SEVENTH FRAMEWORK PROGRAMME

TNOVA

NETWORK FUNCTIONS AS-A-SERVICE
OVER VIRTUALISED INFRASTRUCTURES

GRANT AGREEMENT NO. 619520

Deliverable D5.01

Interim Report on Network Functions and associated framework

Editor Paolo Comi (Italtel)

Contributors Ahmed Abujoda (LUH), Aurora Ramos (ATOS), Bruno Miguel Parreira, José Bonnet (PTInS), Dora Christofi, Savvas Charalambides (PrimeTel), Enzo Figini, Pietro Paglierani (Italtel), Federico Pedersini , George Xilouris, Michail-Alexandros Kourtis (NCSRD), Giuliano Grossi (UniMI), Nicolas Herbaut (VIO), Yacine Rebahi (Fokus)

Version 1.0

Date December 19th, 2014

Executive Summary

This document reports the initial stage of the development of the Virtualised Network Functions (VNFs), as well as the Network Function Framework that is the conceptual element of the T-NOVA system devoted to the definition of the structure and behaviour of the Virtual Network Functions (VNFs). It includes also the description of the Function Store that is the container of the VNF images as well as the information provided by software developers and function providers offering their VNFs to the T-NOVA marketplace.

This report contains the research, design and implementation results and ideas developed in WP5 "Network Functions" work-package of T-NOVA project. This work-package gets inputs from WP2 "System Specification" and in particular from T2.5 "Specification of Network Function framework". WP5 has relationships with WP3 "Orchestrator Platform" mainly because of the VNFs' interactions with T-NOVA orchestrator system. Other relationships are with WP6 "T-NOVA Marketplace" mainly because of the Function Store that is the gate for providing T-NOVA marketplace with suitable VNFs. WP5 has also interactions with WP4 "Infrastructure Virtualisation and Management".

The VNFs developed in WP5 are:

- Security Appliance (SA)
- Session Border Controller (SBC)
- Traffic Classification (DPI)
- Home Gateway (HG)

These virtual network functions will be used for proving the effectiveness of T-NOVA concept. Some of them represent real implementations of first stage prototypes interesting for T-NOVA industrial partners willing to address the NFV market.

Table of Contents

1. INTRODUCTION

1.1. Motivation, objectives and scope

The design and implementation work of WP5 benefits from the extensive work done in WP2 that designed the overall system and detailed its decomposition.

Therefore, even if this document is produced at the initial stage of T-NOVA development, the system design, the objectives and the interactions among system components are almost clear.

We believe reporting the current status of the development will provide benefits to a clear understanding of the actual implementation of the concepts studied in the first phase of T-NOVA project and provides feedbacks for the refinement of the design and specifications of WP2. Moreover, it is a good tool for detailing the interfaces with all the other components of T-NOVA system.

1.2. Document Structure

This document is organized in a modular way following the structure of the tasks composing WP5. Each task is described in a dedicated section.

Section 2 “Function Packaging and Repository” deals with the Network Function Store design and implementation. (Task 5.1)

Section 3 “Function Deployment, Configuration and Management” describes a middleware layer realizing the Network Function API for VNF configuration and management. (Task 5.2)

Section 4 “Development of Network Functions” contains the actual description of the VNFs developed inside of the work-package. The introduction of this section provides general description and design choices applicable to any VNF that is part of T-NOVA ecosystem. (Task 5.3)

Section 5 “Conclusions and Future Work” summarizes the results of WP5 work as described in the previous sections. Future work is meaningful as this document is produced in the initial stage of T-NOVA project development.

Finally, the usual chapters for the list of acronyms and references complete the document.

2. FUNCTION PACKAGING AND REPOSITORY

2.1. Introduction

This section is dedicated to the T-NOVA Network Function Store (NF Store) design and implementation. The NF Store is mainly a repository for the VNFs and their metadata. It contains the virtual machines (VMs) software images and the metadata descriptor composing each VNF. The structure of a VNF is described in [D2.41].

The diagram in Figure 2-1 provides a very high level architectural description of the relationships of NF Store and VNFs with the other elements of T-NOVA architecture.

