should be transparent (transparent proxy). Users do not need to set the proxy's	Low

		IP on their browser. The traffic should be redirected from the user's LAN to the proxy VNF.	
11	Service – user authentication	Only subscribed PXaaS VNF users should be able to access the service.	High
12	Monitoring	The proxy VNF should provide metrics to the T-NOVA's monitoring agent.	High
13	Service provisioning	The proxy VNF should expose an API to be used by the T-NOVA's middleware for service provisioning.	High

3.6.3. Architecture

The PXaaS VNF consists of one VNFC. The VNFC implements both the proxy server software as well as the web server software. The figure below provides a high level topology of the PXaaS VNF. The VNFC is located at the PoP which is found between the user's LAN and the Operator's backbone. Once a user is subscribed with the PXaaS VNF the traffic from the user's LAN is redirected to the PoP and then it passes through the PXaaS VNF. The traffic might pass through some other VNFs according to service function chaining policies. Finally, the proxy handles the requests accordingly and forwards the traffic to the Internet. The user is able to configure the proxy through an easy to use web-based dashboard which is served by the web server. The web server communicates with the proxy server in order to set up the configuration parameters which have been defined by the user.

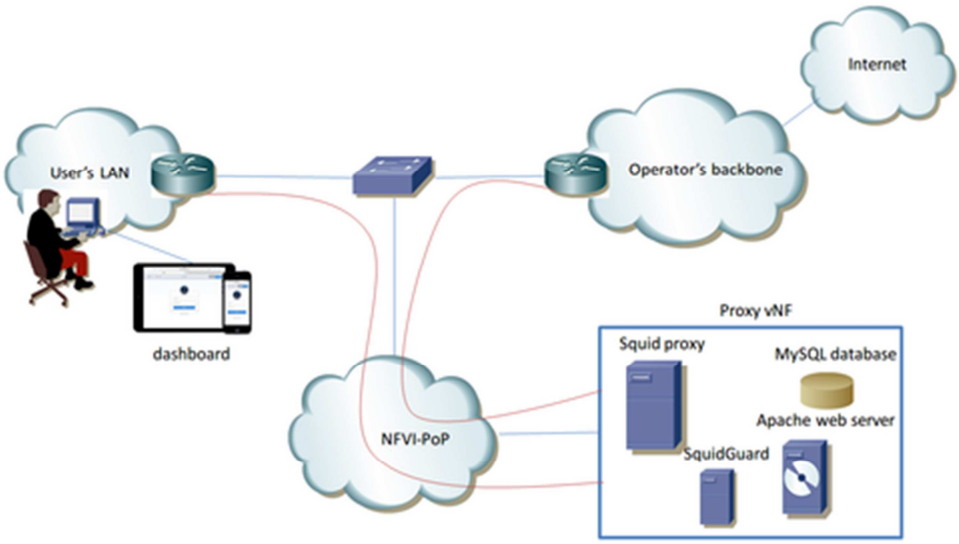


Figure 23. PXaaS high level architecture

3.6.4. Functional description

3.6.4.1. Squid Proxy server

Squid Proxy is a caching and a web proxy. Some of its major features include:

- Web caching;
- Anonymous Internet access;
- Bandwidth control. It introduces bandwidth rate limitations or throttling to a group of users or individuals. For example it allows "normal users" to share some amount of traffic and on the other hand it allows "admin users" to use a dedicated amount of traffic;
- Web access restrictions e.g. allow a company's employees to access Facebook during lunch time only and deny access to some specific web sites.

Bandwidth limitation examples

a) Bandwidth restrictions based on IP

The example below creates an Access Control List (ACL) with the name "regular_users" and is assigned a range of IP addresses. Requests coming from those IPs are restricted to 500KBps bandwidth.

```
acl regular_users src 192.168.1.10 – 192.168.1.20/32 # acl list based
on IPs
delay_pools 1
delay _class 1 1
delay_parameters 1 500000/500000 # 500KBps
delay_access 1 allow regular_users
```

The limitation of this configuration is that Squid should be located inside the LAN in order to understand the private IP address space.

b) Bandwidth restrictions based on user

The following scenario performs the same bandwidth restrictions as the previous one except that the ACL is based on user accounts. Squid supports various authentication mechanisms such as LDAP, Radius and MySQL database. We consider MySQL database for authenticating with the PXaaS VNF.

```
acl regular_users proxy_auth george savvas # acl list based on
usernames
delay_pools 1
delay _class 1 1
delay_parameters 1 500000/500000 # 500KBps
delay_access 1 allow regular_users
```

The limitation of this configuration is that users must authenticate with the Proxy the first time they visit their browser. In this case the proxy is not considered as a transparent proxy. However, by using this scenario, Squid can be deployed on the cloud and can handle devices behind NAT as long as they authenticate with the proxy.

Figure 24. Proxy authentication

3.6.4.2. Apache Web Server

Apache web server is used to serve the dashboard to the clients. The dashboard is responsible to allow users to configure and manage the Squid proxy. Therefore, Apache should have write permissions on Squid's configuration file. In addition, the LAN administrator is able to create user accounts which are stored in the MySQL database. The LAN administrator will be responsible to assign the user accounts to each device in order to achieve the limitations he envisions using the PXaaS.

The figure below presents the first version of the dashboard (version 1). In particular, the home page of the dashboard is presented. The current version supports the following features:

- **User management:** User accounts can be created with a username and password. Those accounts are used to access the proxy services;
- **Access control:** Users must enter their credentials in their browsers in order to surf the web;
- **Bandwidth limitations:** Group of users can be created with a shared amount of bandwidth. In this case bandwidth limitations can be introduced to a group of users;
- **Website filtering:** Group of users can be created with restricted access to a list of websites. Pre-defined lists with urls are provided;
- **Web caching:** Web caching can be enabled in order to cache web content and improve response time;
- **User Anonymity:** Users can surf the web anonymously.

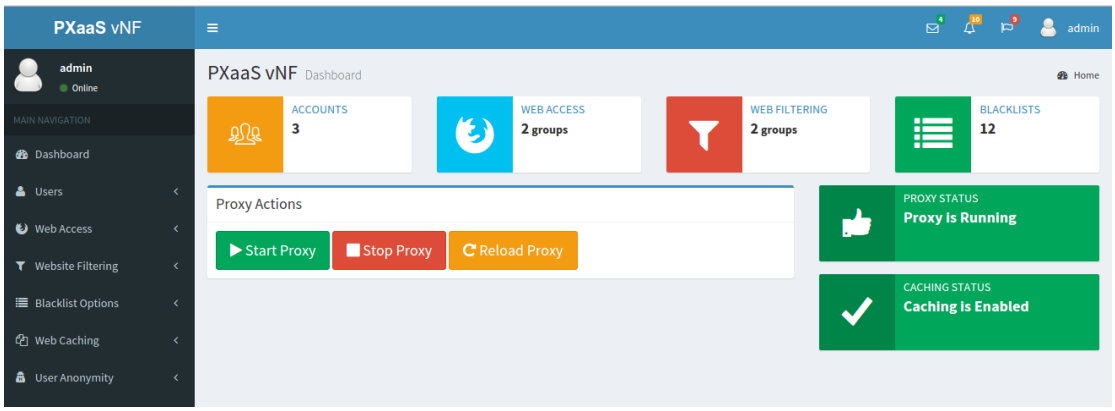


Figure 25 - PXaaS Dashboard

3.6.4.3. MySQL Database server

MySQL Database server maintains a list of user accounts that can be used for proxy authentication in the browser. In addition it stores all the required data needed by the dashboard.

3.6.4.4. SquidGuard

SquidGuard is used on top of Squid in order to block URLs for a group of users. It is used based on pre-defined black lists.

3.6.4.5. Monitoring Agent

The Monitoring Agent is responsible for collecting and sending monitoring metrics to the T-NOVA Monitoring component.

3.6.5. Interfaces

The figure below shows the VNFC in an OpenStack environment. It consists of 3 interfaces connected to 3 networks.

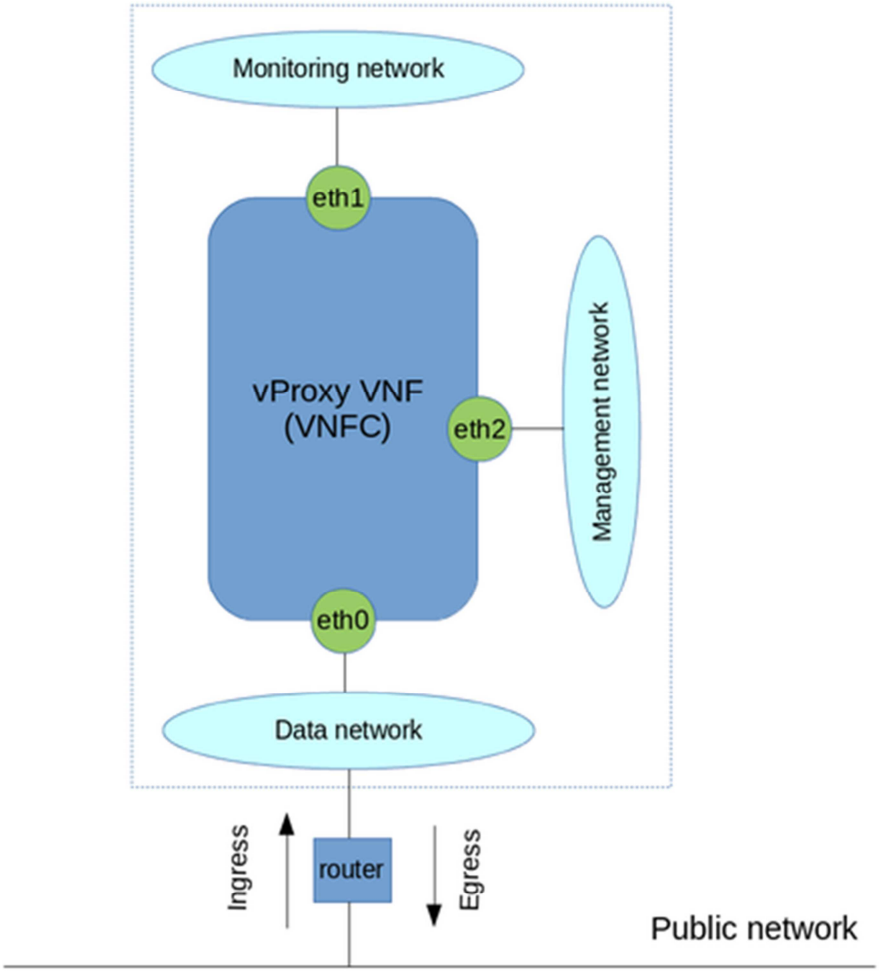


Figure 26. PXaaS in OpenStack

- **eth0**: This is the data interface. A floating IP is associated with this interface in order to send and receive data to/from the Public network.
- **eth1:** This is the monitoring interface which will be used to send metrics periodically to the Monitoring component.
- **eth2:** This is the management interface which will be used in order to communicate with the middleware API.

3.6.6. Technologies

The development environment used for the implementation and testing of the PxaaS is Vagrant with Virtualbox on an Ubuntu 14.04 Desktop machine. The VM itself runs Ubuntu 14.04 server OS.

As described in the Functional description section, Squid Proxy, SquidGuard, Apache Web server and MySql Database server are used. Specifically, the exact versions are:

- Squid Proxy 3.5.5
- SquidGuard 1.5
- Apache2 2.4.7
- Mysql 5.5.44-0ubuntu0.14.04.1

The Dashboard has been developed with the Yii framework (a PHP framework) for the server side and CSS, HTML, Jquery have been used for the client side.

As regards the monitoring agent two different components have been used:

1. Collectd. It collects system performance statistics periodically such as CPU and memory utilization.
2. A python script which collects PxaaS VNF specific metrics such as the number of HTTP requests received by the proxy and the cache hits percentage. The script analyses the results received by the squidclient, a tool which provides Squid's statistics, and send them to the T-NOVA Monitoring component periodically.

Mozilla Firefox is used for accessing Web through the proxy.

3.6.7. Dimensioning and Performance

Some preliminary tests were performed in order to verify whether the expected behavior is achieved. We assume that access to the PXaaS Dashboard is given to a user who acts as the administrator of his LAN in a home scenario. Therefore, the "administrator" sets up the Proxy service for his LAN via the dashboard and creates user accounts in order to allow other users/devices to access the Web via the Proxy. Specifically, the current version of the Dashboard was tested against the following test scenarios:

a) **Testing web access and bandwidth control**. This scenario aims to test if a newly created user is able to access the Web using their credentials and bandwidth limitation is achieved.

Execution: The administrator creates a new user by providing a username and password. Then he adds the newly created user under "research" group (the group was previously created by the administrator) which is restricted to 512Kbps bandwidth. The new user authenticates with the proxy from the browser and downloads a big file.

b) **Testing web site filtering**. The scenario tests whether a user is restricted to access some websites.

Execution: The administrator adds the user to the group "social_networks" (the group was previously created by the administrator and a pre-defined list of social networking websites was assigned to that group) in which all social networking websites are denied.

c) **Testing web caching**. This scenario tests whether web caching works properly.

Execution: Two different users access the same websites from different computers. For example "user1" accesses www.primetel.com.cy and then "user2" accesses the same website.

d) **Testing user anonymity**. This scenario checks whether a user is able to access the Web anonymously. In order to test this scenario and get meaningful results we deployed the PXaaS VNF on a server with public IP.

Execution: The administrator enables the user anonymity feature for a user. http://ip.my-proxy.com/ website is used in order to check whether user's real IP is publicity visible.

3.6.7.1. Test results

Below the results by executing the test scenarios are presented.

a) Once a user is authenticated with the Proxy he is able to access the Web. Then he starts to download an iso file. As we can see from the image below, the download speed is restricted to 61,8 KB/sec which is around to 500 Kb/sec (as we have expected). If another user starts to download a big file as well, then both users will share the 512Kb/sec bandwidth.

Figure 27. Bandwidth limitation

b) A user tries to access www.facebook.com with no success. The proxy denies access to the particular website.

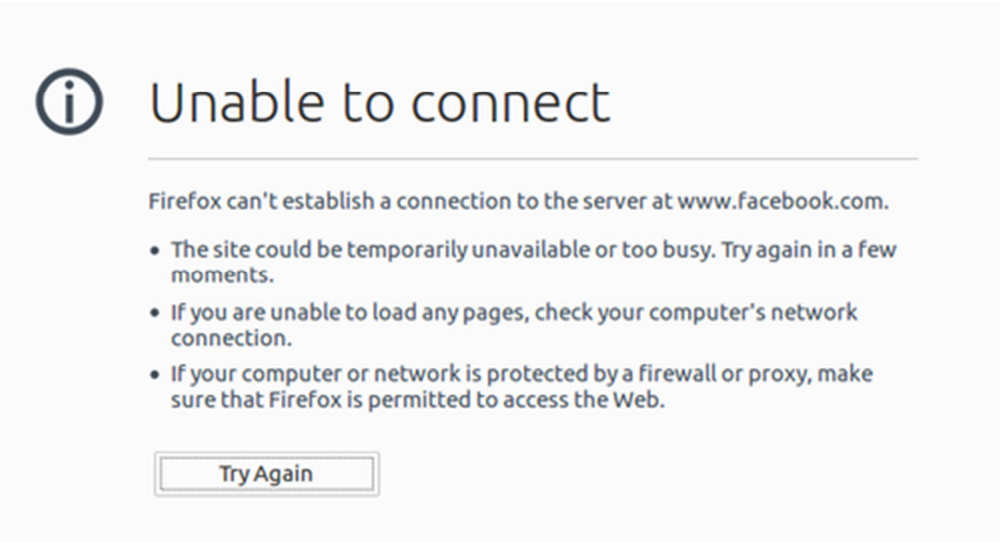


Figure 28. Access denied to www.facebook.com

c) The figure below shows the Squid's logs. In particular, it shows all the HTTP requests received by Squid from the clients and whether those requests result in cache hits. It can be observed that the particular requests were served from the Squid's cache. "TCP_MEM_HIT" shows that a request was served from Squid's Memory Cache (from the RAM).

```
01/Dec/2015:07:59:34 +0000     29 192.168.56.1 TCP_MEM_HIT/200 63401 GET http://primetel.com.cy/wp-co
ntent/uploads/2015/10/MiFi_1200x300_gr.png admin HIER_NONE/- image/png
01/Dec/2015:07:59:34 +0000     45 192.168.56.1 TCP_MEM_HIT/200 152076 GET http://primetel.com.cy/wp-c
ontent/uploads/2015/08/B2S_1120x300_grR.png admin HIER_NONE/- image/png
01/Dec/2015:07:59:34 +0000     53 192.168.56.1 TCP_MEM_HIT/200 210076 GET http://primetel.com.cy/wp-c
ontent/uploads/2015/10/Ninja2_2000x300_Gr.png admin HIER_NONE/- image/png
01/Dec/2015:07:59:34 +0000     58 192.168.56.1 TCP_MEM_HIT/200 294737 GET http://primetel.com.cy/wp-c
ontent/uploads/2015/11/Xmas_slider_2000x300_gr.png admin HIER_NONE/- image/png
01/Dec/2015:07:59:34 +0000     22 192.168.56.1 TCP_CLIENT_REFRESH_MISS/200 84134 GET http://primetel.
com.cy/wp-content/plugins/ubermenu/assets/css/fontawesome/fonts/fontawesome-webfont.woff? admin HIER_
DIRECT/primetel.com.cy text/plain
01/Dec/2015:07:59:34 +0000      2 192.168.56.1 TCP_MEM_HIT/200 17600 GET http://primetel.com.cy/wp-co
ntent/uploads/2015/11/OnlineOffer_228x136-A.png admin HIER_NONE/- image/png
01/Dec/2015:07:59:34 +0000      1 192.168.56.1 TCP_MEM_HIT/200 6323 GET http://primetel.com.cy/wp-con
tent/uploads/2015/09/228x136_red.png admin HIER_NONE/- image/png
01/Dec/2015:07:59:34 +0000    270 192.168.56.1 TCP_MISS/200 44495 GET http://static.hotjar.com/c/hotj
ar-68446.js? admin HIER_DIRECT/static.hotjar.com application/javascript
01/Dec/2015:07:59:34 +0000    155 192.168.56.1 TCP_CLIENT_REFRESH_MISS/200 506242 GET http://primetel
.com.cy/wp-content/uploads/fonts/PFDinDisplayPro-Thin.ttf admin HIER_DIRECT/primetel.com.cy text/plai
n
01/Dec/2015:07:59:35 +0000     77 192.168.56.1 TCP_MISS/204 430 GET http://csi.gstatic.com/csi? admin
 HIER_DIRECT/csi.gstatic.com image/gif
01/Dec/2015:07:59:35 +0000      1 192.168.56.1 TCP_MEM_HIT/200 5820 GET http://primetel.com.cy/wp-con
tent/plugins/revslider/public/assets/js/extensions/revolution.extension.slideanims.min.js admin HIER_
NONE/- text/javascript
01/Dec/2015:07:59:35 +0000      0 192.168.56.1 TCP_MEM_HIT/200 1722 GET http://primetel.com.cy/wp-con
tent/plugins/revslider/public/assets/js/extensions/revolution.extension.actions.min.js admin HIER_NON
E/- text/javascript
```

Figure 29. Squid's logs

d) Figure **30** shows the results from http://ip.my-proxy.com/ when a user accesses the Web without having the user anonymity featured enabled. The most important fields are:

1. "HTTP_X_FORWARDED_FOR" . It shows the user's public IP (e.g. 217.27.32.7) address along with the Proxy's IP (e.g. 217.27.59.141)
2. "HTTP_VIA". It show the proxy's version (e.g. squid 3.5.5)

3. "HTTP_USER_AGENT". It shows the user's browser information (e.g. Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:41.0) Gecko/20100101 Firefox/41.0).

HTTP Header	Value
HTTP_ACCEPT	text/html,application /xhtml+xml,application/xml; q=0.9,*/*; q=0.8
HTTP_ACCEPT_ENCODING	gzip
HTTP_ACCEPT_LANGUAGE	en-US,en; q=0.5
HTTP_CONNECTION	Keep-Alive
HTTP_HOST	ip.my-proxy.com
HTTP_USER_AGENT	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:41.0) Gecko/20100101 Firefox/41.0
REMOTE_ADDR	162.158.38.165
REMOTE_PORT	45374
HTTP_CACHE_CONTROL	max-age=0
HTTP_X_FORWARDED_FOR	217.27.32.7,217.27.59.141 (Dubious)
HTTP_VIA	1.1 proxy-vnf (squid/3.5.5) (Dubious)

Figure 30. Results taken from http://ip.my-proxy.com/ without user anonymity

Figure **31** shows the results while a user accesses the Web anonymously. It can be observed that the user's real IP is hidden and instead the Proxy's IP is shown. In addition the information about the proxy and the user's browser information are hidden.

HTTP Header	Value
HTTP_ACCEPT	text/html,application /xhtml+xml,application/xml; q=0.9,*/*; q=0.8
HTTP_ACCEPT_ENCODING	gzip
HTTP_ACCEPT_LANGUAGE	en-US,en; q=0.5
HTTP_CONNECTION	Keep-Alive
HTTP_HOST	ip.my-proxy.com
HTTP_USER_AGENT	
REMOTE_ADDR	162.158.38.165
REMOTE_PORT	24729
HTTP_CACHE_CONTROL	max-age=259200
HTTP_X_FORWARDED_FOR	217.27.59.141 (Dubious)

Figure 31. Results taken from http://ip.my-proxy.com/ with user anonymity

3.7. FPGA-based H264 Decoder

3.7.1. Introduction

H264 is currently one of the most widespread video formats and files encoded with it make up a significant part of internet traffic. As such there is significant need for accelerating both encoding and decoding of such videos to enabler quicker and more power efficient processing and delivery of such content. This implementation consists of a hardware circuit that can be deployed on programmable logic and decodes an H264 input stream provided at the input, providing raw frames at the output.

This implementation was performed within the context of T-NOVA more as means to highlight the efficacy of the programmable logic-aware OpenStack implementation developed within WP4 and is described in D4.1 [D4.1]. It consists of a high-performance, data-flow, pipeline architecture written in C++ and fine-tuned for synthesis using Xilinx's Vivado HLS software. This is coupled with additional logic for monitoring performance and starting and stopping a VNF instance.

3.7.2. Architecture

The architecture of a VNF instance is shown on Figure 32. It consists of three main elements:

The core processing components which receive and send frame data and decode them.

A HW monitoring agent which taps the input and output lines and records the number of frames that are processed

A VM control block which interfaces with the Orchestrator over the mAPI and is thus responsible for lifecycle management by starting and stopping the VM. Here it should be noted that since HW VMs are a novel concept only the most basic lifecycle events are supports. More advanced ones like scaling and migration will have to be deferred to future research.

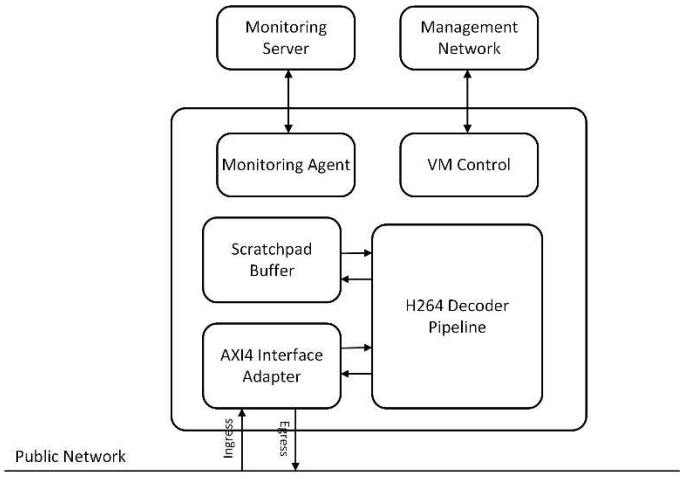


Figure 32 – FPGA-based H264 Decoder Architecture

3.7.3. Functional Description

The VNF's main processing components are:

The AXI4 interface adapter, which converts the memory mapped interface used to communicate with the ARM A9 CPU to the streaming interface used internally by the VNF and back.

The H264 decoding pipeline, that receives an input stream consisting of H264 encoded frames and decodes them into raw frames. The pipeline is implemented as a feed-forward, data flow pipeline and consists of several stages in order to balance the processing overhead and ramp up the achieved clock

frequency. The main steps in this pipeline are:

Error Checking: Verifying that the current decoder context is not corrupted and that decoding can continue.

State Initialization & Reset: Initialize internal decoder variables and reset buffer memories for decoding of the next frame.

NAL Header Parsing & Decoding: H264 bitstreams are organized in packets called NALs. Each NAL is of variable length and contains a header with information regarding its contents. Parsing this information is mandatory to the successful decoding of the frames.

Frame Decoding: This is the core of the accelerator's functionality. It reads the frame data contained in the NAL and decodes them into raw frame bitstream, which is then passed to the output.

A scratchpad buffer, where interim frame data which will be re-used in subsequent processing is being stored.

3.7.4. Interfaces

The VNF's external interfaces comprise four logical network connections (the actual system on which the system is implemented has one physical network port), two of which serve to exchange data traffic to and from the data processing pipeline, one over which lifecycle management is performed, thus enabling the orchestrator to start and stop a VM and one which is used to send monitoring data to the SW monitoring agent which is executed on the ARM processor of the Zynq FPGA SoC (see D4.42 for a detailed description of the monitoring architecture of the FPGA-based NVF).

3.7.5. Technologies

The VNF is designed in C++ and synthesized for the Zynq SoC using Xilinx's Vivado HLS tool. This tool allows FPGA designers to develop and implement designs using a high-level language and thus achieve higher productivity than what would be feasible when using more traditional technologies like Verilog or VHDL.

The VNF has been developed for use with the OpenStack FPGA-based platform developed within T-NOVA in WP4 which includes both means to interconnect the

VNF to the ARM A9 CPU on which OpenStack runs. The hardware side of the platform is based on standard chip interconnection technologies like ARM's AMBA AXI buses which enable for high-speed communication between the ARM subsystem and the VNF.

4. SCALING

Scalability is the capability of a system, network, or process to handle a growing amount of work and to be enlarged in order to accommodate that growth.

There are two methods of adding resources for a particular application: horizontal and vertical scaling:

- to scale horizontally (or scale out/in) means to add more nodes to (or remove nodes from) a system, such as adding a new resource to a distributed software application.
- to scale vertically (or scale up/down) means to add resources to (or remove resources from) a single node in a system, typically involving the addition of CPUs or memory to a single function.

In T-NOVA project only the scale out/in is supported. To support this the scale_in_out section is used in the VDU section of the VNF. The section specifies the max/min instances allowed per VDU defined (i.e VNFC).

4.1. General description

The scaling procedures can be applied to the following T-NOVA VNFs:

1) vSBC (virtual Session Border Controller)
2) vCDN/vHG (virtual CDN / virtual Home Gateway)

This chapter contains a guideline for these procedures, whilst the specific VNF scaling features are described in par. **Erreur ! Source du renvoi introuvable.** (in case of vSBC) and par. **Erreur ! Source du renvoi introuvable.** (in case of vHG).

The following description:

- refers only to the VNF Scaling (ie. increasing the capacity of a VNF), since the Network Service Scaling (ie. Increasing the capacity of a Network service by adding new VNFs) is out of scope for this document
- refers only to the "scale in/out" procedures (adding/removing VDU instances with the same deployment flavour), since the "scale up/down" (adding/removing resources inside a VDU instance) is out of scope of T-NOVA project [D2.41].
- refers mainly to the "*auto-scaling*" use case, depending on the monitoring data generated by the VNF
- consider that the "*on-demand* scaling" use case, from the VNF point of view, to be simply a subset of the "*auto-scaling*" use case.

4.1.1. VNFD parameters for scale in/out

The "auto-scaling" procedures are handled according to the following specific information configured inside the VNF Descriptor (VNFD):

1. *Allowed numbers of instances for scaling*

This information is set using the "*scale_in_out*" attribute, and requires to specify the "minimum" and the "maximum" number of allowed instances.

Example

Assuming that:

- the VNF is composed by two VDUs (VDU1 and VDU2)
- the first VDU mustn't scale
- the second must scale up to 2 instances.

In this case the VNFD "scale-in-out" attribute will be configured in this way:

VDU1 : minimum=1 , maximum=1 → (scaling not allowed)

VDU2 : minimum=1 , maximum=2

2. *Generic scale in/out information*

This information is set using the "*assurance_parameters*" of the VNFD. This attribute allows to define (for the *scale-in* and for the *scale-out*):

- the kind of parameters used for applying the scaling (*param_id*)
- the formula for applying the scaling (*formula*)
- the monitoring interval *(interval*)
- the threshold for the scaling procedures (*value*)
- the kind of threshold (*unit*)
- the number of occurrences requested during the monitoring interval (*breaches_count*).

Example of scale-out

Note: we are assuming to apply the scale-out procedure only if, during the monitoring period (60 sec), the cpu usage exceeds the percentage threshold (80%) for at least 2 times. In this case the VNFD must be initialized in this way:

param_id = "cpu usage"
value: 80
unit : "percentage"
formula: "CPU consumption greater than 80%"
violation:
breaches_count: 2
interval : 60

Example of scale-in

Note: we are assuming to apply the scale-in procedure only if, during the monitoring period (60 sec), the cpu usage exceeds the percentage threshold (30%) for at least 2 times. In this case the VNFD must be initialized in this way:

param_id = "cpu usage"
value: 30
unit : "percentage"
formula: "CPU consumption less than 30%"
violation:
breaches_count: 2
interval : 60

3. *Type of requested scaling*

Two new scale in/out requests (*scale-in* and *scale-out*) were added in the "*vnf_lifecycle_events*" attribute of the VNFD.

Each scaling request must specify also the "VDU-instance ID" that must scale.

4.1.2. Dependencies and impact in T-NOVA subsystems

The implementation of scale in/out procedures needs also the following new developments in charge of WP3/WP4/WP5/WP6, as described in the following chapters.

4.1.2.1. Marketplace (WP6)

1) *VNF creation* (step 2 of the dashboard) : the GUI must allow the definition of the minimum and the maximum number of VDU instances for scaling (since now this operation wasn't possible; minimum and maximum values were always set to "1" by default).
   Moreover the VNFD generated by the Marketplace doesn't contain the "generic" monitoring parameters specified by the GUI.
2) *Lifecycle Events* (step 3 of the dashboard)**:** the GUI must allow the configuration of the new "*scale-in*" and "*scale-out*" events
3) *SLA* (step 4 of the dashboard): since now only the "specific" parameters are shown inside the monitoring parameter section, while the "generic" monitoring parameters are missing. The request is to add also this kind of parameter inside the GUI.

4.1.2.2. Orchestrator/VNFM, VIM, VNF (WP3/WP4/WP5)

The Orchestration/Infrastructure level (WP3/WP4) must implement the following scaling procedures:

1) *Scale-out procedure*
   - The Orchestrator/VIM, on the basis of the collected monitoring data, must verify if they exceed the upper threshold configured inside the VNFD
   - The Orchestrator/VIM, after having created a new VDU instance, must wait until its initialization is finished.
   - The Orchestrator/VNFM sends the VNF Controller a specific scaling event (for example a http command) containing both the type of the requested scaling (scale-out in this case) and the identifier of the VDU instance to be scaled, and waits for the reply.
   - This response is sent by the VNF Controller (O&M) only when the scale-out is finished.

2) Scale-in procedure
   - The Orchestrator/VIM, on the basis of the collected monitoring data, must verify if they exceed the lower threshold configured inside the VNFD
   - The Orchestrator/VNFM sends the VNF Controller a specific scaling event (for example a http command) containing containing both the type of the

requested scaling (scale-in in this case) and the identifier of the VDU instance to be scaled, and waits for the reply.
- The VNF Controller stops the instance that must be scaled. When the instance is fully stopped, the VNF Controller informs the infrastructure (VNFM/Orchestrator).
- Finally, the Orchestrator/VIM can now release all the resources linked to the VNF instance.

4.2. vSBC scaling

4.2.1. Assumptions

- The vSBC scaling procedure is applied only to the "media" stream (not to the "signalling" stream)
- The "*CPU usage*" was chosen as parameter for handling the scale in/out procedures. Each Virtual Machine of the vSBC contains a "collectd" daemon that is able to send this generic data to the T-NOVA Monitoring Manager
- the vSBC *scale-out* procedure is obtained by instantiating a new VM and putting it behind a load balancer belonging to the IBCF function of (see par. **Erreur ! Source du renvoi introuvable.** for further details)
- the T-NOVA lifecycle is handled by means of the http protocol. The *scale in* and *scale-out* events are mapped into the PUT http command.

4.2.2. vSBC architecture for scaling

The vSBC (scaling) architecture is composed by 2 VDUs (VDU1 and VDU2), as depicted in the following Figure.

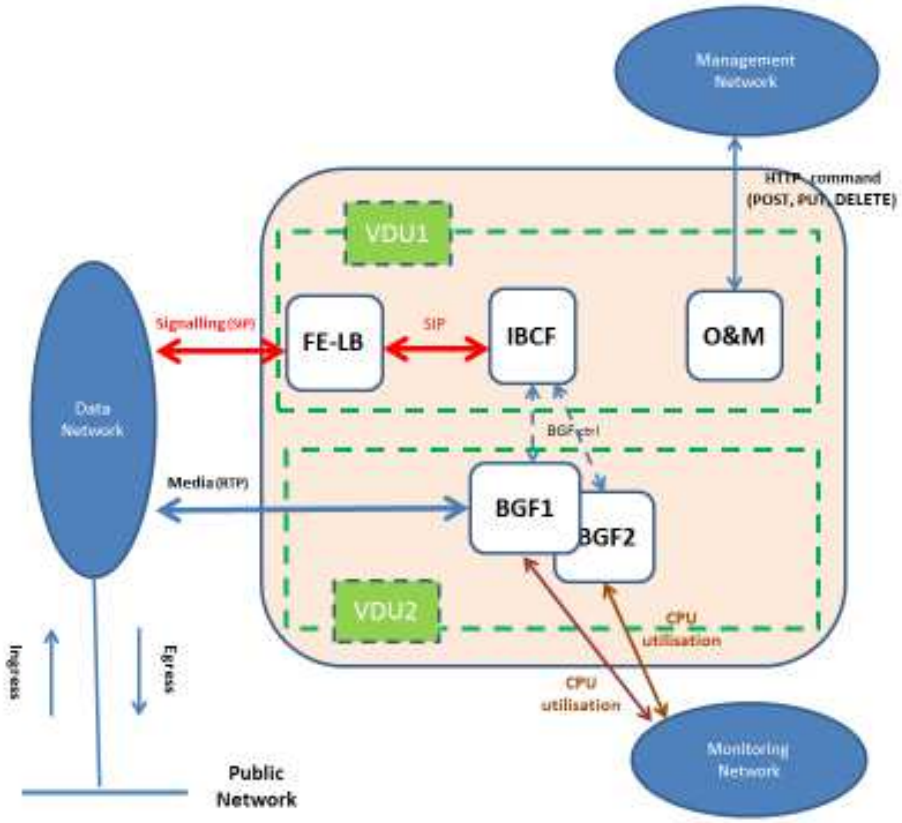


Figure 33 – vSBC architecture for scaling

- VDU1:

  it consists of one scale (1 VM) and performs the following functions:

  - Front-end/Load Balancer function (FE-LB). This functions sends the incoming SIP signalling to the IBCF function
  - Interconnection Border Control Function (IBCF). This function manages the SIP signalling from the initial SIP Request (i.e: INVITE) to the final SIP Request (i.e: BYE), covering all the typical call phases (i.e: setup, renegotiation, tear down, … etc)
  - Operation & Maintenance Function (O&M): it's the *VNF Controller* for the management of the T-NOVA lifecycle (based on http protocol).

- VDU2:

  it consists of two instances (2 VM) and performs the following functions:

  - Border Gateway Function (BGF). This function handles NAT and/or Transcoding of RTP packets.

  The "generic" metrics are sent directly to the Monitoring Manager by the local "Collectd" of each of the two VM hosting the VDU2 instances (BGF1 and BGF2).

4.2.3. Activation of the vSBC scaling procedures

Since the transcoding procedures require a high CPU load, especially in case of video calls, the BGF is the most critical component of the vSBC. So it needs to scale from 1 to "x" instances (for sake of simplicity we have supposed x=2 in Figure 33. The scaling operations have no impact on the configuration of the SIP endpoints, because media ip address and port are exchanged by the SDP Offer/Answer negotiation (typically during the call setup).

The only parameter monitored for the scaling activation is the *CPU utilisation*.

Referring to Figure 33, the first VDU2 instance (BGF1) can handle the incoming media packets at the start of the VNF and, from now on, it is always active and doesn't scale.

The second VDU2 instance (BGF2) can scale according to the "formula" configured inside the VNFD.

The best scaling algorithm could be:

- collect the generic parameter "*CPU utilisation*" data coming from all the BGF instances
- compute the average value
- compare this value with the VNFD thresholds and the "breaches_count" configured inside the VNFD

These operations should be applied by the WP3/WP4, since each VM send the «CPU utilization" data directly to the Monitoring Manager.

If this algorithm can't be applied, a simpler alternative could be the following:

- the second VDU2 instance (BGF2) scales out when the CPU load of the first VDU2 instance (BGF1) exceeds an upper threshold
- the second VDU2 instance (BGF2) scales in when the "CPU load" of the second VDU2 instance (BGF2) goes below a lower threshold.

Once the scale-out procedure is finished, the IBCF function of VDU1 applies an internal load balancing towards the two BGF instances (BGF1 and BGF2), using a round-robin algorithm, unless the BGF instance has already reached an internal threshold of maximum traffic.

4.2.3.1. vSBC scale-out flow chart

The following Figure depicts the scale-out flow chart.

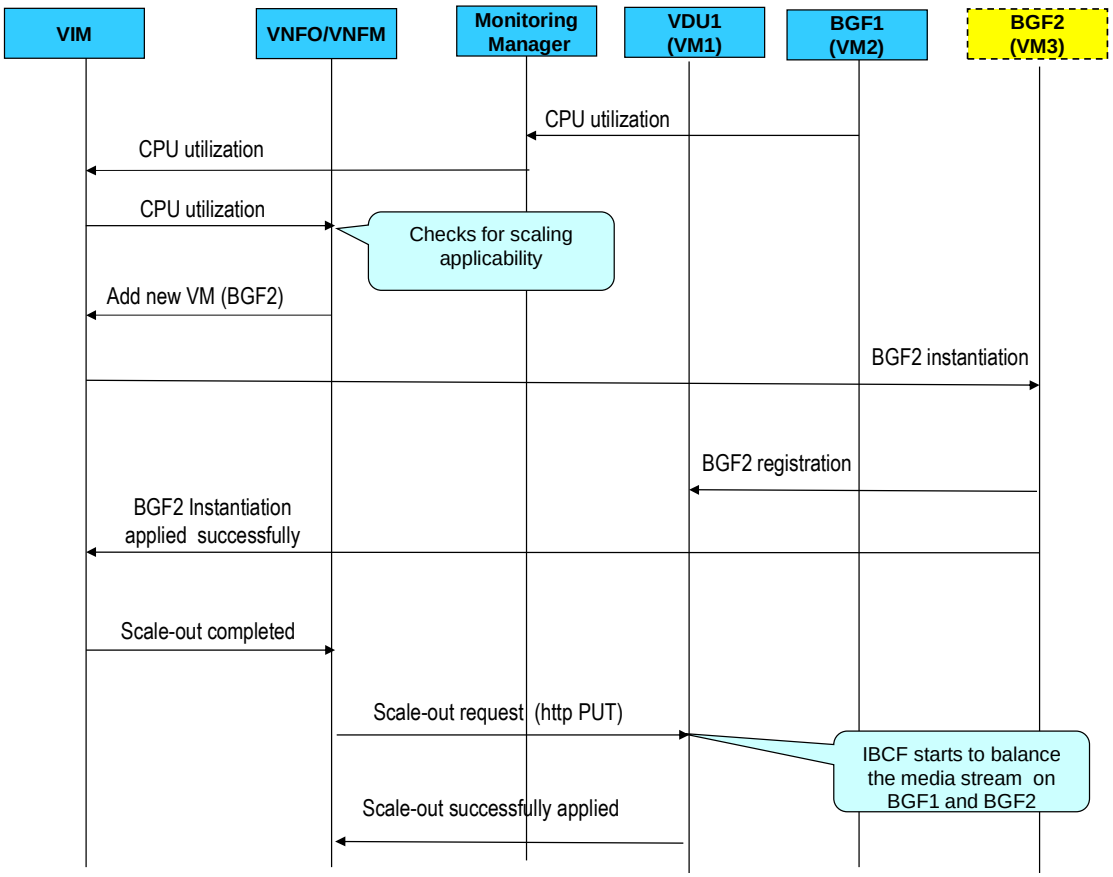


Figure 34 – vSBC "Scale-out" flow chart

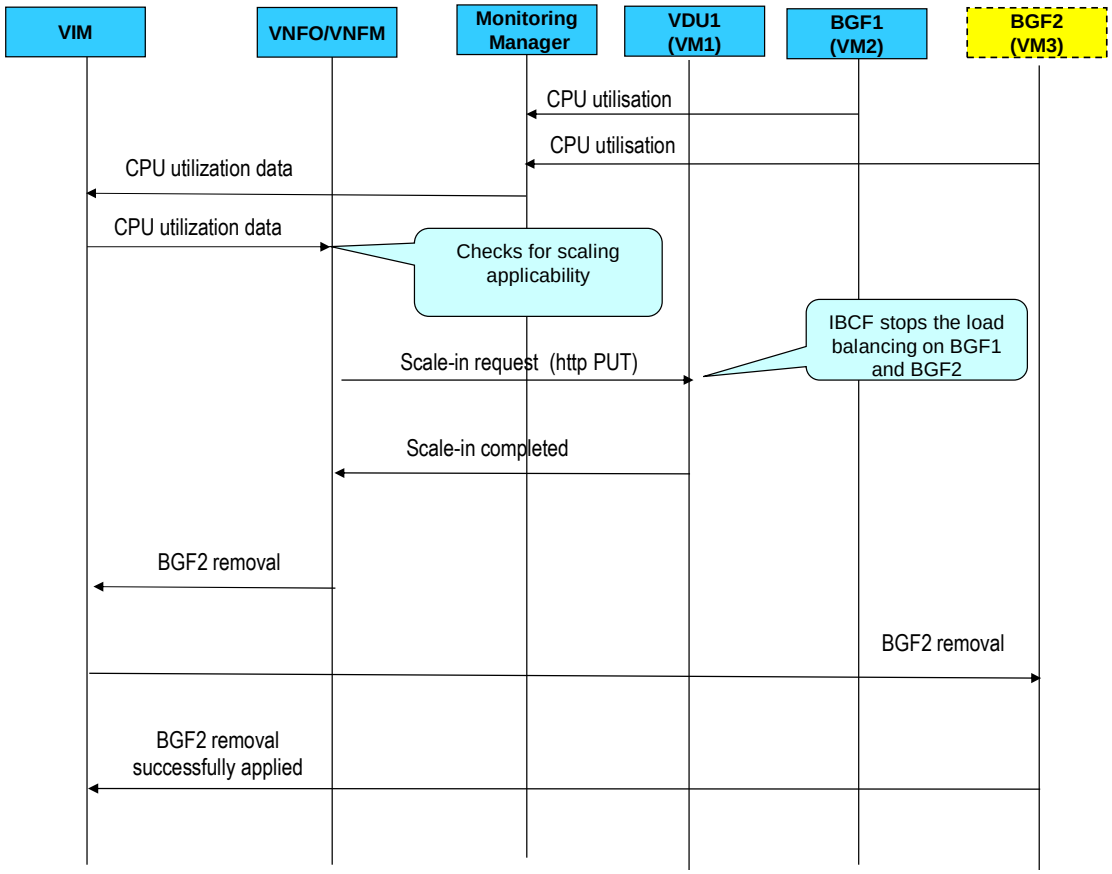
Cyclically the BGF1 instance, by means of its local Collectd daemon, sends the "*CPU utilisation*" metric to the Monitoring Manager. These data are sent to the VIM and then to the VNFO.

If the collected metrics satisfy the VNFD scale-out rules described in par. **Erreur ! Source du renvoi introuvable.** , the VNFO requires a new VDU2 instance (BGF2), and waits the acknowledge coming from the VNF at the end of its initialization phase.

The scale-out request must be sent to the VNF Controller (O&M) of the VDU1 through the Management Network. This request is a http PUT method, different from the "Start" PUT request (i.e: on the basis of a different URL), and must contain the identifier of the new VDU2 instance (BGF2).

A Scale-out request is like a "start" request for the new VDU2 instance (BGF2) since it enables, from now on, the handling of the media stream.

At the receipt of the scale-out command, the VNF Controller (O&M) of the VDU1 sets to "enable" the administration status of the BGF2 instance. From now on the IBCF function starts to balance the media stream on BGF1 and BGF2.



4.2.3.2. vSBC Scale-in flow chart

Figure 35 – vSBC "scale-in" flow chart

Cyclically both BGF1 and BGF2 instances, by means of their local "Collectd" daemon, send the "*CPU utilisation*" metric to the Monitoring Manager. These data are sent to the VIM and then to the VNFO.

If the collected metrics satisfy the VNFD scale-in rules described in par. **Erreur ! Source du renvoi introuvable.**, the VNFO/VNFM requires the VNF to stop the usage of the VDU2 instance (BGF2), and waits the acknowledge coming from the VNF at the end of this operation.

The scale-in request must be sent from the VNFO/VNFM to the VNF Controller (O&M) of the VDU1, through the Management Network. This request is a http PUT method, different from the "Start" request (i.e: on the basis of a different URL), and must contain the identifier of the VDU2 instance that must be removed (BGF2).

A scale-in request is like a "Stop" request for the VNF, since the VNF Controller (OEM) stops, in a graceful way, all the active media sessions of BGF2. The administrative status of BGF2 is set to "disable", so that the IBCF, from now on, can use only the first BGF instance (BGF1) to handle new calls.

During the scale-in procedure no media calls are moved from the BGF2 instance to the BGF1 instance. Only when all active BGF2 calls are finished, then all the resources linked to the BGF2 can be released. Anyways there is a maximum time interval after which all the active sessions are stopped.

Finally the VNF controller (O&M) of the VDU1 notifies the VNFM/VNFO the end of the scale-in operation, so that the VNFO can request the VIM to release all resources linked to the BGF2 instance.