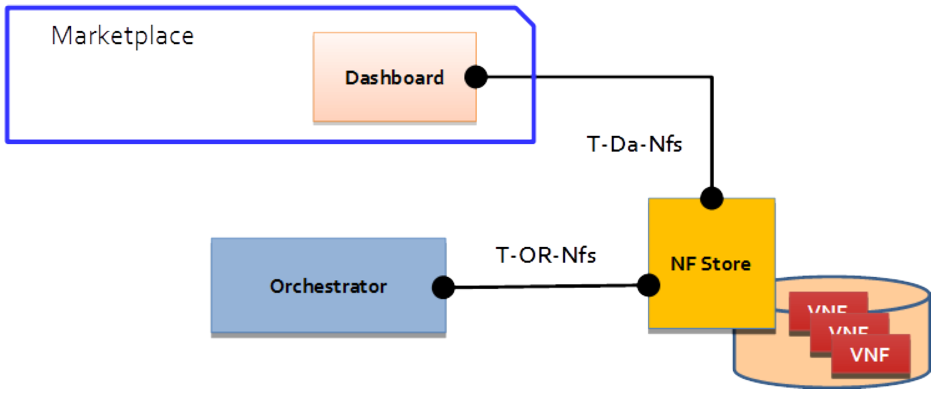

Figure 2-1. NF Store high level architecture

The interfaces T-Da-Nfs and T-OR-Nfs describe the interaction of NF Store with the rest of T-NOVA system.

Software developers can upload VNF images into the NF Store accompanied with metadata descriptor containing both technical and business related information, such as the business description of the cost of using such VNF.

The description of the VNFs in the NF Store is made available to the T-NOVA Marketplace [D2.41] and [D6.01]. Whenever a VNF is chosen for being part of a service, the T-NOVA orchestrator will get the VNF image for deploying and executing it over the virtualized execution environment.

The NF Store is part of the T-NOVA logical infrastructure. With respect to the overall architecture definition [D2.21], a T-NOVA system can be composed by more than one datacenters where the VNFs can actually run. In case of multi-datacenter deployment the NF Store is not constrained to be part of a specific one, while the VNFs stored in it will be. This concept is graphically represented in Figure 2-2 where the NF Store serves all the datacenters but it logically belongs to the common elements of T-NOVA infrastructure.

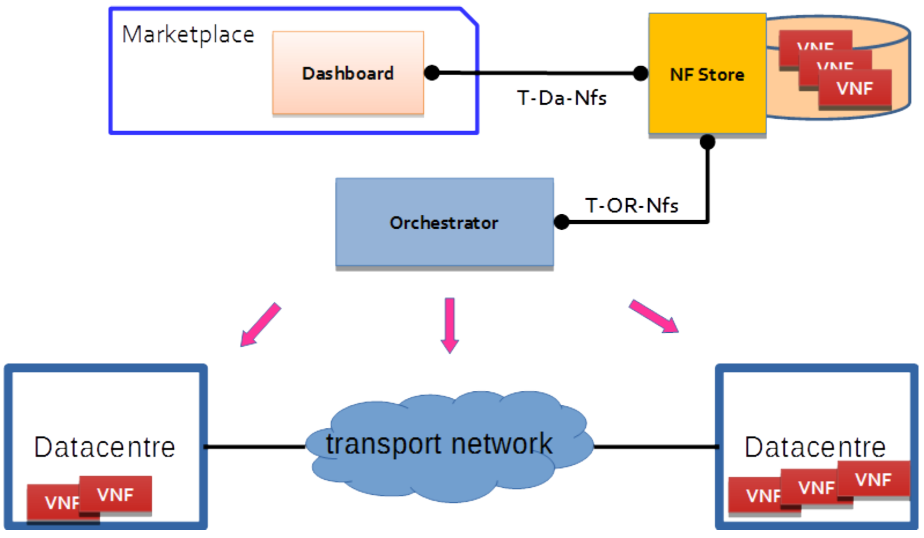

Figure 2-2. NF Store logical deployment view

2.2. State-of-the-art

Many technologies can be considered for NF Store implementation, also because the need of a repository for virtual machine images is common to many software architectures and projects. Nevertheless, none of the available solutions we considered fits with all T-NOVA NF Store requirements.

Our solution availability analysis involved the following three different approaches: (1) look at currently available solutions for image repository; (2) investigate OpenStack approach to the problem; (3) identify the basic components we can use for fresh implementation. These options are described in the following chapters.