4.2.4. Summary of the vSBC scaling procedures

The following Figure summarizes the vSBC "scale-out" procedure previously described in par. **Erreur ! Source du renvoi introuvable.**

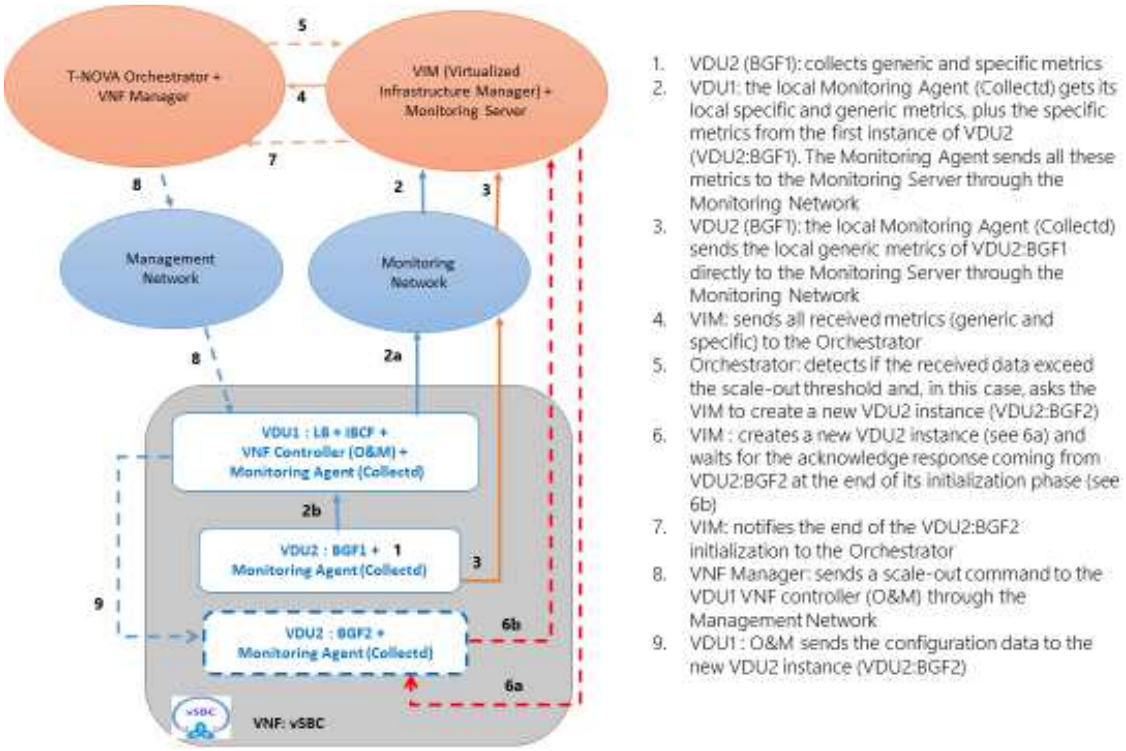


Figure 36 - vSBC "scale out" procedure

The following Figure summarizes the vSBC "scale-in" procedure previously described in par. **Erreur ! Source du renvoi introuvable.**

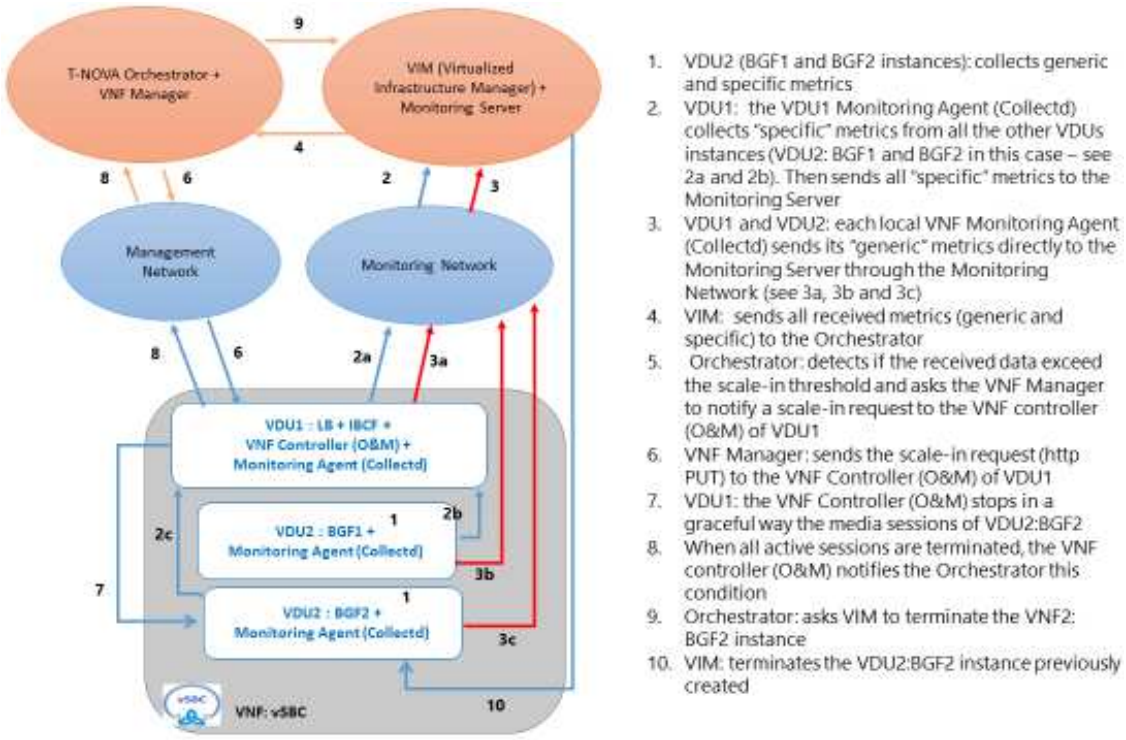


Figure 37 - vSBC "scale-in" procedure

4.3. vHG/ vCDN scaling

Thanks to the use of SaltStack and the Infrastructure as a Service Approach, scaling is made quite easy for the vHG/vCDN. In this section, we briefly describe how we implemented scaling. We followed the antifragile pattern, allowing each micro service to operate in autonomy so that the whole VNF can be resilient and easily scalable.

4.3.1. Scaling out

First, the monitoring agent reports that the storage is running low, or that the ingestion procedure cannot cope with the ingestion demand, thanks to application-specific metrics.

Then, the VNFM launches a new VM (storage node or worker node depending on which metric is breached) and this node is provided with the IP address of the VNF controller. The new VM connects to the controller so that it can be part of the infrastructure under its supervision.

When the VNFM send the scale-up lifecycle event, the controller re-synchronize the infrastructure with all the connected VMs. It's important to note that the exact same procedure is used when scaling the VM and when starting them: software is downloaded (swiftstack or docker + custom made-images), the VM is configured (configuration files are written and services are launched) according to its "role" as a storage node or as a worker.

Once a new worker joins the pool, it is ready to take orders from the caching orchestrator and starts consuming messages on the broker queue. Similarly, once a new storage nodes arrive, swiftstack proxy node re-balance the load amongst the nodes automatically.

4.3.2. Scaling in

Scaling in can occur easily, as workers can be shut instantaneously: killed tasks will not be acknowledged by the workers and will be automatically relaunched after a timeout. For storage nodes, when the node proxy detects that the integrity of the Distributed Hash Table is compromised (when a node terminates), load is rebalanced on existing nodes, automatically in background task.

5. T-NOVA VNFs: Lessons Learnt

In this section, some practical information collected during the development phase of T-NOVA VNFs is reported. In particular, in the following subsections, the most relevant difficulties encountered during the project are reported and briefly discussed, so as to provide guidelines to new developers who wish to develop new VNFs and/or make them available in the T-NOVA framework. Some novel aspects are also considered, such as the use of various types of HW accelerators to enhance the performance of VNFs.

5.1. Architecture

When it comes to the architecture of the various VNFs developed in T-NOVA, all VNF developers planned to have a typical VNF composed of several VNF components reflecting the different functionalities required for the operation of the VNF. Although this distributed architecture works fine with VMs running on hosts, it does not, unfortunately, work in a seamless way when OpenStack is being used. In fact, OpenStack does not really count as a complete NFV infrastructure and raised several networking issues in terms of automated VNF deployment, Service Function Chaining (SFC), traffic forwarding and inter-VM communication, required for VNFs such as the vSA and the Traffic Classifier (vTC) to function properly. The automated and functional integration of the vSA and vTC to OpenStack's networking environment, and more specifically to Neutron service, is non-trivial and remains to be substantiated and implemented as Neutron at the moment does not offer much freedom and flexibility on arbitrary traffic steering.

In order to support direct traffic forwarding, meaning the virtual network interface of one Virtual Network Function Component (VNFC) to be directly connected to another VNFC's virtual network interface, a modification on Neutron's OVS needs to be applied. Each virtual network interface of a VNFC is reflected upon one TAP-virtual network kernel device, a virtual port on Neutron's OVS and a virtual bridge connecting them. This way, packets travel from the VNFC to Neutron's OVS through the Linux kernel. The virtual kernel bridges of the two VNFCs need to be shut down and removed. Then an OVSDB rule needs to be applied at the Neutron OVS, applying an all-forwarding policy between the OVS ports of the corresponding VNFCs.

5.2. Descriptors

The original input for creation of the descriptors for the T-NOVA VNFs was from the early works of ETSI NFV ISG. As the information model used by ETSI ISG NFV is a subject to change, our efforts focused in producing a T-NOVA derivative that would efficiently support the Marketplace imposed rules for Function Developers in delivering their VNFs. Rules as such focus on the the following main sections

- Trading and brokerage support
- SLA assurance
- Networking (via mandatory specified virtual networks)

- QoS not fully supported yes, only BW limiting
- VNFC collapse to single VDU mapping. Due to focusing in VMs rather than containers.

The decided descriptors used by TeNOR when a NS is composed by a number of VNFs and decomposed when the the service is to be deployed according to the PoP selected for the instantiation of each VNF. Internally the system is translating the previously presented descriptor into HOT template (HEAT) assuming that the it is assumed that the PoPs are supporting Openstack. The mapping of the T-NOVA descriptor to HEAT templates is not one to one, additionally for the enforcement of the graph VNF Forwarding Graph information has to pass through the SDN Controller as the integration of the OpenDayLight SDN controller is still not supported by both Openstack and ODL.

5.3. Networking

The network acceleration in a NFV environment is an ever-evolving important aspect of the virtualized network environment under-test. The mitigation of physical network functions to a virtualized environment, improves portability and flexibilty, but also limits and penalizes the performance. This is why several mechanisms have been developed in order to accelerate the packet processing performance in a virtualized environment, such as DPDK and SRIOV.

The Data Plane Development Kit (DPDK) framework succeeds in maximizing packet throughput in a virtualized environment. A novel DPDK-enabled version of the vTC has been in order to optimize the packet-handling and processing for the inspected and forwarded traffic, by bypassing the kernel space. The analyzing and forwarding functions are performed entirely on user-space which enhances the vTC performance. Performance evaluation results showed that with and without DPDK, a significantly higher performance can be achieved compared to packet processing with the Linux kernel network stack. However, during the experimentation process various issues ascended, regarding the DPDK deployment. Firstly, the DPDK compilation with nDPI libraries was not possible in any Linux VM, functional version under-test is a Ubuntu 14.04 VM.

The second issue, was connectivity issues caused by the loading of the DPDK drivers, as they remove the TCP-IP stack of the virtual NICs, by default. The external access was resolved by adding multiple interfaces to the VM, in order not to lose connectivity. The second most important issue was the packet routing issue inside an Openstack network environment, which functions over a series of linux bridges and OVS ports. An overview of the Openstack networking has been presented in detail at the vTC section. The main purpose of this architecture scheme is to demonstrate that network functions, such as port mirroring and traffic forwarding cannot work properly under a clean Openstack network environment.

In order to support direct traffic forwarding, meaning the virtual network interface of one VNFC be directly connected to another VNFC's virtual network interface, a modification on Neutron's OVS needs to be applied. Each virtual network interface of a VNFC is reflected upon one TAP-virtual network kernel device, a virtual port on Neutron's OVS, and a virtual bridge connecting them. This way packets travel from

the VNFC to Neutron's OVS through the Linux kernel. The virtual kernel bridges of the 2 VNFCs need to be shut down and removed, then an OVSDB rule needs to be applied at the Neutron OVS, applying an all-forwarding policy between the OVS ports of the corresponding VNFCs. NCSRD has developped this method in order to tackle traffic steering issues.

Another major issue solved by this implementation was the traffic mirroring feature provided by the vTC VNF. As the vTC VNF consisted of 2 VNFCs, which received duplicated traffic at the same time, the concept of traffic mirroring was realized. Further on, it was implemented in an Openstack environment using the above method described.

Single Root I/O virtualization (SR-IOV) in networking is a very useful and strong feature for virtualized network deployments. SRIOV is a specification that allows a PCI device, for example a NIC or a Graphic Card, to share access to its resources among various PCI hardware functions: Physical Function (PF) (meaning the real physical device), from it a number of one or more Virtual Functions (VF) are generated. Supposedly we have one NIC and we want to share its resources among various Virtual Machines, or in terms of NFV various VNFCs of a VNF. We can split the PF into numerous VFs and distribute each one to a different VM. The routing and forwarding of the packets is done through L2 routing where the packets are forwarded to the matching MAC VF. The purpose of this section is to share a few tips and hacks we came across during our general activities related to SRIOV.

First of all, the SR-IOV enablement in an Openstack environment is by itself a lesson, as SR-IOV networking operates in an independent separate manner than the rest of the standard Openstack networking. It is in a sense a parallel high-speed road to the Neutron-OVS service. SR-IOV uses VLANs by default for packet routing and path selection, which was caused an extra strain in order for the packet to reach the VNF. However, a different approach was also built using a no-VLAN SR-IOV network, which worked properly and facilitated the experimental process of the vTC VNF.

As already mentioned, in an Openstack environment traffic mirroring is not something trivial, this applies also to the parallel SR-IOV path. In order to tackle this, a live modification was performed at the SR-IOV VFs of the port, in order to achieve traffic mirroring. The detailed actions in order to achieve this, are enlisted below.

Let's say you want to send the same flows and packets to 2 VMs simultaneously.

if you enter the ip link show you should see something like this:

p2p1: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT group default qlen 1000

link/ether a0:36:9f:68:fc:f4 brd ff:ff:ff:ff:ff:ff

vf 0 MAC 00:00:00:00:00:00, spoof checking on, link-state auto

vf 1 MAC fa:16:3e:c0:d8:11, spoof checking on, link-state auto

vf 2 MAC fa:16:3e:a1:43:57, spoof checking on, link-state auto

vf 3 MAC fa:16:3e:aa:33:59, spoof checking on, link-state auto

p2p1: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT group default qlen 1000

link/ether a0:36:9f:68:fc:f4 brd ff:ff:ff:ff:ff:ff

vf 0 MAC 00:00:00:00:00:00, spoof checking on, link-state auto

vf 1 MAC fa:16:3e:c0:d8:11, spoof checking on, link-state auto

vf 2 MAC fa:16:3e:a1:43:57, spoof checking on, link-state auto

vf 3 MAC fa:16:3e:aa:33:59, spoof checking on, link-state auto

In order to perform mirroring and send all traffic both ways we need to change the MAC address both on the VM and on the VF and disable the spoof check.

Let's change vf2 -> vf3

On the VM:

ifconfig eth0 down
ifconfig eth0 hw ether fa:16:3e:aa:33:59
ifconfig eth0 up
ifconfig eth0 down
ifconfig eth0 hw ether fa:16:3e:aa:33:59
ifconfig eth0 up

On the host – VF:

ip link set eth0 down
ip link set eth0 vf 2 mac fa:16:3e:aa:33:59
ip link set eth0 vf 2 spoofchk off
ip link set eth0 up
ip link set eth0 down
ip link set eth0 vf 2 mac fa:16:3e:aa:33:59
ip link set eth0 vf 2 spoofchk off
ip link set eth0 up

After that we have 2 VFs with the same MAC.

But it will still not work. What you have to do is, change again the vf 2 to something resembling the latest MAC

ip link set eth0 vf 2 mac fa:16:3e:aa:33:58
ip link set eth0 vf 2 mac fa:16:3e:aa:33:58

After these changes through the experiments we performed we managed to mirror the traffic on 2 different VFs.

5.4. Acceleration support

5.4.1. FPGA

Designing a VNF for a programmable logic based device harbours completely different challenges from its software counterparts. The fact that this VNF was designed for a specific programmable logic based platform imposed one more layer of constraints. The most important such constraint from a T-NOVA perspective is the limited scaling options available to the VNF. Since the platform used in T-NOVA is single-tenant and the hardware of the VNF is fixed, scaling up by dedicating more resources to this instances is not possible. This means that the VNF should be designed with meaningful performance and resource use goals in mind thus minimizing the need for scaling up (and thus occupying a different device) but also optimize use of the device already in use.

One more critical element is to ensure that all platform components work together harmoniously without causing bottlenecks and thus allowing the hardware VNF instance to reach the maximum possible performance. This pertains to both the OpenStack agent running on the ARM A9 CPUs but also on the software and hardware platform that moves data from OpenStack to the HW VNF and back. While the hardware side of the infrastructure provides ample bandwidth for this, the software component remains unpredictable and outstanding interrupts and excessive buffer copying can easily get the best of any implementation. The effect of this was minimized by extensive testing and performance benchmarking at various location in both the hardware and the software.

5.4.2. Graphical Processing Units (GPU)

Hard parallelization of video coding: In spite of the common thought, typical in the HPC research community, that digital image and video processing algorithms can be greatly accelerated when ported on GPU's as they are well-suited for massive parallelization, in the GPU-accelerated algorithms for video encoding/decoding developed in T-NOVA no simple massive parallelization was possible, and much more effort than expected was needed in order to obtain satisfying (but not impressive) speed-ups with respect to the standard CPU-based algorithms, significantly contradicting this common expectation. This is mainly due to the strict and recursive correlation between adjacent data, both in coded and decoded video streams (each coded/decoded unit – the macroblock – refers to several previously coded/decoded units, thereby imposing a fine-grained sequentially (and therefore no chance for parallelism) in the encoding/decoding algorithms. Finally, a performance improvement, in terms of speed-up with respect to CPU-based algorithms, has been obtained exploiting the GPU; this was achieved by adopting a cooperative CPU-GPU approach, in which some computationally-intensive tasks (such as Motion Estimation) that could run autonomously (i.e. in parallel with the main algorithm) have been delegated to GPU, while the CPU is executing the main computation threads and then "picks" the already computed results from the GPU when needed, thus saving the time necessary for computing them [hwAcc].

GPU virtualization: In order to exploit GPU's in the T-NOVA virtualized infrastructure, it was necessary to "see" the GPU as an available virtual resource. GPU virtualization is not a straightforward issue: GPU manufacturers have developed some solutions providing virtual resources (like NVIDIA's GRID architecture and PCI-pass-through enabled on GPU boards, for instance) but they are not so stable and robust against any combination of hardware platform, virtualization environment, and operating system. At the end, satisfying results have been achieved by adopting kvm as virtualization infrastructure, running VMs using Linux Ubuntu 14. Several different GPU platforms from NVIDIA have been successfully virtualized exploiting the PCI-pass-through. This architectural approach has proven to be very efficient, since the virtualized GPU could deliver substantially the same performance as the same physical GPU device in a native environment, in terms of both computation speed and load capabilities.

5.5. VNF initialization

The VNF initialization may require the usage of the "Cloud Init" library at the boot time of the virtual machine, with a specific installation script [4]. In other words, during the bootstrapping phase of a VNF, some information regarding the networking (i.e: Management IP address) can be collected by means of Cloud Init and a specific installation script.

The vSBC initialization is a typical example of this kind of collecting data.

The installation script can be configured using the GUI of the Marketplace, and it covers two main aspects:

1) The first is how to get the networking information (i.e: the IP address of the Management interface), available only after TeNOR has allocated resources for a specific VNF instance. This info can be obtained by means of the "*get_attr*" function, that allows the retrieval of a resource attribute value at runtime. In our case a ***User-Data Script*** is written inside the installation script, beginning typically with "*#!*", as depicted in the following example.
   Example

```
...
resources:
    server_init:
      type: OS::Heat::SoftwareConfig
      properties:
        config:
          str_replace:
            template: |
              #!/bin/bash
                :
            params:
              mng_ip_par: {get_attr: [instance_port, fixed_ips, 0, ip_address]}
```

[4] Cloud init is a UNIX package, supported by multiple cloud providers, that allows developers to initialize cloud instance at boot time. It is installed in the Ubuntu Cloud Images, and also in the official Ubuntu images available on EC2. It is needed in arch "Linux" images that are built with the intention of being launched in cloud (like OpenStack).

```
                :
    get_resource: server_init
```

The "get_attr" function is then copied by Tenor inside the Heat Orchestration Template (HOT).

The VIM provides the information through the Openstack Metadata API, which is collected by the Virtual Machine after its Boot and is sent, via the mAPI, to the VNF controller.

2) Once the on-boarding phase is terminated, TeNor must send the "*Start*" event to the all the VNF instances composing the service that was purchased.
This event must be received when all VNFs have completed their booting process. Only in this way they are available to receive and handle the mAPI lifecycle events. Heat can only provide information whether the requested resources are allocated or not, but can't know if theVNF is still in a booting phase. For this reason the VNF must notify this condition to Heat. This aim can be achieved through a "wc_notify" line in the installation script, as depicted in the following example.

Example

```
...
resources:
    server_init:
      type: OS::Heat::SoftwareConfig
      properties:
        config:
          str_replace:
            template: |
              #!/bin/bash
                :
                 ...
              wc_notify --data-binary '{"status": "SUCCESS"}'

                :
            params:
```

The "wc_notify" line gets replaced by a "curl" line like that one of the following example.

Example

```
curl -i -X POST -H 'X-Auth-Token: 58b96bd5ac50409999bcf20491e8e7fb' -H
'Content-Type: application/json' -H 'Accept: application/json'
http://10.10.1.2:8004/v1/70e1ff61ab0947558833196e4d94f06b/stacks/admin/
5e980984-ab37-449f-9f2c-c3d06d152255/resources/wait_handle1/signal --
data-binary '{"status": "SUCCESS"}'
```

This curl line is executed only at the end of the booting phase and it is used to get the *Wait Condition* resource defined in Heat. This resource will change the VNF stack status to "*completed*" only after receiving a signal (a http POST in the previous example) through the procedure already described.

The VNF Controller component (the O&M component in case of vSBC), once received the *Start* event from the Middleware API, spreads the initialization event to the other VNFC thanks to a specific configuration software.

With this approach the VNF itself is responsible for applying the configuration and the starting of signalling and media flows. The same approach must be used also in case of scale-out procedures, so that the new scaling instance receives the *Start* event when its bootstrap phase is terminated.

6. VNF CHARACTERIZATION

6.1. Scope

This chapter provides some general guidelines to describe how the Virtual Network Functions (VNFs) of T-NOVA framework perform under a particular workload, as required in the task 5.4 of WP5.

The main scope is to suggest a common methodology to execute the Performance testing and to evaluate the obtained measurements by means of load curves, based on the following parameters:

- *Cpu Load* - represents the average system CPU load over a period of time
- *Memory Usage* - represents the amount of system memory used over a period of time
- *Network Throughput* - rate of network message delivery over a communication channel.

Each VNF developer needs to identify the most significant key scenarios, so to derive the load curves related to the generic parameters previously described.

The analysis of the load curves could be useful to the Orchestrator/VIM in order to:

- estimate the configuration resources required during the phase of instantiation
- check, during the live traffic, the compliance with the declared VNF's load curves, in order to recognize some bugs inside the VNF.

The steps for obtaining the load curves and testing the performances are described in par. 6.2.

The final annexes (i.e: Annex A) contain, for each of these steps, some examples related to the T-NOVA VNFs.

6.2. Core Activities of Performance Testing

The performance testing consists of the following six steps:

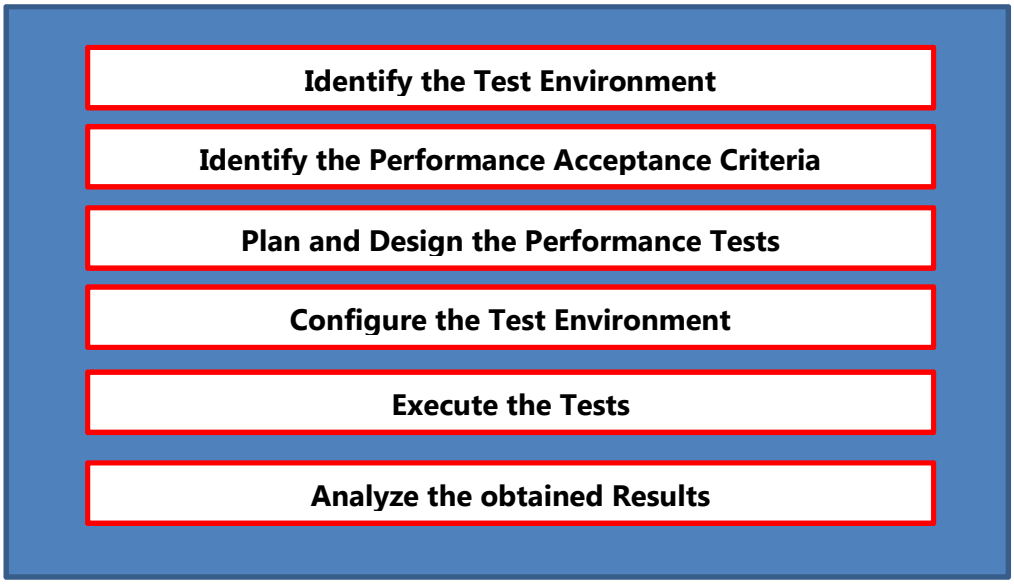


Figure 38 – Steps of Performance Testing

The following chapters will describe in detail each of these steps.