2.2.1. Image repositories

Many solutions are currently available as open-source software implementing image repositories. Some of them are:

DSpace	DSpace [DSpace] is used by educational, government, private and commercial institutions by more than 1000 organizations that are currently using it in a production or project environment. It is also used by museums, state archives, state and National Libraries, journal repositories, consortiums, and commercial companies to manage their digital assets. The DSpace application can recognize and manage a large number of file format and mime types. It also provides simple file format registry where you can register any unrecognized format, so that it can be identified in the future. This software can be customized in many ways like user interface, metadata, browse and search, authentication, compatibility and database to be used.
Fedora	Fedora Repository software [Fedora] is used by more than 300 institutions for storing, managing, and accessing digital content. Fedora defines a set of abstractions for expressing digital objects, asserting

	relationships among digital objects, and linking "behaviours" (i.e. services) to digital objects. It can store all types of content and metadata adopting many storage options (e.g. database, file system, …). It provides access via Web API (REST/SOAP) and supports disaster recovery and data migration. It is able to scale to millions of objects.
Invenio	Invenio is a free software suite [Invenio] enabling running a digital library or document repository on the web. The technology covers all aspects of digital library management from document ingestion through classification, indexing, and curation to dissemination. Documents are organised in collections, with powerful search engine, using standard MARC (MAchine-Readable Cataloging) metadata for handling articles, books, theses, photos, videos, museum objects and more.
Eprints	Eprints is a software platform [Eprints] for building high quality OAI-PMH (The Open Archives Initiative Protocol for Metadata Harvesting) compliant repositories with a flexible plugin architecture for developing extensions for import and export options, changes to the interface and new ways for users to enter the data. It gives also an 'auto completion' feature to assist in better quality metadata.

The list of available software can continue long. Most of them are meant to particular purposes like software development or digital archive offering many functions that are not needed in T-NOVA. Most important, all of them don't have the concept of VNF as a group of related VMs. Therefore, none of them can be used as-it-is.

In order to be used for T-NOVA NF Store we shall add an external component that manages the VNF as aggregation of VM images. Then, it shall report VNF updates to T-NOVA orchestrator.

2.2.2. OpenStack repositories

An interesting possibility comes from OpenStack Murano component [Murano]. It is an application catalogue which allows application developers and cloud administrators to publish cloud-ready applications. It is composed by three parts: the main part with core and engine, an agent which runs on guest VMs that executes deployment plan and a dashboard implementing a plugin for OpenStack Dashboard (see Figure 2-3).

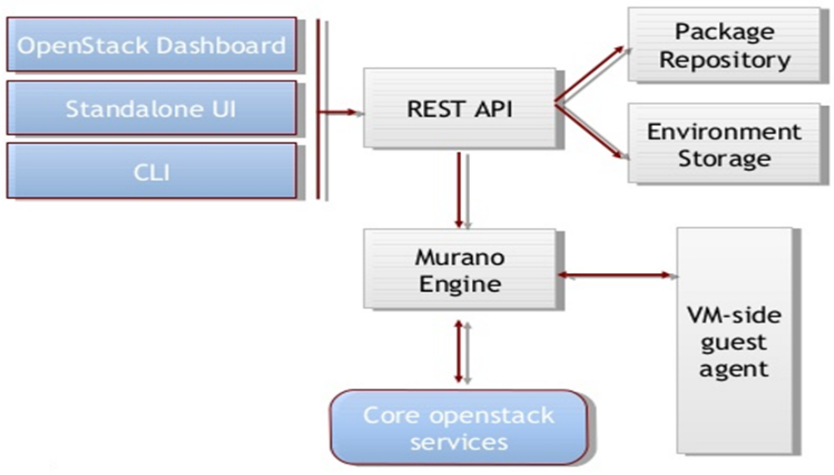

Figure 2-3. Murano functional architecture

Murano is a promising project for NF Store implementation. As T-NOVA NF Store it provides a repository for applications in a cloud environment. Nevertheless, there are some considerations that prevent Murano to be used in T-NOVA as it is:

- Lack of VNF concept as a group of VMs.
- Strong relation and integration with OpenStack. Murano extends OpenStack services providing an application catalogue and is not meant to live outside OpenStack.
- The deployment location can be different from the datacenter where the VNF will run. This will require interactions between diverse OpenStack installations.
- VMs shall include a guest agent. This introduces additional constraints with respect to T-NOVA open marketplace.

Notwithstanding, Murano project is very attracting because we believe it can evolve in the future to cope with requirements similar to T-NOVA ones. The telco Workgroup that has been created within the frame of Openstack, has already specified as a near term research item the mapping of the Openstack components to the ETSI NFV MANO architecture, as an initial step in the time plan for accommodating NFV in their agenda.

2.2.3. Web-frameworks

Considering the limits of the available solutions described above, we studied the option of fresh implementation of the NF Store. Fortunately many powerful tools are currently available mitigating the risk of re-inventing the wheel again and again. The main technologies we considered are mainly web-frameworks, databases and generic storage techniques.

Web-frameworks support web and REST interfaces. Therefore, they match the requirements about NF Store external interfaces. Moreover, they can be used for the implementation of NF Store core logic.

We grouped the available technologies based on the implementation language.

java	JAX-RS	API that provides support in creating web services according to the REST architectural pattern that uses annotations to simplify the development of web service clients and endpoints.
	Spring	application framework and inversion of control container that contains extensions for building web applications on top of the J2EE platform.
python	Django	web application framework which follows the MVC architectural pattern.
	Flask	microframework for Python based on Werkzeug and Jinja 2
	web2py	full-stack framework for rapid development of web-based applications
	TurboGears	microframework that can scales up to a fullstack solution, extensible and pluggable
php	Symfony	web application framework for Model View Controller (MVC) applications inspired by other Frameworks such as Django, and Spring
	Zend	object-oriented web application framework

Libraries and APIs such as JAX-RS shall be deployed in an application container such as [JBoss], [Tomcat], or [Jetty].

VNF metadata can be saved with VNF images, but the data can also be maintained in a database. There are lots of DBs that can be used such as [MySQL], [H2], or [PostgreSQL]. Some of them have no limits, or have a very high limit, for table size. In this case VM images can be stored in the database too. Another option is to store VM images in the file system directly leaving only metadata descriptors in the database.

2.3. Requirements

The main requirements for the Network Function Store are summarized here.

General requirements

1. NF Store is the repository for the VNF images and its metadata.
2. A VNF can be composed of x VM images and one metadata descriptor
3. The VNF metadata descriptor and all the VM images composing a VNF shall be correlated, i.e. associated to a unique identifier (VNF-id).
4. The VNF-id is part of the VNF metadata descriptor.
5. A VNF can be versioned.

Functional requirements

1. The NF Store informs the orchestrator whenever a VNF is added to or removed from the repository.

2. The NF Store supports the following interfaces: T-Da-Nfs, T-OR-Nfs.
3. T-Da-Nfs shall allow to upload the VNF metadata descriptor and each VNF VM image.
4. T-Da-Nfs shall allow to remove a VNF. Then, all VNF components are deleted from the NF Store.
5. T-OR-Nfs shall allow to download the VNF metadata descriptor and each VNF VM image.
6. The operations over the supported interfaces shall be allowed upon authentication and authorization.

Non-Functional Requirements

1. The NF Store shall provide storage capacity for a reasonable number of VNFs and VM images. These numbers belong to the set of NF Store configuration parameters that shall be provided at NF Store deployment time.
2. NF Store shall not introduce additional performance constraints beyond the available bandwidth and throughput.

2.4. Architecture

High level design of NF Store includes:

- NF repository
    - Decomposed in VNF metadata repository and VNF image(s) repository.
- NF Store manager
    - Implements the logic governing the repository and the interactions over the exposed interfaces.
- NF Store interfaces or front-ends
    - Provides interfaces for interacting with the NF repository.