6.2.1. Identify the Test Environment

This step requires the knowledge of the physical environment in which the performance tests will be executed, along with the tools and the hardware required to execute these tests (i.e: load-generation tools and resource monitoring tools). It is requested the knowledge of all details of the hardware, software and network configurations before beginning the testing process.

It is important to have a very high degree of similarity between the hardware, software, and network configuration of the VNF under test in the local laboratory and in the TNOVA testbed.

Some examples related to this testing step can be found in the final Annexes of this document (i.e: Annex A).

6.2.2. Identify the Performance Acceptance Criteria

This step includes goals and constraints for the network throughput, the response times and the resources allocation.
It is very important to start identifying, or at least estimating, both the desired performance and the most common parameters of the VNFs, such as the CPU load, the memory usage and the network throughput (defined in part. 6).

For each of these generic parameters it makes sense to define some reference values, for example the CPU Load not over 85% when the VNF load curve is at its maximum value.
Some examples related to this testing step can be found in the final Annexes of this document (i.e: Annex A).

6.2.3. Plan and Design the Performance Tests

This step identifies the main use cases to be tested and it defines also the metrics to be collected.
These basic parameters, once identified, captured, and correctly reported, can help to identify problems and bottlenecks within the VNF, and this could be helpful also to the T-NOVA Orchestrator/VIM for choosing the VNFs able to manage a specific service.
They could also be useful to identify the most critical scenarios for the basic parameters previously described (CPU load, memory usage and network throughput).
Some examples related to this testing step can be found in the final Annexes of this document (i.e: Annex A).

6.2.4. Configure the Test Environment

This step requires to prepare the test environment in laboratory, composed by the VNF, the Measurement tools and the Load generator.
The test environment must be able to monitor the resources, if necessary.
Normally, it is important to consider some key points when configuring the test environment:

- install in your VNF the same version of software already existing in the T-NOVA testbed
- make a complete configuration of your VNF for all kind of scenarios that you are going to test
- make sure that the test environment is reserved only for these tests
- determine how much load you can generate before the system reaches a bottleneck.

Some examples related to this testing step can be found in the final Annexes of this document (i.e: Annex A).

6.2.5. Execute the Tests

This step executes all tests, and then validates the obtained results.

The test execution can be viewed as a combination of the following sub-tasks:

1. Reset the system.
2. Coordinate the test execution with the other colleagues that could use the VNF.
3. Check the configurations and the state of the environments and data.
4. Begin the test execution, that normally must run for some hours (in this way any peaks will be averaged).

5. While the tests are running, monitor and validate the scripts, the system, and the received data.
6. Upon the test completion, quickly review the results in case of obvious indications that the test was not correctly handled (for example the restart of a VNF component).
7. Archive the tests, the test data, the results, and other information.
8. Rerun the tests by changing the amount of emulated traffic, so to obtain new points of the load curve.

Some examples related to this testing step can be found in the final Annexes of this document (i.e: Annex A).

6.2.6. Analyze the obtained Results

This step analyzes the test results in order to highlight some critical issues while using the VNF, or to obtain input data for applying special services (for example the scaling procedures).
For this goal it is important to analyze the final load curve: its minimum point will be the one with no traffic, while its maximum point will be the one where the traffic cannot be increased because of the overuse of the CPU or memory, a too high network throughput, or a high number of service failures.

Some examples related to this testing step can be found in the final Annexes of this document (i.e: Annex A).

7. CONCLUSIONS

This document provides the final versions of the description of the VNFs developed in T-NOVA. Those VNFs demonstrate, with real-life applications, the capabilities offered by the overall T-NOVA framework.

The developed VNFs cover a wide range of applications, and thus demonstrate the versatility of the overall T-NOVA system to offer commercially attractive appliances and services to real users.

The use of HW acceleration in a virtualized infrastructure has also been addressed: different types of hw accelerators have been used in T-NOVA VNFs – GPU, FPGA and networking accelerators such as SRIOV.

Many performance tests have been carried out, and general guidelines for developers have been provided.

8. REFERENCES

[Abgrall] Daniel Abgrall, "Virtual Home Gateway, How can Home Gateway virtualization be achieved?," EURESCOM, Study Report P055.

[Ansibl] http://en.wikipedia.org/wiki/Ansible_%28software%29

[Chef] http://en.wikipedia.org/wiki/Chef_%28software%29

[Chellouche2012] S. A. Chellouche, D. Negru, Y. Chen, et M. Sidibe, « Home-Box-assisted content delivery network for Internet Video-on-Demand services », in 2012 IEEE Symposium on Computers and Communications (ISCC), 2012, p. 000544-000550.

[Cruz] T. Cruz, P. Simões, N. Reis, E. Monteiro, F. Bastos, and A. Laranjeira, "An architecture for virtualized home gateways," in 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), 2013, pp. 520–526.

[Collectd] https://collectd.org/

[CUDA] NVIDIA, CUDA C Programming Guide, 2013.

[D2.1] T-NOVA: System Use Cases and Requirements

[D2.21] T-NOVA: Overall System Architecture and Interfaces

[D2.41] T-NOVA: Specification of the Network Function framework and T-NOVA Marketplace

[D4.01] T-NOVA: Interim Report on Infrastructure Virtualisation and Management

[D4.1] T-NOVA: Infrastructure Virtualisation

[D5.01] T-NOVA: Interim report on Network functions and associated framework

[D6.01] T-NOVA: Interim report on T-NOVA Marketplace implementation

[DITG] D-ITG, link: http://traffic.comics.unina.it/software/ITG/

[django] https://www.djangoproject.com/

[Docker] Docker https://www.docker.com/

[DPDK] Data Plane Development Kit, on-line: http://dpdk.org

[DPDK_NIC] DPDK Supported NICs, on-line: http://dpdk.org/doc/nics

[DPDK_RN170] INTEL, "Intel DPDK Kit", http://dpdk.org/doc/intel/dpdk-release-notes-1.7.0.pdf

[DSpace] DSpace http://www.dspace.org/

[Duato] Duato, J.; Pena, A.J.; Silla, F.; Fernandez, J.C.; Mayo, R.; Quintana-Orti, E.S., "Enabling CUDA acceleration within virtual machines using

rCUDA," High Performance Computing (HiPC), 2011 18th International Conference on , vol., no., pp.1,10, 18-21 Dec. 2011 doi: 10.1109/HiPC.2011.6152718

[Egi] Egi, Norbert, et al. "Evaluating xen for router virtualization." Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on. IEEE, 2007.

[Eprints] Eprints http://www.eprints.org/

[ES282.001] ETSI ES 282 001: Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN); NGN Functional Architecture

[Fedora] http://www.fedora-commons.org/

[Felter] Felter, Wes, et al. "An Updated Performance Comparison of Virtual Machines and Linux Containers." technology 28: 32.

[Gelas] J.-P. Gelas, L. Lefevre, T. Assefa, and M. Libsie, "Virtualizaing home gateways for large scale energy reduction in wireline networks," in Electronics Goes Green 2012+ (EGG), 2012, 2012, pp. 1–7.

[GetA] http://docs.openstack.org/developer/heat/template_guide/hot_spec.html#hot-spec-intrinsic-functions

[GlueCon14] "Containers At Scale. At Google, the Google Cloud Platform and Beyond". Joe Beda. GlueCon 2014.

[Gupta] V. Gupta, A. Gavrilovska, K. Schwan, H. Kharche, N. Tolia, V. Talwar, and P. Ranganathan, "GViM: GPU-accelerated virtual machines," in 3rd Workshop on System-level Virtualization for High Performance Computing. NY, USA:ACM, 2009, pp. 17-24

[H2] H2 http://www.h2database.com/html/main.html

[Halon] http://www.halon.se/

[Herbaut2015] Herbaut N. ; Negru, D.; Xilouris G, Chen Y." in Network of the Future (NOF), 2015 International Conference and Workshop on the , 1-2 Oct. 2015, doi: 10.1109/NOF.2014.7119778

[hwAcc] P. Comi, P.S. Crosta, M. Beccari, P. Paglierani, G. Grossi, F. Pedersini, A. Petrini, "Hardware-accelerated High-resolution Video Coding in Virtual Network Functions", 2016 European Conference on Networks and Communications (EuCNC 2016), Athens, Greece, June 27–30, 2016.

[IETF2013] https://datatracker.ietf.org/documents/LIAISON/liaison-2014-03-28-broadband-forum-the-ietf-broadband-forum-work-on-network-enhanced-residential-gateway-wt-317-and-virtual-business-gateway-wt-328-attachment-1.pdf

[Invenio] Cdsware http://cdsware.cern.ch/invenio/index.html

[IPERF] Iperf, link: https://iperf.fr/

[IPTraf] http://iptraf.seul.org/

[ITG] http://traffic.comics.unina.it/software/ITG/

[JBoss] JBoss http://www.jboss.org/

[Jetty] Eclipse http://eclipse.org/jetty/

[Karimi] K. Karimi, N.G. Dickson, F. Hamze, A Performance Comparison of CUDA and OpenCL, arXiv:1005.2581v3, arxiv.org.

[KeanMohd] http://core.kmi.open.ac.uk/download/pdf/11778682.pdf

[Kirk] D.B. Kirk, W.W. Hwu, Programming Massively Parallel Processors, 2nd ed., Morgan Kaufmann, 2013.

[Lauro] Di Lauro, R.; Giannone, F.; Ambrosio, L.; Montella, R., "Virtualizing General Purpose GPUs for High Performance Cloud Computing: An Application to a Fluid Simulator," Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on , vol., no., pp.863,864, 10-13 July 2012 doi: 10.1109/ISPA.2012.136

[LR_CPE] http://www.lightreading.com/nfv/nfv-elements/huawei-china-telecom-claim-virtual-cpe-first/d/d-id/710980

[LXC] https://linuxcontainers.org/

[IPTR] Iptraf, link: http://iptraf.seul.org/

[m0n0wall] http://m0n0.ch/wall/

[Maurice] C. Maurice, C. Neumann, Olivier Heen, and A. Francillon, “Confidentiality Issues on a GPU in a Virtualized Environment,” Proceedings of the Eighteenth International Conference on Financial Cryptography and Data Security (FC'14),

[Mei] Mei Hwan Loke t al. (2006) Comparison of Video Quality metrics on multimedia videos

[Mikityuk] Mikityuk, A., J.-P. Seifert, and O. Friedrich. “The Virtual Set-Top Box: On the Shift of IPTV Service Execution, Service Amp; UI Composition into the Cloud.” In 2013 17th International Conference on Intelligence in Next Generation Networks (ICIN), 1–8, 2013. doi:10.1109/ICIN.2013.6670887

[Modig] Modig, Dennis. "Assessing performance and security in virtualized home residential gateways." (2014).

[Murano] Murano https://murano.readthedocs.org/en/latest/

[MySQL] MySQL http://www.mysql.com/

[Nafaa2008] A. Nafaa, S. Murphy, et L. Murphy, « Analysis of a large-scale VOD architecture for broadband operators: a P2P-based solution », IEEE Communications Magazine, vol. 46, nº 12, p. 47- 55, déc. 2008.

[Nec] http://www.nec.com/en/press/201410/global_20141013_01.html

[Netmap] http://info.iet.unipi.it/~luigi/netmap/

[Netty] netty.io

[NFV001] "Network Functions Virtualisation (NFV); Use Cases," ETSI GS NFV 001 V1.1.1, Oct. 2013.

[NFVMAN] ETSI NFV ISG, ETSI GS NFV-MAN 001: "Network Functions Virtualisation (NFV); Management and Orchestration" v1.1.1, ETSI, Dec 2014

[NFVSWA] ETSI NFV ISG, ETSI GS NFV-SWA 001: "Network Functions Virtualisation (NFV); Virtual Network Functions architecture" v1.1.1, ETSI, Dec 2014

[Nvidia] Nvidia, Cuda C Programming Guide, 2015

[OpenCL] AMD, Introduction to OpenCLTM Programming, 2014.

[OpenNF] http://opennf.cs.wisc.edu/

[OpenStack] OpenStack http://www.openstack.org/

[OpenVZ] http://openvz.org/Main_Page

[OPNFV] https://www.opnfv.org/

[Ostinato] https://code.google.com/p/ostinato/

[PFRing] http://www.ntop.org/products/pf_ring/

[pfSense] https://www.pfsense.org/

[PostgreSQL] PostgreSQL http://www.postgresql.org/

[Puppet] http://en.wikipedia.org/wiki/Puppet_%28software%29

[RD048] "HG REQUIREMENTS FOR HGI OPEN PLATFORM 2.0," HGI - RD048, May 2014.

[reddit] http://www.reddit.com/r/networking/comments/1rpk3f/evaluating_virtual_firewallrouters_vsrx_csr1000v/

[REFnDPI] ntop, "Open and Extensible LGPLv3 Deep Packet Inspection Library", on-line: http://www.ntop.org/products/ndpi/

[REFPACE] IPOQUE, "Protocol and Application Classification with Metadata Extraction", on-line: http://www.ipoque.com/en/products/pace

[REFWind] Wind River, "Wind River Content Inspection Engine" on-line: http://www.windriver.com/products/product-overviews/PO_Wind-River-Content-Inspection-Engine.pdf

[RFC1157] A Simple Network Management Protocol (SNMP)

[RFC2647] Benchmarking Terminology for Firewall Performance

[RFC3511] RTP Profile for Audio and Video Conferences with Minimal Control

[RFC4080] Next Steps in Signaling (NSIS)

[RFC4540] NEC's Simple Middlebox Configuration (SIMCO)

[RFC4741] NETCONF Configuration Protocol

[RFC7047] The Open vSwitch Database Management Protocol

[Salt] http://www.saltstack.com/

[SBC_ACME] http://www.acmepacket.com/products-services/service-provider-products/session-border-controller-net-net-session-director, Retrived Nov 2014

[SBC_ALU] http://www.alcatel-lucent.com/solutions/ip-border-controllers, Retrived Nov 2014

[SBC_Audiocodes] http://www.audiocodes.com/sbc, Retrived Nov 2014

[SBC_Italtel] http://www.italtel.com/en/products/session-border-controller, Retrived Nov 2014

[SBC_Metaswitch] http://www.metaswitch.com/products/sip-infrastructure/perimeta, Retrived Nov 2014

[SBC_Sonus] http://www.sonus.net/en/products/session-border-controllers/sonus-session-border-controllers-sbc, Retrived Nov 2014

[Shi] Lin Shi; Hao Chen; Jianhua Sun; Kenli Li, "vCUDA: GPU-Accelerated High-Performance Computing in Virtual Machines," Computers, IEEE Transactions on , vol.61, no.6, pp.804,816, June 2012 doi: 10.1109/TC.2011.112

[Sigcomm] http://www.sigcomm.org/sites/default/files/ccr/papers/2012/October/2378956-2378962.pdf

[Silva] R. L. Da Silva, M. A. C. Fernandez, L. E. I. Gamir, and M. F. Perez, "Home routing gateway virtualization: An overview on the architecture alternatives," in Future Network Mobile Summit (FutureNetw), 2011, 2011, pp. 1–9.

[SNORT] Snort, link: https://www.snort.org/

[TheAge] http://www.theage.com.au/it-pro/business-it/telstra-and-ericsson-testing-virtual-home-gateway-20140721-zv6rh.html

[Tomcat] Tomcat http://tomcat.apache.org/

[TR-124] Broadband Forum, "Functional Requirements for Broadband Residential Gateway Devices, "TECHNICAL REPORT TR-124", Dec. 2006.

[TS23.228] 3GPP TS 23.228 IP Multimedia Subsystem (IMS)

[TS29.238] 3GPP TS 29.238 Interconnection Border Control Functions (IBCF) - Transition Gateway (TrGW) interface, Ix interface

[VBOX] VirtualBox, link: https://www.virtualbox.org/

[vSwitch] INTEL, "Intel DPDK vSwitch", on-line: https://01.org/sites/default/files/downloads/packet-processing/329865inteldpdkvswitchgsg09.pdf

[Vuurmuur] http://www.vuurmuur.org/trac/

[Vyatta] http://www.vyatta.com

[Walters] J. P. Walters, A. J. Younge,D.-I. Kang, K.-T. Yao, M. Kang, S. P. Crago, and G. C. Fox, "GPU-Passthrogh Performance: A Comparison of KVM, Xen, VMWare ESXi, and LXC for CUDA and OpenCL Applications," in Proceedings of the 7th IEEE International Conference on Cloud Computing (CLOUD 2014), IEEE. Anchorage, AK: IEEE, 06/2014

[web2py] http://www.web2py.com/

[Wfirewalls] http://en.wikipedia.org/wiki/Comparison_of_firewalls

9. LIST OF ACRONYMS

Acronym	Explanation
3GPP	Third Generation Partnership Project
API	Application Programming Interface
BGF	Border Gateway Function
CAPEX	Capital Expenditures
CLI	Command Line Interface
CRUD	Create, Read, Update, Delete
DDoS	Distributed Denial of Service
DFA	Deterministic Finite Automaton
DHCP	Dynamic Host Configuration Protocol
DoS	Denial of Service
DPDK	Data Plane Development Kit
DPI	Deep Packet Inspection
DPS	Data Plane Switch
DSP	Digital Signal Processor
DUT	Device Under Test
EMS	Element Management System
ETSI	European Telecommunications Institute
FW	Firewall
HGI	Home Gateway Initiative
HPC	High Performance Computing
HTTP	Hyper Text Transport Protocol
IBCF	Interconnection Border Control Function
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IOMMU	I/O Memory Management Unit
ITU-T	International Telecommunication Union – Telecommunication Standardization Bureau
JSON	JavaScript Object Notation
KVM	Kernel-based Virtual Machine

Acronym	Explanation
LB	Load Balancer
MANO	Management and Orchestration
MIB	Management Information Base
NAT	Network Address Translation
NDVR	Network Digital Video Recorder
NETCONF	Network Configuration Protocol
NF	Network Function
NFaaS	Network Function as a Service
NFVI	Network Function Virtualization Infrastructure
NIO	Non Blockio I/O
NN	Neural Network
NPU	Network Processor Unit
NSIS	Next Steps In Signaling
O&M	Operating and Maintenance
OPEX	Operational Expenditures
OSGI	Open Service Gateway Initiative
OTT	over-the-top
PCI	Peripheral Component Interconnect
PCIe	Peripheral Component Interconnect Express
POC	Proof of Concept
PSNR	Peak Signal-to-Noise Ratio
QoE	Quality of Experience
RAID	Redundant Array of Independent Disks
REST	Representational State Transfer
RFC	Request For Comments
RGW	Residential Gateway
RTP	Real-time Transport Protocol
SA	Security Appliance
SBC	Session Border Controller
SIMCO	Simple Middlebox Configuration
SIP	Session Initiation Protocol
SNMP	Simple Network Management Protocol

Acronym	Explanation
SOM	Self Organizing Maps
SQL	Structured Query Language
SR-IOV	Single Root I/O Virtualization
SSH	Secure Shell
STB	Set Top Box
TSTV	Time Shifted TV
UTM	Unified Threat Management
vCPE	virtualized customer premises equipment
VF	Virtual Firewall
vHG	Virtual Home Gateway
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VNF	Virtual Network Function
VOD	Video On Demand
VQM	Video Quality Metric
vSA	Virtual Security Appliance
vSBC	Virtual Session Border Controller
XML	Extensible Markup Language

10. ANNEX A: vSBC PERFORMANCE CHARACTERIZATION

10.1. Identify the Test Environment

10.1.1. vSBC scenario

The scenario in which the vSBC operates is briefly summarized and discussed in the following sections.

10.1.1.1. vSBC – Architecture

The basic architecture of the virtualized SBC is depicted in the following figure.

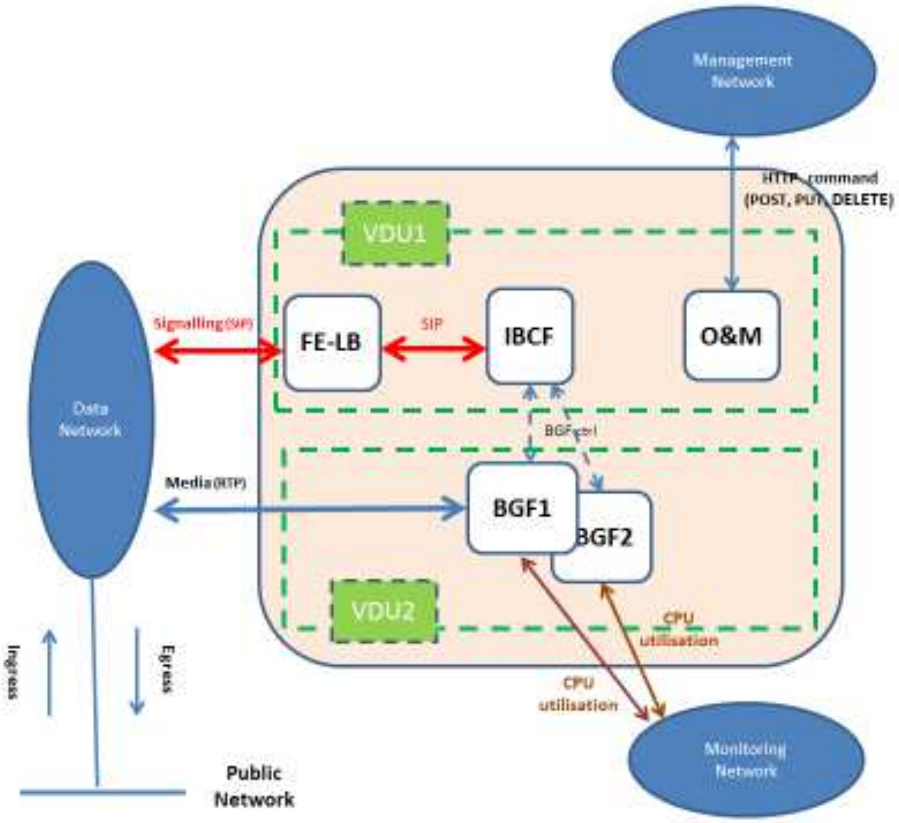


Figure 39 – Basic vSBC architecture

The vSBC consists mainly of two VDUs:

1. VDU1 performs the following functions:
   - Front-end/Load Balancer function (FE-LB): this function sends the incoming SIP signaling to the IBCF function
   - Interconnection Border Control Function (IBCF): this function manages the SIP signaling from the initial SIP Request (i.e: INVITE) to the final SIP Request (i.e:

BYE), covering all the typical call phases (i.e: setup, renegotiation, teardown, … etc)
    - Operating and Maintenance (O&M): it supervises the operating and maintenance functions of the VNF. The O&M component interacts (via HTTP) with the VNF manager for applying the T-NOVA lifecycle.
2. VDU2 performs the following functions:
    - Border Gateway Function (BGF): this function handles NAT and/or Transcoding of RTP packets.

10.1.1.2. vSBC - Installation Server

The testbed is composed of:

- *Server HP ProLiant DL380 G5*, single server equipped with 2 Processors Quad-Core (8 CPU) Intel® Xeon® Processor E5335 (2.00 GHz, 80 Watts, 1333 FSB), Memory 32 GB.
- *CentOS* (version 7.2)
- *Openstack* (Liberty)

10.1.1.3. vSBC- Resource Utilization

The sizing of each single node (virtual machine), in terms of vCPU, RAM and HDD, is described below:

VIRTUAL DEPLOYMENT UNIT	RAM (Gb)	vCPU	HDD (Gb)
VDU1 (VM1: FE-LB+ IBCF+ OEM)	16	6	70
VDU2 (VM2: BGF1)	4	8	64
VDU2 (VM3 : BGF2)	4	8	64

Table 3 – Sizing of each vSBC VDUs

10.1.1.4. vSBC - Traffic Generator

The Traffic Generators used to emulate SIP and Media flows towards the vSBC are:

- *Catapult* (up to about 1500 sip and media concurrent calls)
- *SIPP* (up to about 240 sip and media concurrent calls)
- *NeTracker* (up to about 360 sip and media concurrent calls)

10.1.1.5. vSBC - Network Connections

In the testbed each network connection can handle up to 1Gb/sec.
The basic architecture of the network is depicted in the following figure.

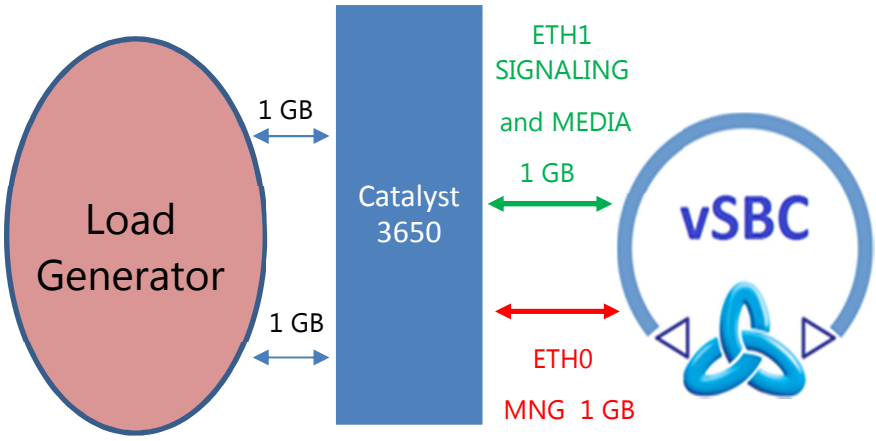


Figure 40 – vSBC network connections

10.1.1.6. vSBC - Traffic Generator Environment

The basic architecture of the test plan between the vSBC and the Traffic Generators (Catapult, or Sipp, or Netracker) is depicted in the following figure.

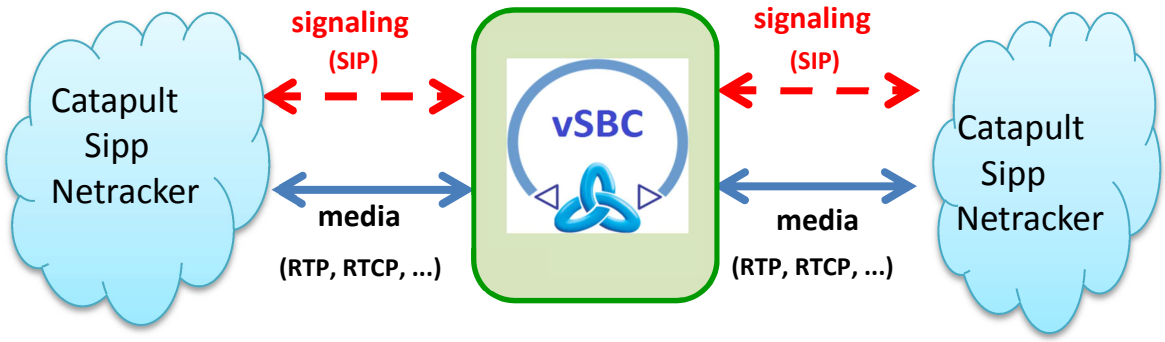


Figure 41 – Test environment plan

10.2. Identify the Performance Acceptance Criteria

10.2.1. vSBC scenario

The estimated values of the CPU load, the memory usage and the network throughput at its maximum value, during the key usage scenarios, are shown in the following table:

TRAFFIC TYPE	TOTAL ACTIVE SESSIONS	CHT [sec]	CPS	FE-LB+ IBCF+ OEM CPU LOAD [%]	FELB+ IBCF+ OEM MEM. USAGE [GB]	BGF CPU LOAD [%]	BGF MEM. USAGE [GB]	THR [bits/sec]
SIP + NAT AUDIO g711-g711	1000	120	8,33	15	14	35	3,5	45000
SIP + TU AUDIO g711-g729	360	120	3	10	14	55	3,5	11000
SIP + TU VIDEO VP8-AVG	10	30	5 every 30 sec.	5	14	90	3,5	8400
SIP + NAT + TU AUDIO + TU VIDEO	1000 (847 NAT + 150 TU AUDIO + 3 TU VIDEO)	120/30	8,33	15	14	90	3,5	45000

Table 4 – vSBC estimated values of cpu load, memory usage and network throughput

10.3. Plan And Design of The Performance Tests

10.3.1. vSBC scenario

The key usage scenarios of the T-NOVA project are summarized in the following table.

TRAFFIC TYPE	TOT. SESS	CHT [sec]	CPS	NUMBER OF SIP MSG	TRANSPORT	MEDIA CODEC	PACK. PERIOD	SIP FAILURE RATE	TO COLLECT
NONE	0	0	0	0	//	//	//	//	cpu load, memory usage, throughput
SIP + NAT AUDIO	1000	120	8,33	7 + 7	UDP	g711-g711	20 ms	<1*10-4	cpu load, memory usage, throughput
SIP + NAT AUDIO	500	120	4,16	7 + 7	UDP	g711-g711	20 ms	<1*10-4	cpu load, memory usage, throughput
SIP + NAT AUDIO	120	120	1	7 + 7	UDP	g711-g711	20 ms	<1*10-4	cpu load, memory usage, throughput
TRAFFIC TYPE	**TOT. SESS**	**CHT [sec]**	**CPS**	**NUMBER OF SIP MSG**	**TRANSPORT**	**MEDIA CODEC**	**PACK. PERIOD**	**SIP FAILURE RATE**	**TO COLLECT**
SIP + TU AUDIO	360	120	3	7 + 7	UDP	g711-g729	20 ms	<1*10-4	cpu load, memory usage, throughput
SIP + TU AUDIO	240	120	2	7 + 7	UDP	g711-g729	20 ms	<1*10-4	cpu load, memory usage, throughput
SIP + TU AUDIO	120	120	1	7 + 7	UDP	g711-g729	20 ms	<1*10-4	cpu load, memory usage, throughput
TRAFFIC TYPE	**TOT. SESS**	**CHT [sec]**	**CPS**	**NUMBER OF SIP MSG**	**TRANSPORT**	**MEDIA CODEC**	**PACK. PERIOD**	**SIP FAILURE RATE**	**TO COLLECT**
SIP + TU VIDEO	10	30	5 every 30 sec.	7 + 7	UDP	VP8-AVC	20 ms	<1*10-4	cpu load, memory usage, throughput

SIP + TU VIDEO	5	30	5 every 30 sec.	7 + 7	UDP	VP8-AVC	20 ms	<1*10-4	cpu load, memory usage, throughput
SIP + TU VIDEO	1	30	5 every 30 sec.	7 + 7	UDP	VP8-AVC	20 ms	<1*10-4	cpu load, memory usage, throughput
TRAFFIC TYPE	**TOT. SESS**	**CHT [sec]**	**CPS**	**NUM. OF SIP MSG**	**TRANSPORT**	**MEDIA CODEC**	**PACK. PERIOD**	**SIP FAILURE RATE**	**TO COLLECT**
SIP + NAT + TU AUDIO + TU VIDEO	1000 (620 NAT + 360 TU AUDIO + 20 TU VIDEO)	120/30	8,33	7 + 7	UDP	g711-g711 + g711-g729 + VP8-AVC	20 ms	<1*10-4	cpu load, memory usage, throughput
SIP + NAT+ TU AUDIO + TU VIDEO	500 (398 NAT+100 TU AUDIO +2 TU VIDEO)	120/30	4,16	7 + 7	UDP	g711-g711 + g711-g729 + VP8-AVC	20 ms	<1*10-4	cpu load, memory usage, throughput
SIP + NAT + TU AUDIO + TU VIDEO	250 (200 NAT+50 TU AUDIO +1 TU VIDEO)	120/30	2,08	7 + 7	UDP	g711-g711 + g711-g729 + VP8-AVC	20 ms	<1*10-4	cpu load, memory usage, throughput

Table 5 - vSBC - Key usage scenarios

10.4. Configure The Test Environment

10.4.1. vSBC scenario

Information about the test environment in Italtel's laboratory is described below.

1. *vSBC sw info:*

- Test plant internal name p33i13a
- Software version (derived from product version of NM-S 2.8-00)
- Patch list
  (CollectD: net-snmp-libs-5.5-54.el6_7.1.x86_64.rpm, libcollectdclient-5.4.1-1.el6.x86_64.rpm, collectd-5.4.1-1.el6.x86_64.rpm, collectd.conf, types.db, types.db.custom, collectd-BGW.conf, send_data.sh)

  (Middleware API: NMS-VNF-MiddlewareApi.zip)

  (Cloud Init: cloud-init-0.7.4-2.el6.noarch.rpm, heat-cfntools-1.2.6-2.el6.noarch.rpm, libyaml-0.1.5-1.el6.x86_64.rpm, python-argparse-1.2.1-2.el6.noarch.rpm, python-backports-1.0-3.el6.x86_64.rpm, python-backports-ssl_match_hostname-3.4.0.2-1.el6.noarch.rpm, python-boto-2.27.0-1.el6.noarch.rpm, python-chardet-2.0.1-1.el6.noarch.rpm, python-cheetah-2.4.1-1.el6.x86_64.rpm, python-configobj-4.6.0-3.el6.noarch.rpm, python-jsonpatch-1.2-2.el6.noarch.rpm, python-jsonpointer-1.0-3.el6.noarch.rpm, python-ordereddict-1.1-2.el6.noarch.rpm, python-prettytable-0.7.2-1.el6.noarch.rpm, python-psutil-0.6.1-1.el6.x86_64.rpm, python-requests-1.1.0-4.el6.noarch.rpm, python-six-1.4.1-1.el6.noarch.rpm, python-urllib3-1.5-7.el6.noarch.rpm, PyYAML-3.10-3.el6.x86_64.rpm)

2. *vSBC internal logical configuration*

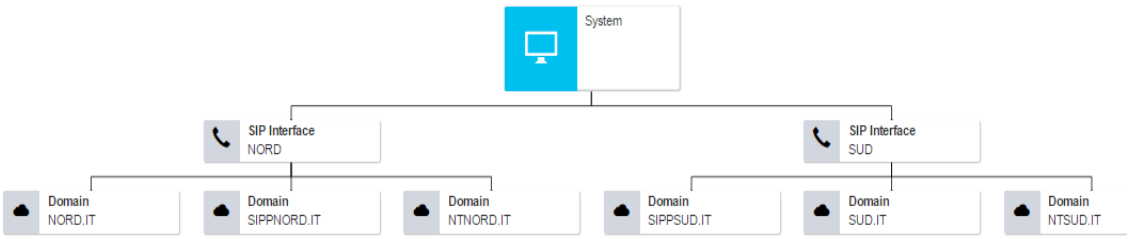


Figure 42 - Internal Overview of logical configuration

3. *vSBC SIP Interfaces*

SIP Interfaces 2

Show 10 entries Search

Name	SIP Interface	TLS	TLS Port	Media Interface	Status
NORD	UDP/172.26.46.135:5060	false	-	media_nord - 172.26.46.164:[10000-13999] 172.26.46.165:[10000-13999]	InService
SUD	UDP/172.26.46.66:5060	false	-	media_sud - 172.26.46.99:[10000-13999] 172.26.46.100:[10000-13999]	InService

Figure 43 - SIP Interfaces (address and port)

4. *vSBC Media Interfaces*

MediaInterfaces 2

Show 10 entries Sear

Name	Ip Address	Port range	ABGW	SRTP
media_nord	172.26.46.164 172.26.46.165	10000 - 13999	true	DISABLED
media_sud	172.26.46.99 172.26.46.100	10000 - 13999	true	DISABLED

Figure 44 – Media Interfaces (address and port)

5. *vSBC SIP Peers*

SIP Peer Groups 8

Show 10 entries Search:

Name	Scan mode	TLS	SIP Peers
NT1	-	false	nt1 - 172.22.58.77:5060
NT2	-	false	nt2 - 172.22.59.77:5060
NT3	-	false	nt3 - 172.22.58.82:5060
NT4	-	false	nt4 - 172.22.59.82:5060
ORIG	ROUND_ROBIN	false	orig1 - 172.23.24.10:5060 / orig2 - 172.23.24.11:5060 / orig3 - 172.23.24.12:5060 / orig4 - 172.23.24.13:5060 / orig5 - 172.23.24.14:5060
SIPPA	-	false	sippa - 172.22.103.12:5060
SIPPB	-	false	sippb - 172.22.225.12:5060
TERM	ROUND_ROBIN	false	term1 - 172.23.32.10:5060 / term2 - 172.23.32.11:5060 / term3 - 172.23.32.12:5060 / term4 - 172.23.32.13:5060 / term5 - 172.23.32.14:5060

Figure 45 – vSBC SIP Peers

Example of SIP Peers

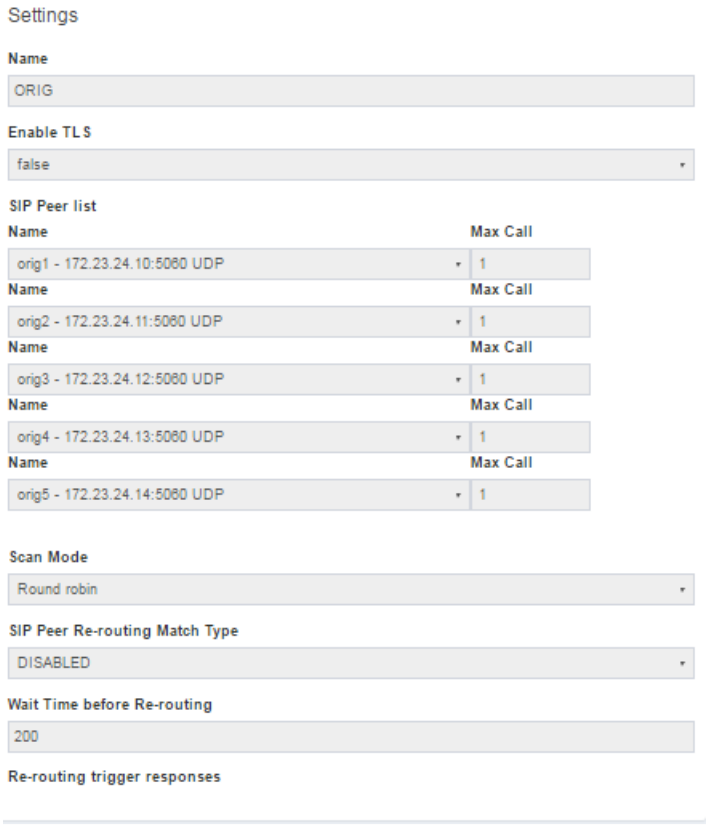


Figure 46 – Example of SIP Peer

6. *vSBC Domains*

Domains 6

Show 10 entries

Name	Interface	Policy	SIP Profile
nord.it	NORD	INGRESS_ADDRESS	SYSTEM_DEFAULTS
ntNord.it	NORD	SOURCE_ADDRESS	SYSTEM_DEFAULTS
ntSud.it	SUD	SOURCE_ADDRESS	SYSTEM_DEFAULTS
sippNord.it	NORD	INGRESS_ADDRESS	SYSTEM_DEFAULTS
sippSud.it	SUD	SOURCE_ADDRESS	SYSTEM_DEFAULTS
sud.it	SUD	SOURCE_ADDRESS	SYSTEM_DEFAULTS

Figure 47 – vSBC Domains

Example of vSBC Domain:

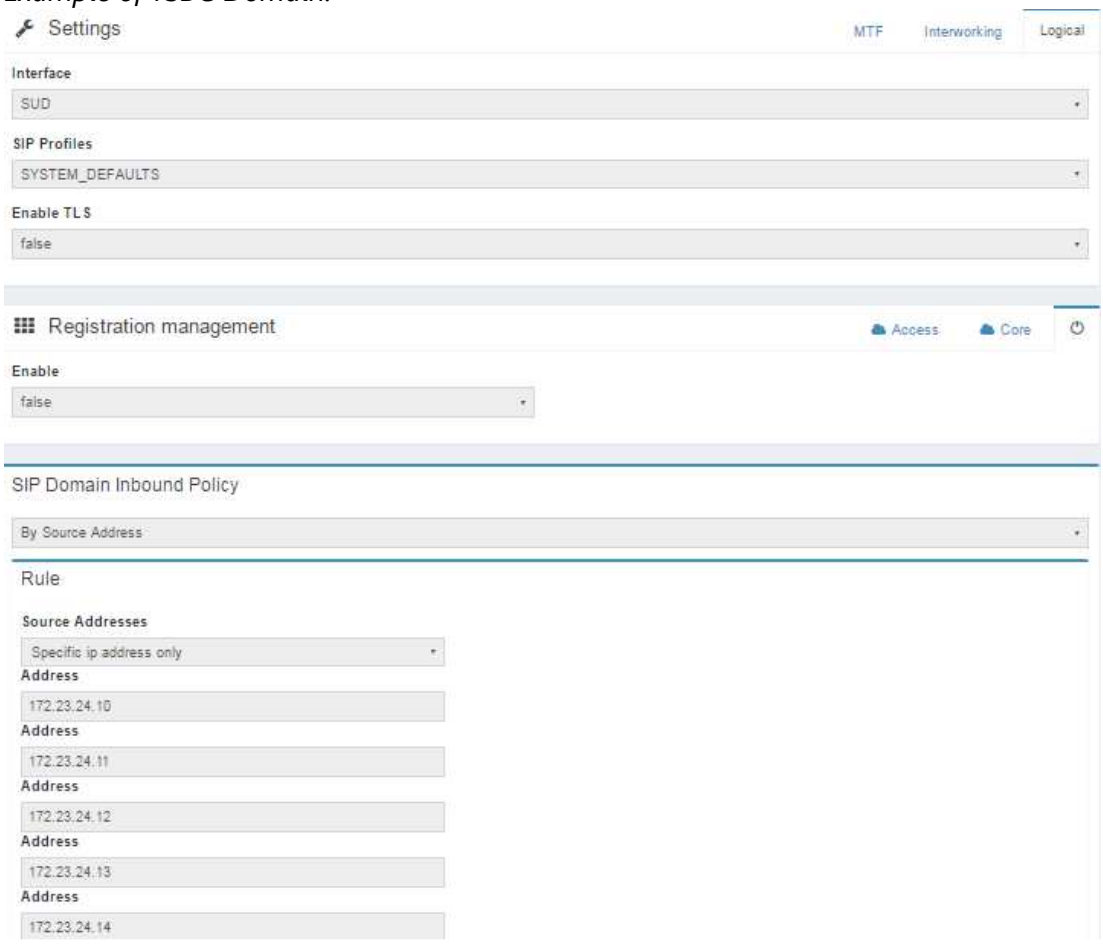


Figure 48 – Example of vSBC Domains

7. *vSBC interconnection*

Figure 49 – vSBC interconnection

Example of vSBC interconnection:

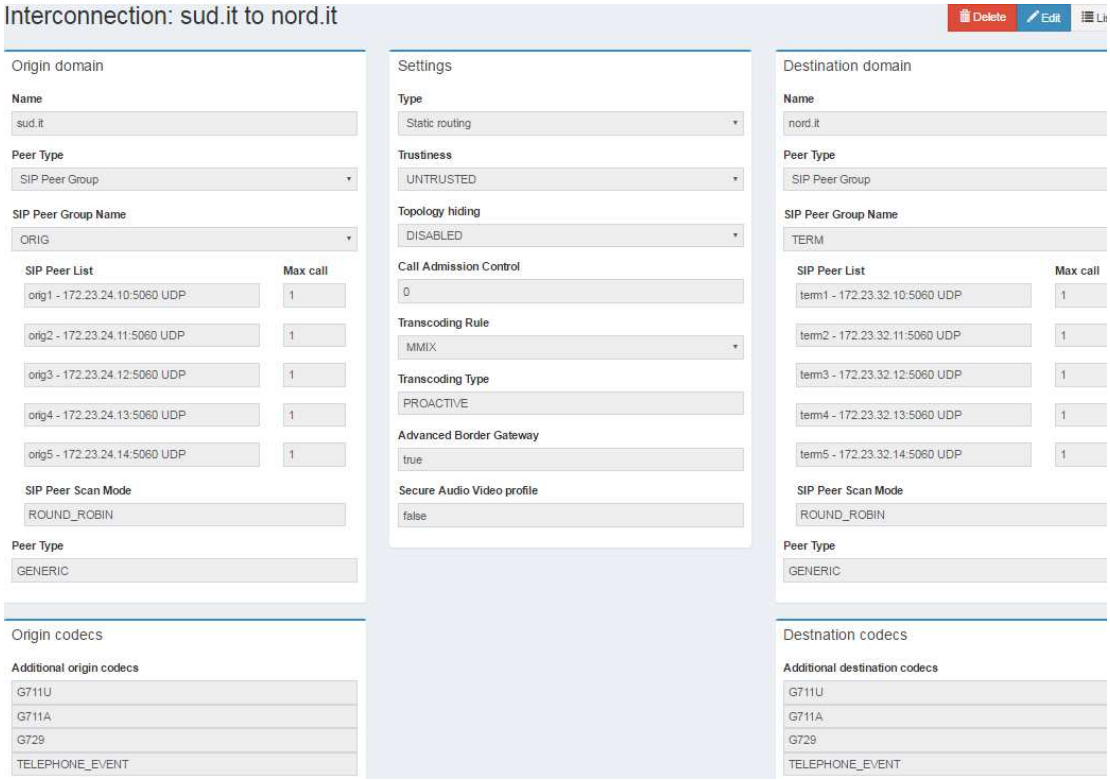


Figure 50 – Example of vSBC interconnection

8. *vSBC codec setting*

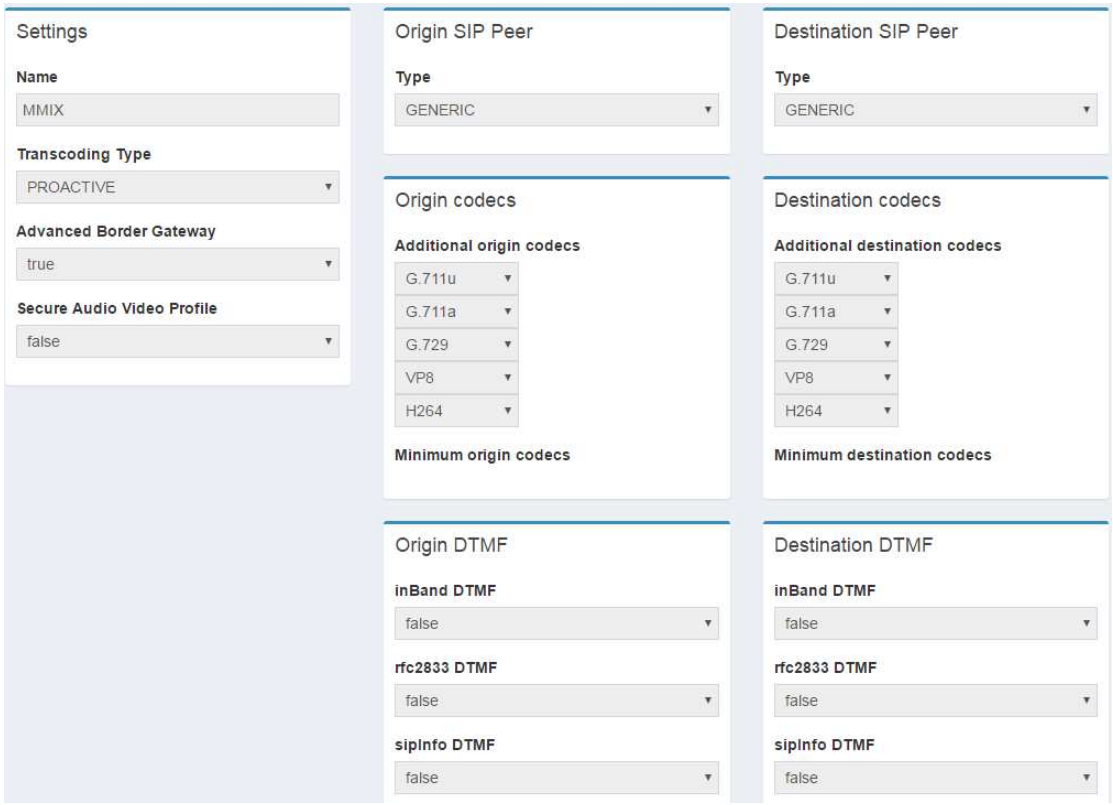


Figure 51 – vSBC codec setting

9. *Load Generators info*:
   - Catapult IP address 138.132.110.16:7
     - Catapult scenario /home/catapult/scenari/CI/NM-S-MLE-p33i13a
     - Catapult traffic Type NAT-TU
   - SIPP IP address 138.132.110.16:7
     - SIPP scenario (./sipp -sf uac_scenario.xml -r 1 -bind_local -i 172.22.103.12 -p 5060 172.26.46.66:5060
     - ./sipp -sf uas_scenario.xml -rtp_echo -bind_local -i 172.22.225.12 -p 5060)
     - SIPP traffic type NAT-TU
   - Netracker IP address 138.132.110.223
     - Netracker scenario NMS-NAT-TU-p33i13a
     - Netracker traffic type NAT-TU

10. *Measurement tools info*:
    - pclServer tool — Internal vSBC tool used to collect CPU and MEMORY usage and THR values
    - Load Generators — Call failure rate, signalling message counters, media counters

10.5. Execute The Test

10.5.1. vSBC scenario

This chapter contains the reports of vSBC test in some typical network scenarios.