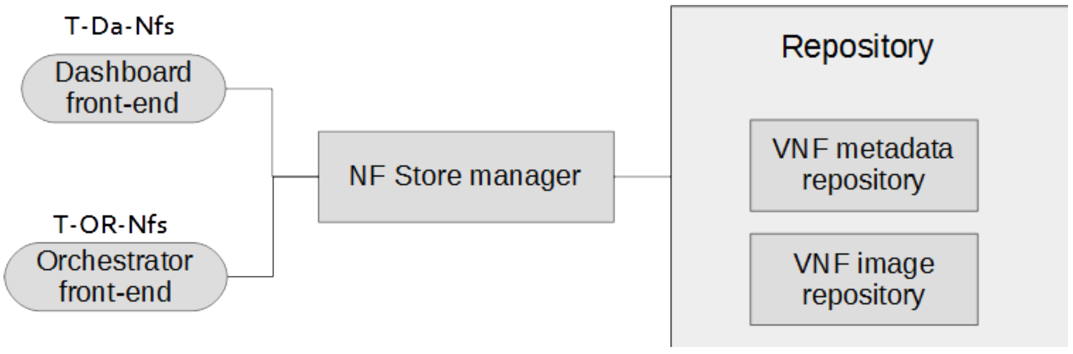

Figure 2-4. NF Store High Level Design

The NF repository is responsible for concurrent operations that are performed on the store (CRUD operations) on VM images and metadata.

The NF Store Manager block implements the interfaces to the repositories. In addition, it delegates some traits of its responsibility (like authentication and authorization access) to an external authentication and authorization (AA) module. Security mechanisms can be adopted for data exchange over these interfaces.

The NF Store provides interfaces to the orchestrator (T-Or-Nfs) and marketplace's Dashboard (T-Da-Nfs). An additional interface (not shown in the picture) for console management is foreseen.

The NF Store high level architecture (Figure 2-4) remains valid independently from the specific technology we will use to implement it. As we introduced in the analysis of the available technologies, many alternatives can be considered for NF Store implementation. Our decision has been to divide the implementation in phases.

The goal of the first phase is to provide a usable implementation in a short time also reducing risks of using interesting technologies but requiring unpredictable amount of modifications. The second phase will be driven to the exploration of the interesting research. The research activities we have identified are: (1) considering alternative implementations for studying differences in performance and reliability; (2) extending our knowledge of interesting projects such as OpenStack Murano and also considering to contribute to the open-source community.

2.4.1. Implementation design

The NF Store will be implemented by a java web-framework. The implementation architecture is represented in Figure 2-5.

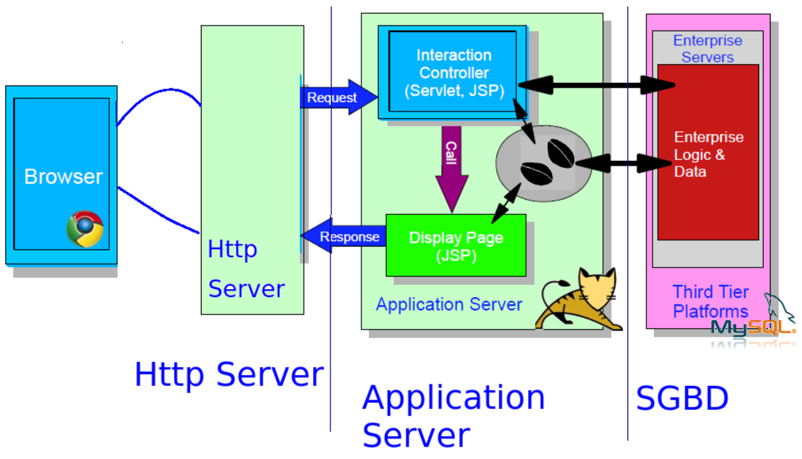

Figure 2-5. NF Store java web-framework implementation

The NF Store interfaces will be developed as a web service using java language and deployed into a servlet container such as Tomcat or Jetty. The same architecture can be implemented using python or PHP languages.

Figure 2-5 shows a standard implementation with front-end and back-end of inside the same application. In our case the front-end is part of Marketplace dashboard.

Using a single application server the http server is not necessary but it is required to implement load sharing of requests in case of two or more application servers are instantiated to increase performances (see Figure 2-6).