10.5.1.1. vSBC-Without-Traffic

Target:

Measurement of the basic parameters (cpu load, memory usage and network throughput**)** of the vSBC without traffic.

*Further data***:**

- Test Duration about 2 hours

Results:

NODE TYPE	TOTAL ACTIVE SESSIONS	CHT [sec]	CPS	CPU LOAD [%]	MEM USAGE [GB]	THR TX+RX ALL ETH [kB/s]	TOTAL CALLS	FAILED CALLS	FAILURE RATE
VM1: FE-LB+ IBCF+OEM	0	0	0	3,93	9,9	0	0	0	0
VM2: (VDU2 - BGF1)	0	0	0	0,51	2,2	0	0	0	0

Table 6 – vSBC Without-Traffic Results

10.5.1.2. vSBC-NAT-Traffic-1000-Active Calls

Target:

Measurement of the basic parameters (cpu load, memory usage and network throughput**)** of the vSBC with traffic SIP calls and Media NAT traffic with 1000 Active Calls.

Further data:

- Test Duration about 2 hours
- Traffic Type NAT
- Codec G711-G711
- Pack Period 20 msec

Results:

NODE TYPE	TOTAL ACTIVE SESSIONS	CHT [sec.]	CPS	CPU LOAD [%]	MEM. USAGE [GB]	THR TX+RX ALL ETH [kB/s]	TOTAL CALLS	FAILED CALLS	FAILURE RATE
VM1: FE.LB + IBCF+OEM	1000	120	8,33	7,08	9,9	66	40024	0	0
VM2: (VDU2 - BGF1)	1000	120	8,33	32,11	2,5	44800	40024	0	0

Table 7 – vSBC NAT-Traffic-1000-Active-Calls Results

10.5.1.3. vSBC-NAT-Traffic-500-Active Calls

Target:

Measurement of the basic parameters (cpu load, memory usage and network throughput**)** of the vSBC with traffic SIP calls and Media NAT traffic with 500 Active Calls.

Further data:

- Test Duration about 1 hour
- Traffic Type NAT
- Codec G711-G711
- Pack Period 20 msec.

Results:

NODE TYPE	TOTAL ACTIVE SESSIONS	CHT [sec]	CPS	CPU LOAD [%]	MEM USAGE [GB]	THR TX+RX ALL ETH [kB/s]	TOTAL CALLS	FAILED CALLS	FAILURE RATE
VM1: FE-LB+ IBCF+OEM	500	120	4,16	5,35	9,5	33	14317	0	0
VM2: (VDU2 - BGF1)	500	120	4,16	15,31	1	22400	14317	0	0

Table 8 – vSBC NAT-Traffic-500-Active Call Results

10.5.1.4. vSBC-NAT-Traffic-120-Active Calls

Target:

Measurement of the basic parameters (cpu load, memory usage and network throughput**)** of the vSBC with traffic SIP calls and Media NAT traffic with 120 Active Calls.

Further data:

- Test Duration about 1 hour
- Traffic Type NAT
- Codec G711-G711
- Pack Period 20 msec

Results:

NODE TYPE	TOTAL ACTIVE SESSIONS	CHT [sec]	CPS	CPU LOAD [%]	MEM USAGE [GB]	THR TX+RX ALL ETH [kB/s]	TOTAL CALLS	FAILED CALLS	FAILURE RATE
VM1: FE-LB+ IBCF+OEM	120	120	1	4,73	9,9	8	3971	0	0
VM2: (VDU2 - BGF1)	120	120	1	3,76	2,3	5200	3971	0	0

Table 9- vSBC NAT-Traffic-120-Active Calls Results

10.5.1.5. vSBC-TU-Audio-Traffic-360-Active Calls

Target:

Measurement of the basic parameters (cpu load, memory usage and network throughput**)** of the vSBC with traffic SIP calls and Media TU Audio traffic with 360 Active Calls.

Further data:

- Test Duration about 2 hours
- Traffic Type TU AUDIO
- Codec G711-G729
- Pack Period 20 msec

Results:

NODE TYPE	TOTAL ACTIVE SESSIONS	CHT [sec]	CPS	CPU LOAD [%]	MEM. USAGE [GB]	THR TX+RX ALL ETH [kB/s]	TOTAL CALLS	FAILED CALLS	FAILURE RATE
VM1: FE-LB+ IBCF+OEM	360	120	3	6,00	9,9	24	11314	0	0
VM2: (VDU2 - BGF1)	360	120	3	53,88	2,4	10500	11314	0	0

Table 10 - vSBC TU-Audio-Traffic-360-Active Calls Results

10.5.1.6. vSBC-TU-Audio-Traffic-240-Active Calls

Target:

Measurement of the basic parameters (cpu load, memory usage and network throughput**)** of the vSBC with traffic SIP calls and Media TU Audio traffic with 240 Active Calls.

Further data:

- Test Duration about 2 hours
- Traffic Type TU AUDIO
- Codec G711-G729
- Pack Period 20 msec

Results:

NODE TYPE	TOTAL ACTIVE SESSIONS	CHT [sec]	CPS	CPU LOAD [%]	MEM USAGE [GB]	THR TX+RX ALL ETH [kB/s]	TOTAL CALLS	FAILED CALLS	FAILURE RATE
VM1: FE-LB+ IBCF+OEM	240	120	2	5,26	9,9	16	14769	0	0
VM2: (VDU2 - BGF1)	240	120	2	35,81	2,3	7000	14769	0	0

Table 11 – vSBC TU-Audio-Traffic-240-Active Calls Results

10.5.1.7. vSBC-TU-Audio-Traffic-120- Active Calls

Target:

Measurement of the basic parameters (cpu load, memory usage and network throughput**)** of the vSBC with traffic SIP calls and Media TU Audio traffic with 120 Active Calls.

Further data:

- Test Duration about 1 hours
- Traffic Type TU AUDIO
- Codec G711-G729
- Pack Period 20 msec

Results:

NODE TYPE	TOTAL ACTIVE SESSIONS	CHT [sec]	CPS	CPU LOAD [%]	MEM USAGE [GB]	THR TX+RX ALL ETH [kB/s]	TOTAL CALLS	FAILED CALLS	FAILURE RATE
VM1: FE-LB+ IBCF+OEM	120	120	1	4,54	9,9	8	3609	0	0
VM2: (VDU2 - BGF1)	120	120	1	16,58	2,2	3500	3609	0	0

Table 12 – vSBC TU-Audio-Traffic-120-Active-Calls Results

10.5.1.8. vSBC-TU-Video-Traffic-10-Active Calls

Target:

Measurement of the basic parameters (cpu load, memory usage and network throughput**)** of the vSBC with traffic SIP calls and Media TU Video traffic with 10 Active Calls.

Further data:

- Test Duration about 1 hour
- Traffic Type TU VIDEO
- Codec VP8(VGA)-AVC(VGA)
- Pack Period 20 msec
- FPS 30

Results:

NODE TYPE	TOTAL ACTIVE SESSIONS	CHT [sec]	CPS	CPU LOAD [%]	MEM. USAGE [GB]	THR [bits/sec]	TOTAL CALLS	FAILED CALLS	FAILURE RATE
VM1: FE-LB+ IBCF+OEM	10	30	5 every 30 sec.	5,02	10	3,5	1164	0	0
VM2: (VDU2 - BGF1)	10	30	5 every 30 sec.	90,78	2,5	8350	1164	0	0

Table 13 – vSBC TU-Video-Traffic-10-Active-Calls Results

10.5.1.9. vSBC-TU-Video-Traffic-5-Active Calls

Target:

Measurement of the basic parameters (cpu load, memory usage and network throughput**)** of the vSBC with traffic SIP calls and Media TU Video traffic with 5 Active Calls.

Further data:

- Test Duration about 2 hours 10 minutes
- Traffic Type TU VIDEO
- Codec VP8(VGA)-AVC(VGA)
- Pack Period 20 msec
- FPS 30

Results:

NODE TYPE	TOTAL ACTIVE	CHT [sec]	CPS	CPU LOA	MEM. USAGE	THR [bits/sec]	TOTAL CALLS	FAILED CALLS	FAILURE RATE

	SESSIONS			D [%]	[GB]				
VM1: FE-LB+ IBCF+OEM	5	30	5 every 30 sec.	5,02	10	1,8	1285	0	0
VM2: (VDU2 - BGF1)	5	30	5 every 30 sec.	48,24	2,4	4200	1285	0	0

Table 14 – vSBC TU-Video-Traffic-5-Active Calls Results

10.5.1.10. vSBC-TU-Video-Traffic-1-Active Calls

Target:

Measurement of the basic parameters (cpu load, memory usage and network throughput**)** of the vSBC with traffic SIP calls and Media TU Video traffic with 1 Active Calls.

Further data:

- Test Duration about 1 hour
- Traffic Type TU VIDEO
- Codec VP8(VGA)-AVC(VGA)
- Pack Period 20 msec
- FPS 30

Results:

NODE TYPE	TOTAL ACTIVE SESSIONS	CHT [sec]	CPS	CPU LOAD [%]	MEM. USAGE [GB]	THR [bits/sec]	TOTAL CALLS	FAILED CALLS	FAILURE RATE
VM1: FE-LB+ IBCF+OEM	1	30	5 every 30 sec.	5,01	10	0,36	123	0	0
VM2: (VDU2 - BGF1)	1	30	5 every 30 sec.	9,64	2,4	850	123	0	0

Table 15 – vSBC TU-Video-Traffic-1-Active Calls Results

10.5.1.11. vSBC-NAT-TU-Audio-TU Video-Traffic-1000-Active Calls

Target:

Measurement of the basic parameters (cpu load, memory usage and network throughput**)** of the vSBC with traffic SIP calls and Media NAT, TU Audio, TU Video traffic with 1000 Active Calls.

Further data:

- Test Duration about 3 hours
- Traffic Type TU AUDIO
- Codec NAT G711-G711
- Codec TU AUDIO G711-G729
- Codec TU VIDEO VP8(VGA)-AVC(VGA)
- Pack Period 20 msec
- FPS 30

Results:

NODE TYPE	TOTAL ACTIVE SESSIONS	CHT [sec]	CPS	CPU LOAD [%]	MEM USAGE [GB]	THR [bits/sec]	TOTAL CALLS	FAILED CALLS	FAILURE RATE
VM1: FE-LB+ IBCF+OEM	1000 (847 NAT+150 TU AUDIO +3 TU VIDEO)	120 (NAT and TU AUDIO) 30 (TU VIDEO)	8,33 (7 NAT + 1,2 TU AUDIO +1 every 30 sec. TU VIDEO	7,71	10	64	83204	14	1,68 * 10-4
VM2: (VDU2 - BGF1)	1000 (847 NAT+150 TU AUDIO +3 TU VIDEO)	120	8,33 (7 NAT + 1,2 TU AUDIO +1 every 30 sec. TU VIDEO	84,32	2,5	42000	83204	14	1,68 * 10-4

Table 16 – vSBC NAT-TU Audio-TU-Video-Traffic-1000-Active Calls Results

10.5.1.12. vSBC-NAT-TU-Audio-TU Video-Traffic-500-Active Calls

Target:

Measurement of the basic parameters (cpu load, memory usage and network throughput**)** of the vSBC with traffic SIP calls and Media NAT, TU Audio, TU Video traffic with 500 Active Calls.

Further data:

- Test Duration about 1 hour and 30 minutes
- Traffic Type TU AUDIO
- Codec NAT G711-G711
- Codec TU AUDIO G711-G729
- Codec TU VIDEO VP8(VGA)-AVC(VGA)
- Pack Period 20 msec

- FPS 30

Results:

NODE TYPE	TOTAL ACTIVE SESSIONS	CHT [sec]	CPS	CPU LOAD [%]	MEM USAGE [GB]	THR [bits/sec]	TOTAL CALLS	FAILED CALLS	FAILURE RATE
VM1: FE-LB+ IBCF+OEM	500 (398 NAT+100 TU AUDIO +2 TU VIDEO)	120 (NAT and TU AUDIO) 30 (TU VIDEO)	4,16 (3,31 NAT + 0,83 TU AUDIO +1 every 30 sec. TU VIDEO	6,22	10	32	22883	0	0
VM2: (VDU2 - BGF1)	500 (398 NAT+100 TU AUDIO +2 TU VIDEO)	120	4,16 (3,31 NAT + 0,83 TU AUDIO +1 every 30 sec. TU VIDEO	48,88	2,4	21500	22883	0	0

Table 17 – vSBC NAT-TU-Audio-TU-Video-Traffic-500 Active Calls Results

10.5.1.13. vSBC-NAT-TU-Audio-TU Video-Traffic-250-Active Calls

Target:

Measurement of the basic parameters (cpu load, memory usage and network throughput**)** of the vSBC with traffic SIP calls and Media NAT, TU Audio, TU Video traffic with 250 Active Calls.

Further data:

- Test Duration about 1 hour 30 minutes
- Traffic Type TU AUDIO
- Codec NAT G711-G711
- Codec TU AUDIO G711-G729
- Codec TU VIDEO VP8(VGA)-AVC(VGA)
- Pack Period 20 msec
- FPS 30

Results:

NODE TYPE	TOTAL ACTIVE SESSIONS	CHT [sec]	CPS	CPU LOAD [%]	MEM USAGE [GB]	THR [bits/sec]	TOTAL CALLS	FAILED CALLS	FAILURE RATE
VM1: FE-LB+	250 (200 NAT+50 TU AUDIO +1 TU	120 (NAT and TU	2,08 (1,66 NAT + 0,41	5,03	10	16	11827	0	0

IBCF+OEM	VIDEO)	AUDIO) 30 (TU VIDEO)	TU AUDIO +1 every 30 sec. TU VIDEO						
VM2: (VDU2 - BGF1)	250 (200 NAT+50 TU AUDIO +1 TU VIDEO)	120	2,08 (1,66 NAT + 0,41 TU AUDIO +1 every 30 sec. TU VIDEO	24,80	2,4	10800	11827	0	0

Table 18 – vSBC NAT-TU-Audio-TU-Video-Traffic-250-Active-Calls Results

10.6. Analyze The Obtained Results

10.6.1. vSBC scenarios

This paragraph contains some examples of load curves related to the CPU and Memory Usage and Network Throughput of the vSBC described in par. 3.2.

10.6.1.1. vSBC - CPU Load Curves

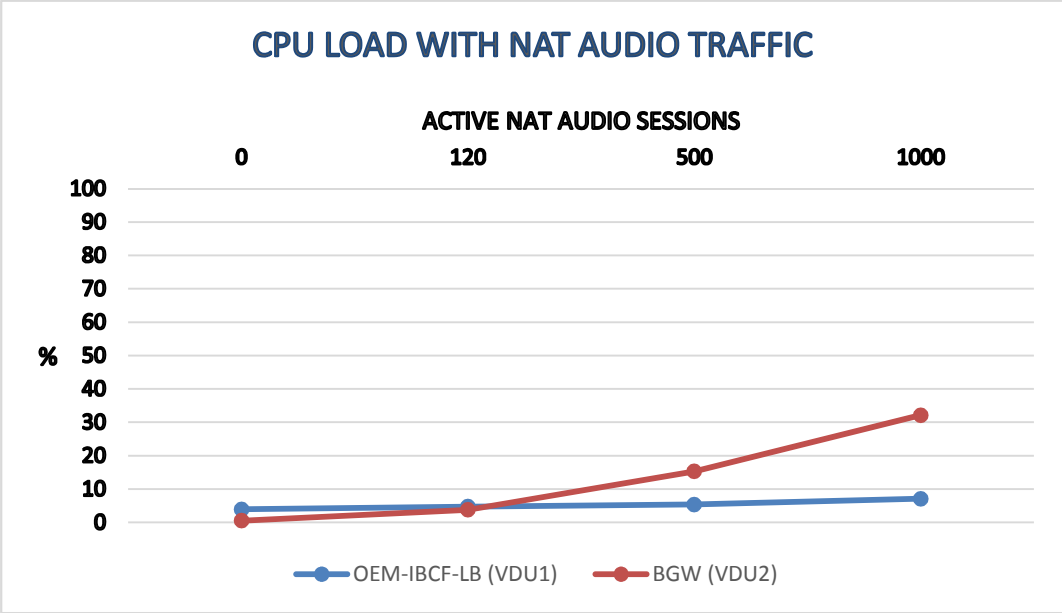


Figure 52 – CPU Load Curve (audio traffic with NAT)

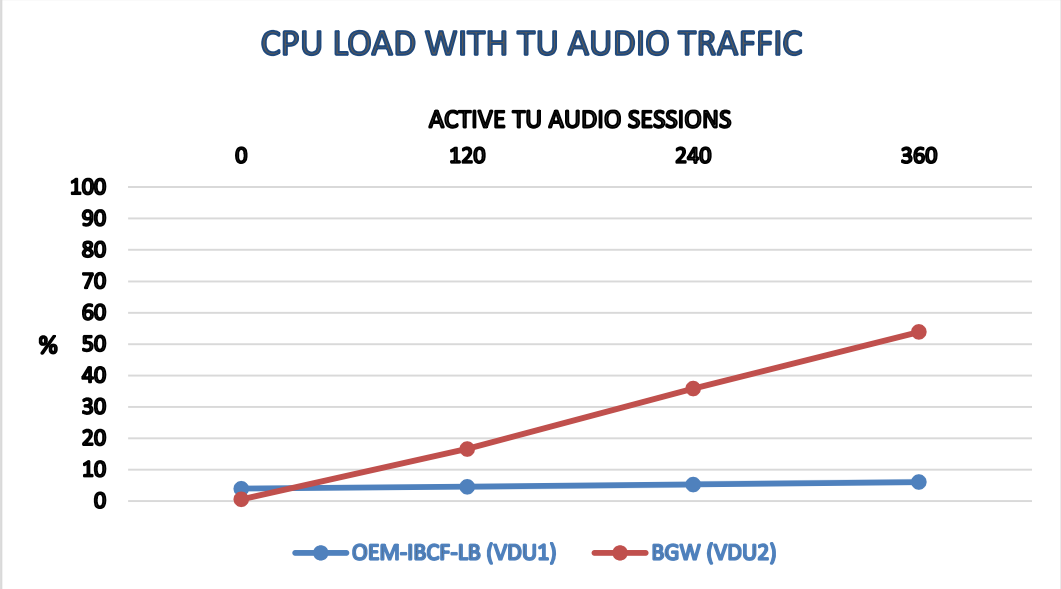


Figure 53 – CPU Load Curve (audio traffic with Trascoding)

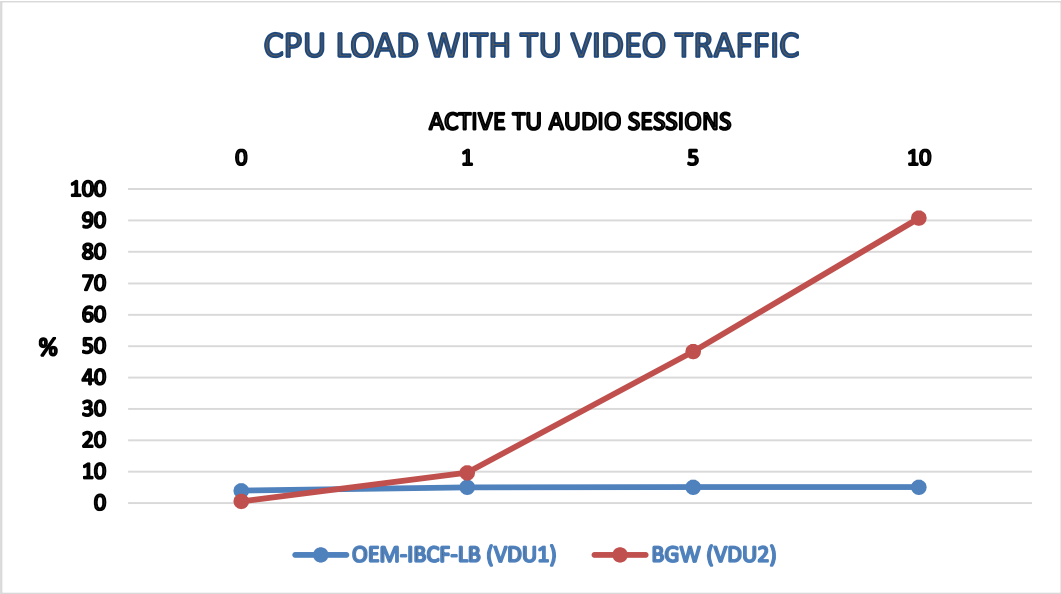


Figure 54 – CPU Load Curve (video traffic with Trascoding)

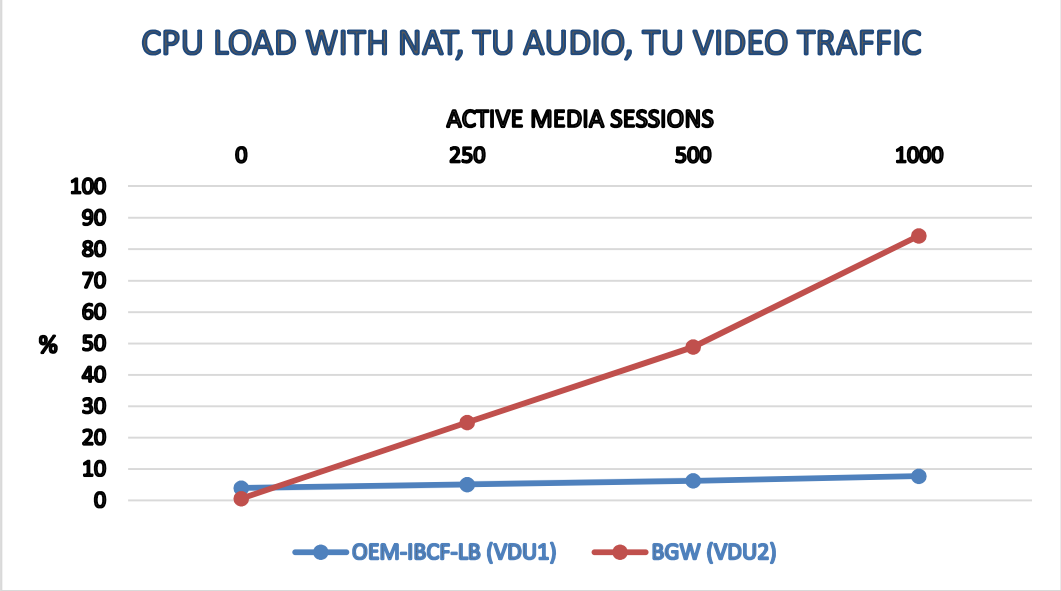


Figure 55 – CPU Load Curve (audio traffic with NAT, audio and video traffic with Trascoding)