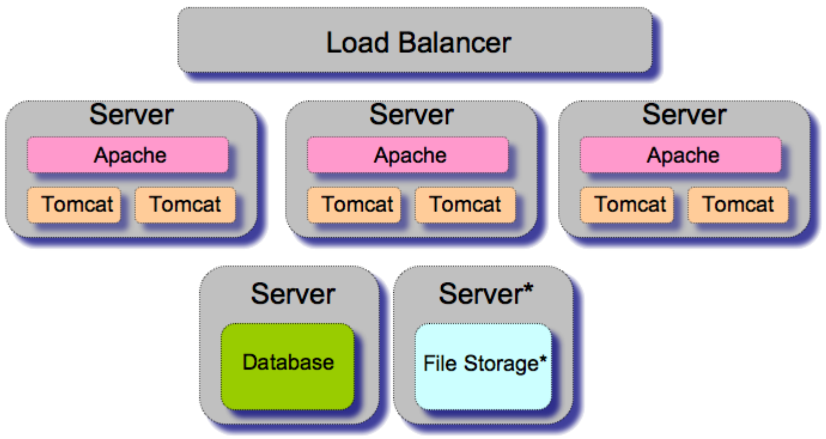

Figure 2-6. NF Store multi-server deployment

The database is used to store information about VNFs coming from metadata files while VNF VM images are saved in a file system. Saving VNF VM images in the database is an option that will be explored and evaluated.

2.5. Functional description

The Dashboard front-end provides the T-Da-Nfs interface, used by software developers to publish their VNFs. The supported operations are: publication, modification, withdrawal of a VNF and its metadata.

Considering that a VNF can be composed by several VM images, the amount of data to be transmitted can become very large. Therefore, each VM image will be transmitted separately. The VNF metadata descriptor shall contain information of each VM composing the VNF. The VNF metadata descriptor is the first file to be uploaded. The descriptor is parsed by the NF Store manager. Then it waits for the user to upload each VNF image. At the end of this process the VNF is considered uploaded.

The same policy is implemented in the orchestrator front-end over the T-Or-Nfs interface. In this case the VNF metadata descriptor is downloaded first. The VM images can be downloaded in a second time even sometime later. In the download process the NF Store does not verify that each VM image has been downloaded. This task is up to the user of the orchestrator front-end interface.

Each time a modification of the repository occurs, the NF Store triggers a message through the orchestrator front-end. It usually serves to T-NOVA orchestrator to be informed of such modification.

2.5.1. Sequence diagrams

The description of the interactions over the NF Store interfaces is described here on per interface basis.

2.5.1.1. Dashboard front-end

The T-Da-Nfs interface supports the following operations:

- publish_VNF_metadata
- publish_VNF_image
- withdraw_VNF
- get_VNF_metadata
- get_VNF_list

The diagram in figure 2-7 describes the publication procedure: publish_VNF_metadata, publish_VNF_image.

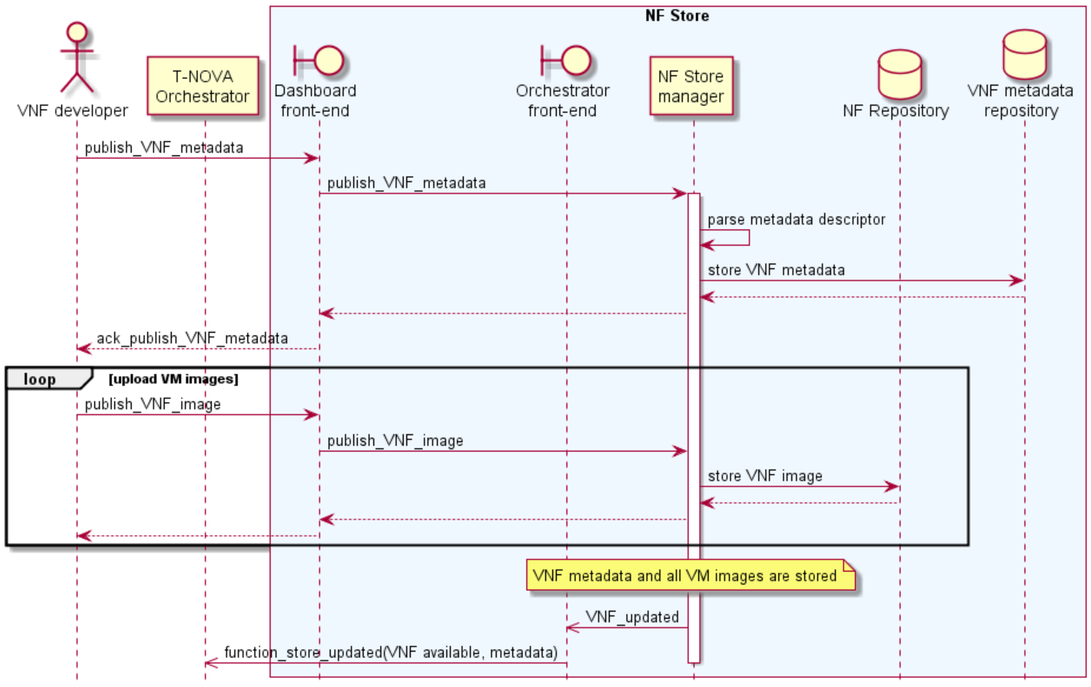

Figure 2-7. VNF publication in the NF Store

Two main attributes are used in order to identify a VNF:

1. the metadata file name
2. the data inside of metadata descriptor.