10.6.1.2. vSBC - Memory Usage Load Curves

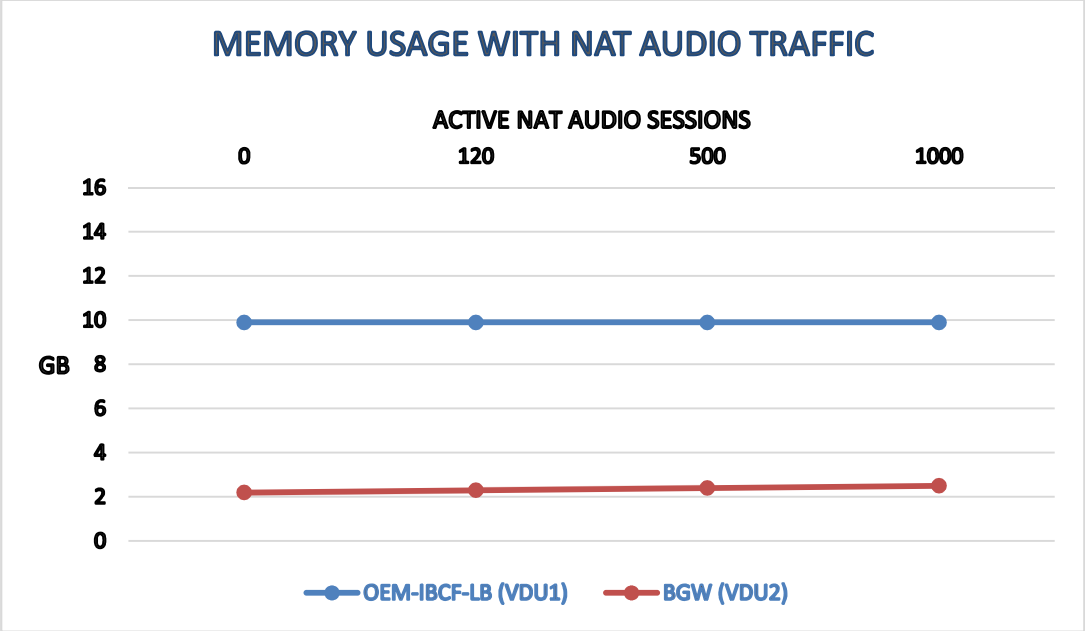


Figure 56 – Memory Usage Load Curve (audio traffic with NAT)

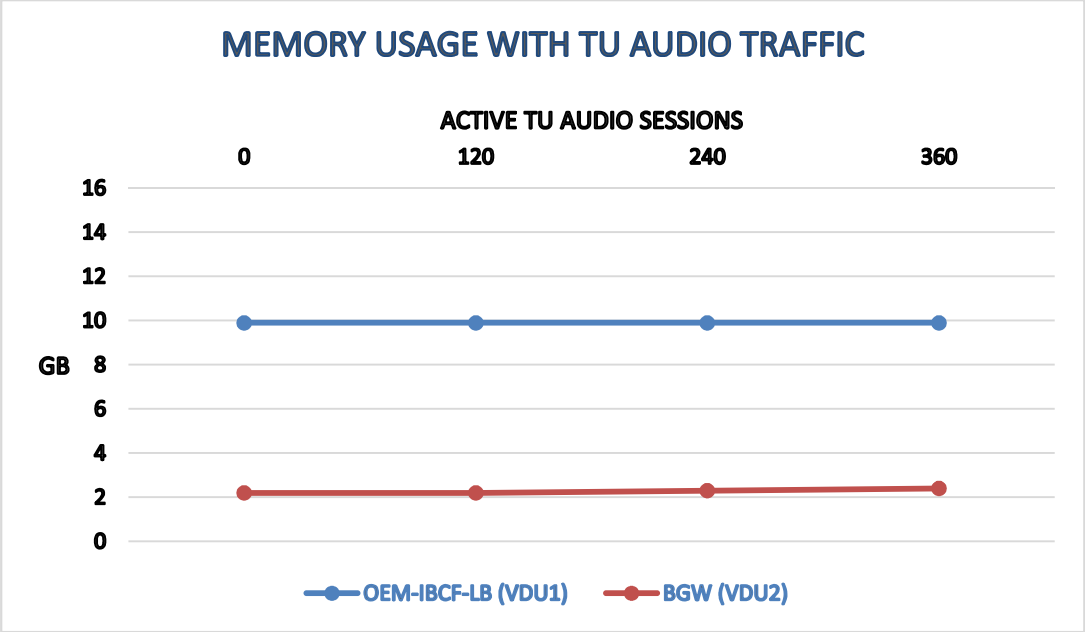


Figure 57 – Memory Usage Load Curve (audio traffic with Transcoding)

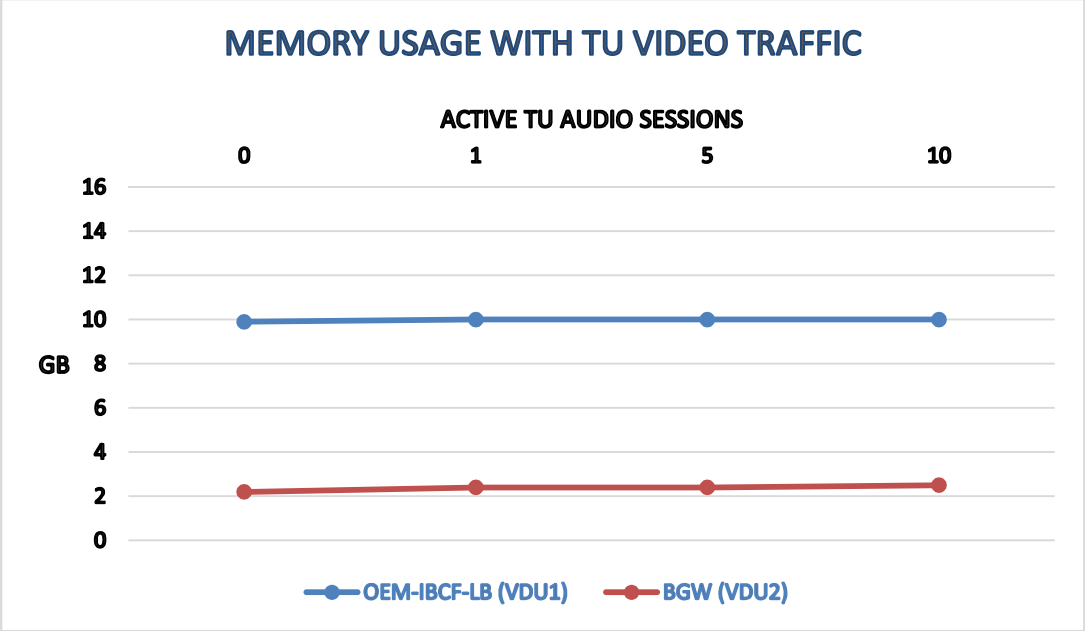


Figure 58 – Memory Usage Load Curve (video traffic with Transcoding)

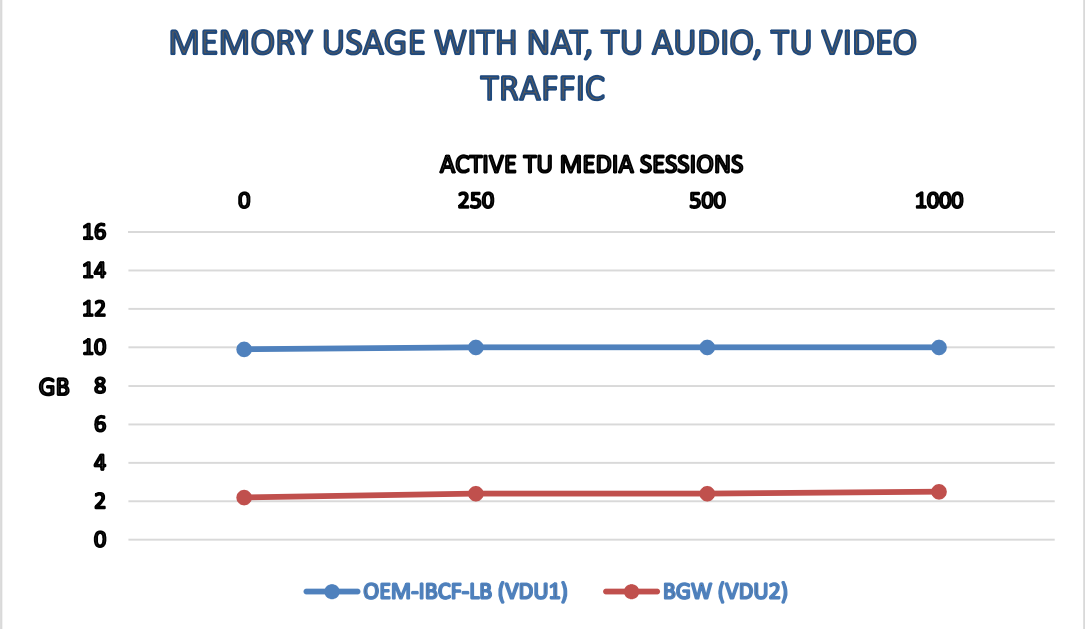


Figure 59 – Memory Usage Load Curve (audio traffic with NAT ,video and audio traffic with Transcoding)

10.6.1.3. vSBC - Network Throughput Load Curves

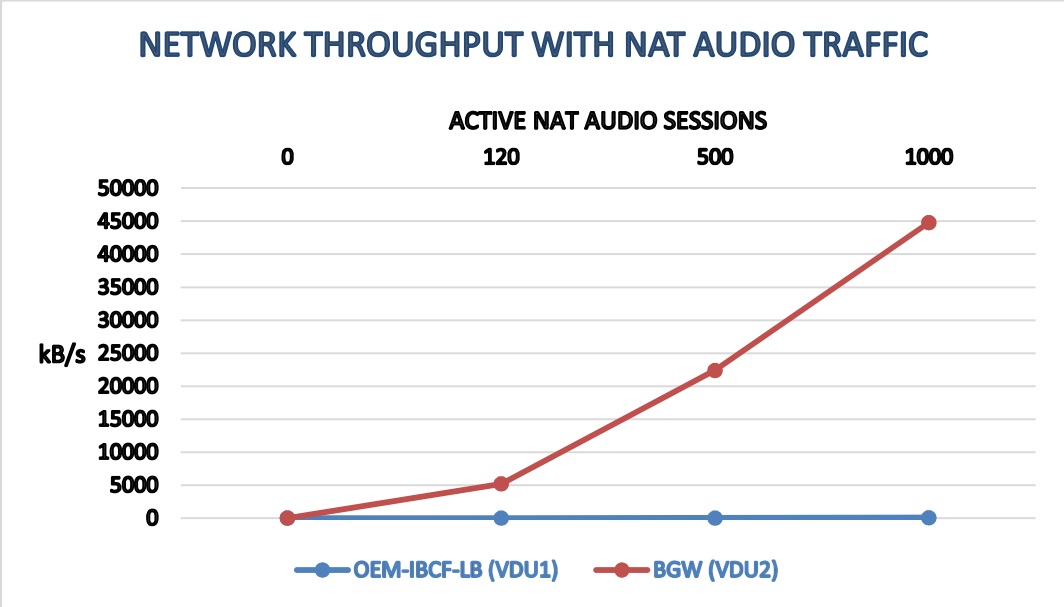


Figure 60 – Network Throughput Load Curve (audio traffic with NAT)

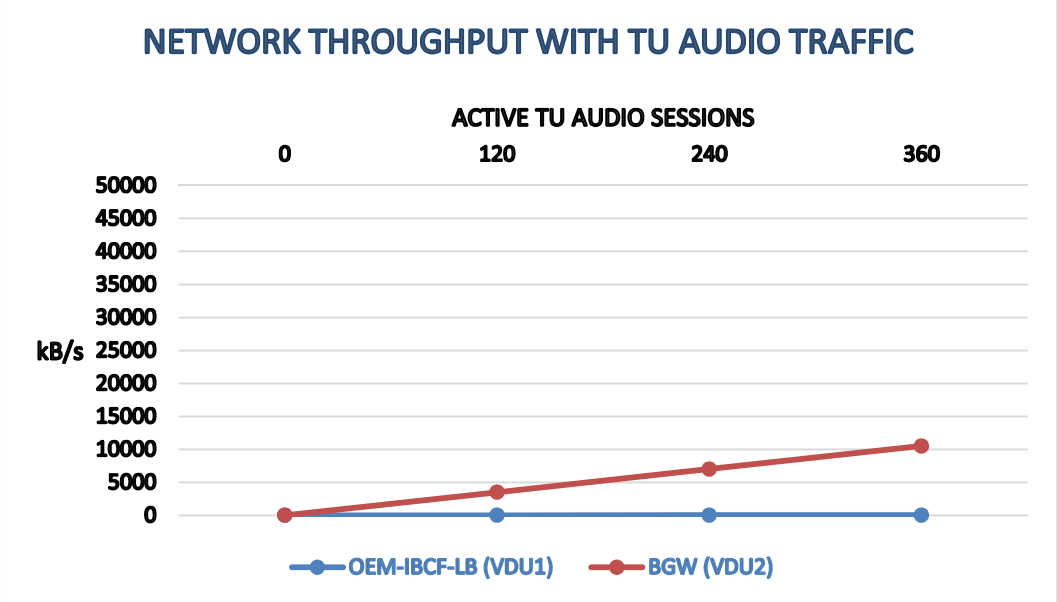


Figure 61 – Network Throughput Load Curve (audio traffic with Transcoding)

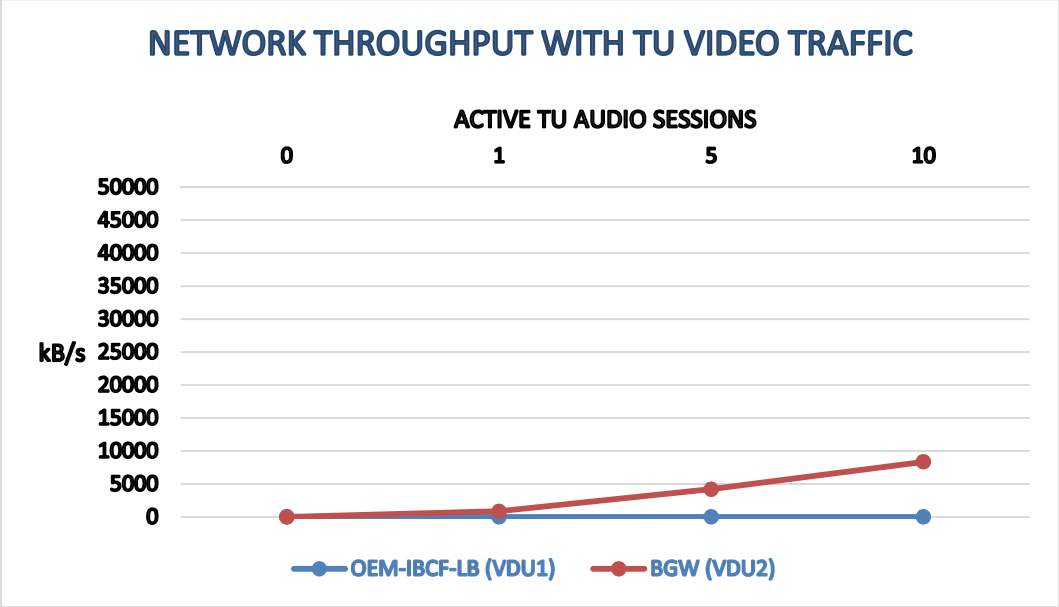


Figure 62 – Network Throughput Load Curve (video traffic with Transcoding)

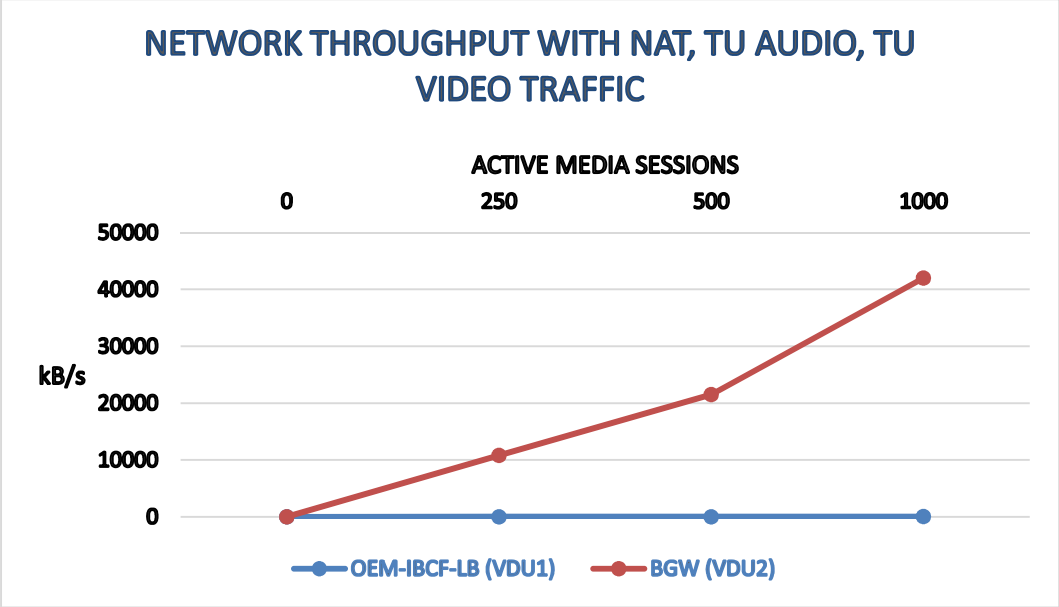


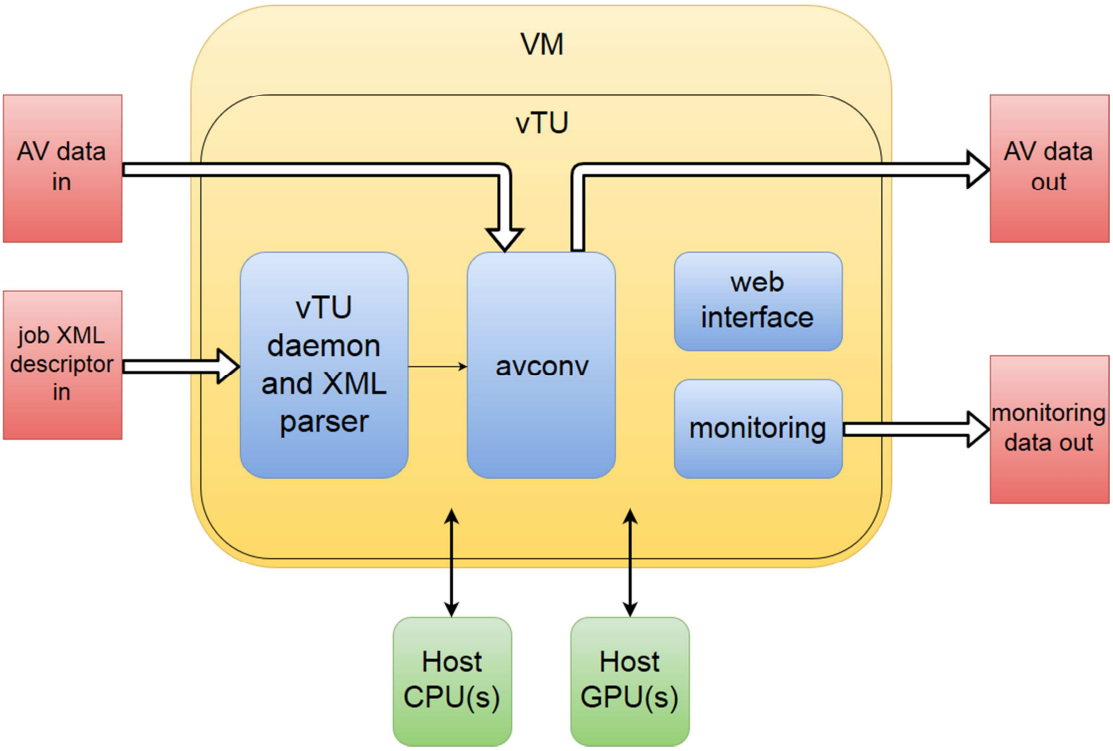
Figure 63 – Network Throughput Load Curve (audio traffic with NAT, video and audio traffic with Transcoding)

11. ANNEX B: VTU PERFORMANCE CHARACTERIZATION

11.1. vTU Test Environment

11.1.1. vTU Architecture

The basic architecture of the virtual Transcoding Unit is depicted in the following picture:



The vTU consist of a single vdu running:

- the web interface, which exposes the user accessible folders and all the monitoring parameters (via Graphana interface);
- the vTUdaemon, which manages the transcoding tasks;
- one or more instances of libav's avconv, the actual transcoder

11.1.2. vTU installation server

The testbed is composed by a single custom made server having the following specifications:

- motherboard Intel S2600 CW
- dual Intel Xeon CPU E5-2620 v3 hex-core running at 2.40GHz
- 64GB DDR3 ram
- NVidia GeForce GTX980 (2480 CUDA cores, 4GB GDDR5 memory)
- Ubuntu Linux 14.04 LTS

11.1.3. vTU resource utilization

Reference tests have been run without the virtualization layer, in order to fully evaluate the speed-up given by the GPU hardware acceleration. That means that this instance of the vTU has access to all the 12 cores, 64 GB of RAM and gains exclusive access to the only GPU card.

The lack of the virtualization layer does not void the validity of the tests, since it has been proved that performance loss in virtualized environment having KVM as hypervisor is negligible, even for GPU-passthrough configurations.

11.2. vTU Acceptance Criteria

Despite being used as basic evaluation parameters for the acceptance criteria in most of the VNFs, it makes little to no sense using cpu load and network throughput as estimators for the vTU performances.

Being video transcoding a quite cpu intensive task, while evaluating the vTU performances in non-virtualized environment it has been reported that **CPU load** rarely went below 100%, even for low resolution and low-bitrate video streams.

Also, **network throughput** is not a good estimator since transcoding tasks are often performed keeping a constant bitrate which is user-definable, therefore bandwidth requirements may be estimated and known in advance by the user.

Also, **memory** requirements do not vary between virtualized and non-virtualized environment, since each transcoding task uses a well-defined amount of memory for storing the required framebuffers. The required amount of memory is allocated at the beginning of each transcoding task and does not change while the task is running; moreover, the total amount of required memory is only dependent by the input and output file resolution, therefore may be estimated and known in advance.

Instead, in real-time video encoding and transcoding, the peak and/or average **number of processed frames per second** is more meaningful regarding the performance of the task.

In order to evaluate the vTU performance and identify an acceptance ratio for the average processed frames per second, several encodings have been made over increasing number concurrent sessions. We considered the most CPU demanding encoding which also supports the GPU hardware acceleration, that is VP8 encoding from raw YUV to webm (VP8 encoded).

The avconv used in the vTU is based on version 11, and it is slightly modified in order to exploit the GPU accelerated libvpx VP8/9 encoding / decoding libraries.

Source files were locally stored in order not to introduce latencies due to high bandwidth usage. Also, all the encoding sessions were targeted to obtain the best quality.

The following table summarizes the achieved FPS for 7 different encodings; all the concurrent session were single-treaded and pinned on a single core; values in the "total FPS" column are calculated as #frames * "total concurrent sessions" / encoding time. Since only one core performs the concurrent encodings, constant framerate is expected.

Filename and resolution	TOTAL CONCURRENT SESSIONS	Total FPS (CPU only)	Total FPS (CPU and GPU)
Old Town Cross FullHD (1920 x 1080) 100 frames	1 2 3 4 6 8 10 12 14	0.89 0.89 0.89 0.89 0.88 0.87 0.87 0.87 0.86	1.03 1.03 1.03 1.03 1.03 1.05 1.03 1.02 1.02
Pedestrian FullHD (1920 x 1080) 100 frames	1 2 3 4 6 8 10 12 14	1.81 1.80 1.80 1.80 1.78 1.76 1.75 174 1.73	2.03 2.07 2.08 2.03 2.15 2.09 2.08 2.04 2.05
Rush Hour FullHD (1920 x 1080) 100 frames	1 2 3 4 6 8 10 12 14	2.06 2.06 2.06 2.06 2.04 2.02 2.00 1.99 1.99	2.31 2.36 2.36 2.34 2.35 2.34 2.33 2.35 2.33
Tractor FullHD (1920 x 1080) 100 frames	1 2 3 4 6 8 10 12 14	1.52 1.52 1.51 1.49 1.48 1.47 1.47 1.46 1.46	1.99 1.95 2.01 1.93 1.86 1.90 1.89 1.89 1.89
Crowd Run 4K (3860 x 2140) 50 frames	1 2 3 4 6 8 10	0.59 0.59 0.58 0.58 0.57 0.56 0.56	0.67 0.69 0.67 0.67 0.66 0.66 0.66

	12 14	0.56 0.56.	0.66 0.65
Ducks Take Off 4K (3860 x 2140) 50 frames	1 2 3 4 6 8 10 12 14	0.80 0.80 0.80 0.79 0.77 0.76 0.76 0.76 0.76	0.97 0.98 0.98 0.98 0.96 0.96 0.95 0.95 0.95
In To Tree 4K (3860 x 2140) 50 frames	1 2 3 4 6 8 10 12 14	0.82 0.82 0.82 0.81 0.80 0.79 0.79 0.79 0.79	1.05 1.08 1.07 1.07 1.07 1.07 1.07 1.06 1.06

The following table summarizes the achieved FPS for same encodings of the previous table; this time, all the concurrent session were single-treaded, but each thread were pinned to a different core, except for the 14 concurrent sessions, due to lack of free cores (tests were run on a dual hex-core system); again, values in the "total FPS" column are calculated as #frames * "total concurrent sessions" / encoding time.

Filename and resolution	**TOTAL CONCURRENT SESSIONS**	**Total FPS (CPU only)**	**Total FPS (CPU and GPU)**
Old Town Cross FullHD (1920 x 1080)	1 2 3 4 6 8 10 12 14	0,89 1,76 2,56 3,29 4,94 6,61 7,71 9,25 6,51	1,03 2,01 2,66 3,51 5,30 6,97 8,23 9,66 7,34
Pedestrian FullHD (1920 x 1080)	1 2 3 4 6 8 10 12 14	1,81 3,58 5,19 6,65 10,03 13,36 15,56 18,67 13,41	2,04 3,98 5,26 6,90 10,46 13,27 16,35 18,82 15,01
Rush Hour	1	2,07	2,31

FullHD (1920 x 1080)	2	4,08	4,54
	3	5,92	6,20
	4	7,60	8,10
	6	11,43	11,96
	8	15,28	15,74
	10	17,75	18,20
	12	21,17	21,97
	14	15,64	16,98
Tractor FullHD (1920 x 1080)	1	1,53	1,99
	2	3,02	3,87
	3	4,35	4,78
	4	5,61	6,27
	6	8,46	9,12
	8	11,27	12,01
	10	13,13	13,99
	12	15,72	16,72
	14	11,42	13,13
Crowd Run 4K (3860 x 2140)	1	0,59	0,67
	2	1,16	1,31
	3	1,67	1,74
	4	2,13	2,23
	6	3,22	3,32
	8	4,30	4,36
	10	4,99	5,03
	12	5,99	5,98
	14	4,38	4,62
Ducks Take Off 4K (3860 x 2140)	1	0,80	0,97
	2	1,58	1,88
	3	2,26	2,55
	4	2,89	3,25
	6	4,36	4,89
	8	5,73	6,35
	10	6,72	7,36
	12	8,04	8,72
	14	5,81	6,52
In To Tree 4K (3860 x 2140)	1	0,41	0,52
	2	0,81	1,03
	3	1,17	1,41
	4	1,49	1,80
	6	2,25	2,70
	8	3,00	3,56
	10	3,49	4,02
	12	4,18	4,81
	14	3,03	3,74

The acceptance rate will be the 90% of minimum FPS per core achieved on the tests.

11.3. vTU Plan and Design Test

Test in virtualized environment were designed to match at the best of our possibility the non-virtualized experiment run on non-virtualized machine. The same version of

the vTU and os (including the same NVidia video driver version) were used, as well as the same batch of input video files.

The following table summarizes the test performed in the Demokritos testbed:

Filename and resolution	Assigned VCPUs	TOTAL CONCURRENT SESSIONS	Total FPS (CPU only)	Total FPS (CPU + GPU)
All of the above	1	1 2 3 4	To be collected	To be collected
All of the above	2	1 2 3 4	To be collected	To be collected
All of the above	3	1 2 3 4	To be collected	To be collected
All of the above	4	1 2 3 4	To be collected	To be collected

11.4. vTU Test Enviroments

Hardware configuration of the host machine in the Demokritos testbed is slightly different than the server used in the first experiments:

- Intel i7-4790 quad-core running at 3.60GHz
- 16GB DDR3 memory
- NVidia Quadro M4000 (1664 CUDA cores, 8GB GDDR5 memory)
- Ubuntu Linux 14.05 LTS
- KVM Hypervisor
- Openstack
- VM: variable number of vCPU, 4GB RAM, 20GB Hdd, PCI pass-through enabled for bypassing the virtualization layer while accessing the graphic accelerator

11.5. vTU Test Execution

Below, the summary of the test execution:

Filename and resolution	Assigned VCPUs	TOTAL CONCURRENT SESSIONS	Total FPS (CPU only)	Total FPS (CPU and GPU)
Old Town Cross FullHD (1920 x 1080)	1	1 2 3 4	1.14 1.16 1.15 1.16	1.23 1.30 1.29 1.28
Pedestrian FullHD (1920 x 1080)	1	1 2 3 4	2.19 2.22 2.33 2.30	2.57 2.58 2.58 2.64
Rush Hour FullHD (1920 x 1080)	1	1 2 3 4	2.48 2.65 2.64 2.64	2.72 2.86 2.97 2.84
Tractor FullHD (1920 x 1080)	1	1 2 3 4	1.88 2.00 1.93 1.95	2.21 2.43 2.32 2.32
Crowd Run 4K (3860 x 2140)	1	1 2 3 4	0.70 0.72 0.72 0.63	0.75 0.79 0.80 0.81
Ducks Take Off 4K (3860 x 2140)	1	1 2 3 4	0.91 0.95 0.90 0.95	1.10 1.15 1.12 1.18
In To Tree 4K (3860 x 2140)	1	1 2 3 4	0.50 0.52 0.52 0.51	0.60 0.64 0.65 0.66

Filename and resolution	Assigned VCPUs	TOTAL CONCURRENT SESSIONS	Total FPS (CPU only)	Total FPS (CPU and GPU)
Old Town Cross FullHD (1920 x 1080)	2	1 2 3 4	1.14 2.24 2.04 2.10	1.23 2.35 2.42 2.41
Pedestrian FullHD (1920 x 1080)	2	1 2 3 4	2.19 4.01 4.37 4.44	2.57 4.81 4.83 4.93
Rush Hour FullHD (1920 x 1080)	2	1 2 3 4	2.48 4.97 5.13 4.93	2.72 5.41 5.52 5.76
Tractor FullHD	2	1 2 3	1.88 3.47 3.43	2.21 4.20 4.38

(1920 x 1080)		4	3.66	4.09
Crowd Run 4K (3860 x 2140)	2	1 2 3 4	0.70 1.39 1.39 1.38	0.75 1.40 1.42 1.47
Ducks Take Off 4K (3860 x 2140)	2	1 2 3 4	0.91 1.92 1.81 1.81	1.10 2.05 2.12 2.17
In To Tree 4K (3860 x 2140)	2	1 2 3 4	0.50 1.00 0.99 0.99	0.60 0.64 1.11 1.21

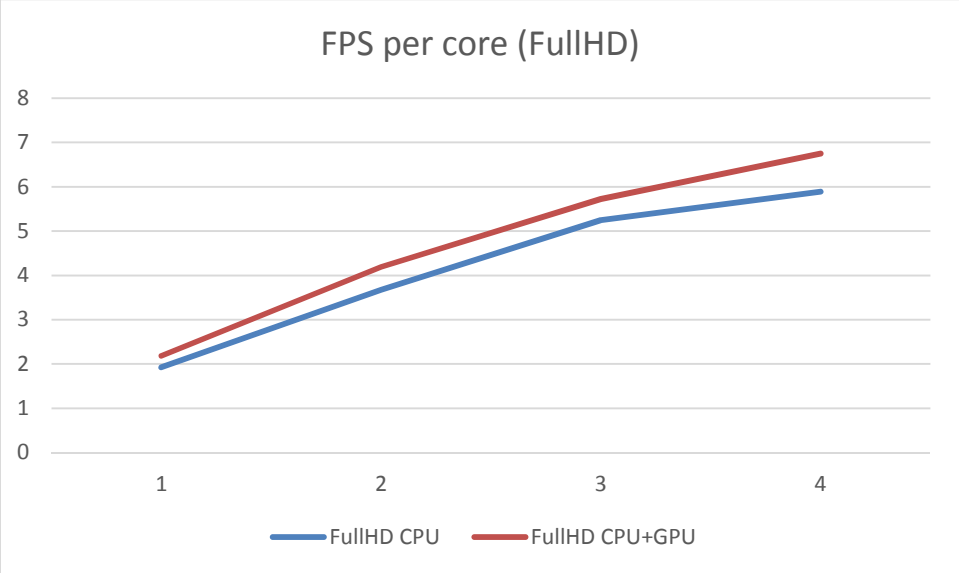
Filename and resolution	**Assigned VCPUs**	**TOTAL CONCURRENT SESSIONS**	**Total FPS (CPU only)**	**Total FPS (CPU and GPU)**
Old Town Cross FullHD (1920 x 1080)	3	1 2 3 4	1.14 2.24 3.12 2.19	1.23 2.35 3.20 2.44
Pedestrian FullHD (1920 x 1080)	3	1 2 3 4	2.19 4.01 5.89 4.31	2.57 4.81 6.15 4.61
Rush Hour FullHD (1920 x 1080)	3	1 2 3 4	2.48 4.97 6.85 4.89	2.72 5.41 7.60 5.89
Tractor FullHD (1920 x 1080)	3	1 2 3 4	1.88 3.47 5.14 3.81	2.21 4.20 5.95 4.56
Crowd Run 4K (3860 x 2140)	3	1 2 3 4	0.70 1.39 1.73 1.39	0.75 1.40 1.90 2.19
Ducks Take Off 4K (3860 x 2140)	3	1 2 3 4	0.91 1.92 2.28 1.87	1.10 2.05 2.88 2.08
In To Tree 4K (3860 x 2140)	3	1 2 3 4	0.50 1.00 1.40 0.95	0.60 0.64 1.56 1.18

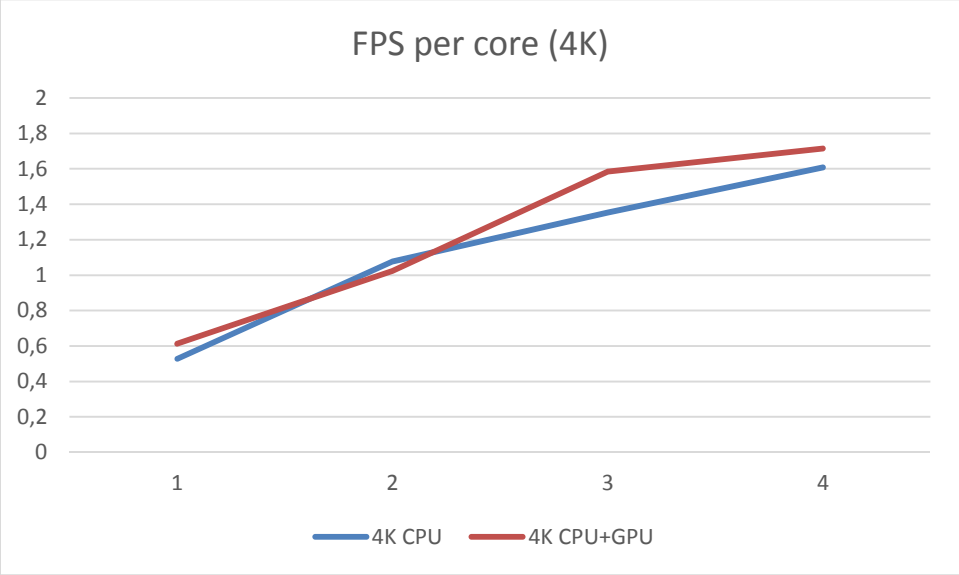
Filename and resolution	**Assigned VCPUs**	**TOTAL CONCURRENT SESSIONS**	**Total FPS (CPU only)**	**Total FPS (CPU and GPU)**

Old Town Cross FullHD (1920 x 1080)	4	1 2 3 4	1.14 2.24 3.12 3.04	1.23 2.35 3.20 3.73
Pedestrian FullHD (1920 x 1080)	4	1 2 3 4	2.19 4.01 5.89 6.78	2.57 4.81 6.15 7.92
Rush Hour FullHD (1920 x 1080)	4	1 2 3 4	2.48 4.97 6.85 7.76	2.72 5.41 7.60 9.09
Tractor FullHD (1920 x 1080)	4	1 2 3 4	1.88 3.47 5.14 5.99	2.21 4.20 5.95 6.27
Crowd Run 4K (3860 x 2140)	4	1 2 3 4	0.70 1.39 1.73 1.85	0.75 1.40 1.90 2.02
Ducks Take Off 4K (3860 x 2140)	4	1 2 3 4	0.91 1.92 2.28 2.91	1.10 2.05 2.88 3.13
In To Tree 4K (3860 x 2140)	4	1 2 3 4	0.50 1.00 1.40 1.67	0.60 0.64 1.56 1.71

11.6. vTU Results

Below the curve of the average achievable FPS per core for each video resolution:





12. VNF DESCRIPTOR EXAMPLES

The example below provides the case of a single VNF, composed of one VNFC(VDU). The VNF functionality is basically a PROXY server with content filtering and user groups support.


```
{
  "provider_id": 4,
  "vdu": [
    {
      "resource_requirements": {
        "network_interface_bandwidth_unit": "",
        "hypervisor_parameters": {
          "version": "10002|12001|2.6.32-358.el6.x86_64",
          "type": "QEMU-KVM"
        },
        "memory_unit": "GB",
        "network_interface_card_capabilities": {
          "SR-IOV": true,
          "mirroring": false
        },
        "storage": {
          "size_unit": "GB",
          "persistence": false,
          "size": 32
        },
        "network_interface_bandwidth": "",
        "platform_pcie_parameters": {
          "SR-IOV": true,
          "device_pass_through": true
        },
        "vcpus": 1,
        "vswitch_capabilities": {
          "version": "2.0",
          "type": "ovs",
          "overlay_tunnel": "GRE"
        },
        "data_processing_acceleration_library": "",
        "memory": 1,
        "memory_parameters": {
          "large_pages_required": false,
          "numa_allocation_policy": ""
        },
        "cpu_support_accelerator": "AES-NI"
      },
      "vm_image": "http://10.10.1.167:8080/NFS/files/PXaaS-180216.qcow2",
```

```
    "vm_image_format": "qcow2",
    "networking_resources": "",
    "monitoring_parameters_specific": [
     {
       "metric": "httpnum",
       "unit": "INT",
       "desc": "Number of HTTP requests received by Squid"
     },
     {
       "metric": "hits",
       "unit": "%",
       "desc": "Cache hits percentage of all requests for the last 5 minutes"
     },
     {
       "metric": "hits_bytes",
       "unit": "%",
       "desc": "Cache hits percentage of bytes sent for the last 5 minutes"
     },
     {
       "metric": "memoryhits",
       "unit": "%",
       "desc": "Memory hits percentage for the last 5 minutes (hits that are logged as
TCP_MEM_HIT)"
     },
     {
       "metric": "diskhits",
       "unit": "%",
       "desc": "Disk hits percentage for the last 5 minutes (hits that are logged as
TCP_HIT)"
     },
     {
       "metric": "cachediskutilization",
       "unit": "%",
       "desc": "Cache disk utilization"
     },
     {
       "metric": "cachememkutilization",
       "unit": "%",
       "desc": "Cache memory utilization"
     },
     {
       "metric": "usernum",
       "unit": "INT",
       "desc": "Number of users accessing the proxy"
     },
     {
       "metric": "cpuusage",
```

```
        "unit": "%",
        "desc": "CPU consumed by Squid for the last 5 minutes"
       }
     ],
     "id": "vdu0",
     "alias": "proxy",
     "controller": true,
     "connection_points": [
      {
       "vlink_ref": "vl0",
       "id": "CPzc4j"
      },
      {
       "vlink_ref": "vl1",
       "id": "CPv41w"
      },
      {
       "vlink_ref": "vl2",
       "id": "CP796o"
      }
     ],
     "monitoring_parameters": [
      {
       "metric": "httpnum",
       "unit": "INT",
       "desc": "Number of HTTP requests received by Squid"
      },
      {
       "metric": "hits",
       "unit": "%",
       "desc": "Cache hits percentage of all requests for the last 5 minutes"
      },
      {
       "metric": "hits_bytes",
       "unit": "%",
       "desc": "Cache hits percentage of bytes sent for the last 5 minutes"
      },
      {
       "metric": "memoryhits",
       "unit": "%",
       "desc": "Memory hits percentage for the last 5 minutes (hits that are logged as
TCP_MEM_HIT)"
      },
      {
       "metric": "diskhits",
       "unit": "%",
```

```
      "desc": "Disk hits percentage for the last 5 minutes (hits that are logged as TCP_HIT)"
    },
    {
      "metric": "cachediskutilization",
      "unit": "%",
      "desc": "Cache disk utilization"
    },
    {
      "metric": "cachememkutilization",
      "unit": "%",
      "desc": "Cache memory utilization"
    },
    {
      "metric": "usernum",
      "unit": "INT",
      "desc": "Number of users accessing the proxy"
    },
    {
      "metric": "cpuusage",
      "unit": "%",
      "desc": "CPU consumed by Squid for the last 5 minutes"
    },
    {
      "metric": "processes_blocked",
      "unit": "INT",
      "desc": "Blocked Processes"
    },
    {
      "metric": "processes_paging",
      "unit": "INT",
      "desc": "Paging Processes"
    },
    {
      "metric": "processes_running",
      "unit": "INT",
      "desc": "Running Processes"
    },
    {
      "metric": "processes_sleeping",
      "unit": "INT",
      "desc": "Sleeping Processes"
    },
    {
      "metric": "processes_stopped",
      "unit": "INT",
      "desc": "Stopped Processes"
```

```
      },
      {
        "metric": "processes_zombie",
        "unit": "INT",
        "desc": "Zombie Processes"
      }
    ],
    "vm_image_md5": "90f23ff2b1146e0a6a088ac1a96b7975",
    "scale_in_out": {
      "minimum": 1,
      "maximum": 1
    }
  }
],
"name": "PXaaS",
"created_at": "2016-03-09T09:55:14Z",
"modified_at": "2016-03-09T09:55:14Z",
"vlinks": [
  {
    "leaf_requirement": "Unlimited",
    "connectivity_type": "E-LAN",
    "vdu_reference": [
      "vdu0"
    ],
    "external_access": true,
    "connection_points_reference": [
      "CPzc4j",
      "CPzc4j"
    ],
    "access": true,
    "alias": "management",
    "root_requirement": "Unlimited",
    "dhcp": true,
    "id": "vl0",
    "qos": ""
  },
  {
    "leaf_requirement": "Unlimited",
    "connectivity_type": "E-LINE",
    "vdu_reference": [
      "vdu0"
    ],
    "external_access": true,
    "connection_points_reference": [
      "CPv41w",
      "CPv41w"
    ],
```

```
    "access": false,
    "alias": "data_in",
    "root_requirement": "Unlimited",
    "dhcp": false,
    "id": "vl1",
    "qos": ""
   },
   {
    "leaf_requirement": "Unlimited",
    "connectivity_type": "E-LINE",
    "vdu_reference": [
     "vdu0"
    ],
    "external_access": true,
    "connection_points_reference": [
     "CP796o",
     "CP796o"
    ],
    "access": false,
    "alias": "data_out",
    "root_requirement": "Unlimited",
    "dhcp": false,
    "id": "vl2",
    "qos": ""
   }
  ],
  "trade": true,
  "descriptor_version": "1",
  "deployment_flavours": [
   {
    "vdu_reference": [
     "vdu0"
    ],
    "constraint": "",
    "flavour_key": "gold",
    "vlink_reference": [
     "vl0",
     "vl1",
     "vl2"
    ],
    "id": "flavor0",
    "assurance_parameters": [
     {
      "violation": [
       {
        "interval": 360,
        "breaches_count": 2
```

```
          }
         ],
         "value": 100,
         "penalty": {
          "type": "Discount",
          "expression": 20,
          "validity": "P1D",
          "unit": "INT"
         },
         "formula": "httpnum GT 100",
         "rel_id": "param0",
         "id": "httpnum",
         "unit": "INT"
        }
       ]
     }
   ],
   "version": "1",
   "vnf_lifecycle_events": [
    {
     "authentication_username": "root",
     "authentication":
"AAAAB3NzaC1yc2EAAAABIwAAAQEAklOUpkDHrfHY17SbrmTIpNLTGK9Tjom/BWDS
U\nGPl+nafzlHDTYW7hdI4yZ5ew18JH4JW9jbhUFrviQzM7xlELEVf4h9lFX5QVkbPppSw
g0cda3\nPbv7kOdJ/MTyBlWXFCR+HAo3FXRitBqxiX1nKhXpHAZsMciLq8V6RjsNAQwd
sdMFvSlVK/7XA\nt3FaoJoAsncM1Q9x5+3V0Ww68/eIFmb1zuUFljQJKprrX88XypNDvjY
Nby6vw/Pb0rwert/En\nmZ+AW4OZPnTPI89ZPmVMLuayrD2cE86Z/il8b+gw3r3+1nKa
tmIkjn2so1d01QraTlMqVSsbx\nNrRFi9wrf+M7Q==",
     "authentication_port": 22,
     "vnf_container": "/opt/proxy",
     "events": {
      "start": {
       "command": "service proxy start",
       "template_file": "{}",
       "template_file_format": "JSON"
      },
      "stop": {
       "command": "service proxy stop",
       "template_file": "{}",
       "template_file_format": "JSON"
      }
     },
     "flavor_id_ref": "flavor0"
    }
   ],
   "billing_model": {
    "price": {
```

```
      "min_per_period": 10,
      "max_per_period": 20,
      "setup": 20,
      "unit": "EUR"
    },
    "model": "PAYG",
    "period": "P7D"
  },
  "provider": "TEIC",
  "release": "T-NOVA",
  "type": "vPXAAS",
  "id": 1004,
  "description": "PXaaS Desc"
}
```