In both cases the identification should include the VNF name with version/revision. This combination must be unique into the NF Store.

The workflow for withdrawing the VNF from the NF Store is slightly different. It is represented in figure 2-8.

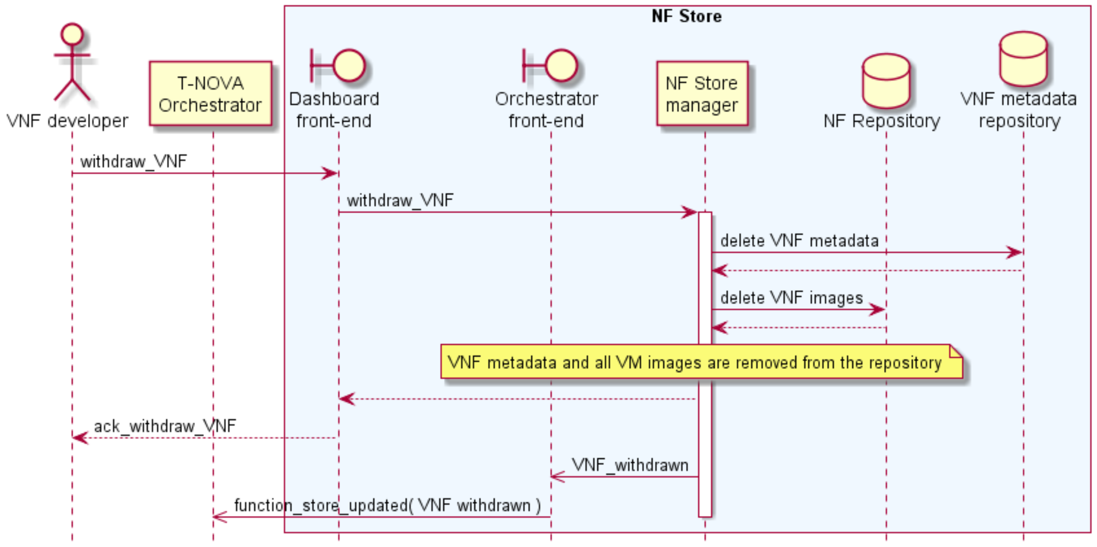

Figure 2-8. VNF withdrawal from the NF Store

The needed data (name/version/revision) to identify the VNF to be removed shall be provided by calling withdraw_VNF. The NF Store will remove all the files and data related to the VNF.

It is useful for the dashboard presentation to have a way to get information about the available VNF inside the NF Store. This list is obtained by get_VNF_list operation as described in Figure 2-9.

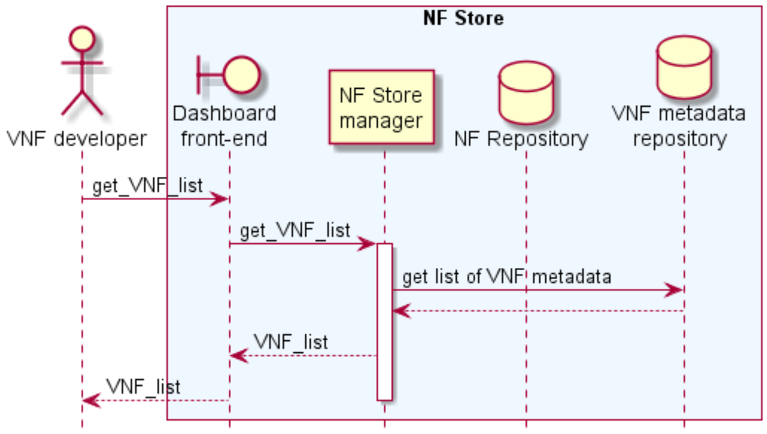

Figure 2-9. Get list of VNFs in the NF Store (Dashboard front-end)

Operation get_VNF_metadata retrieves the metadata descriptor of a given VNF (see Figure 2-10).

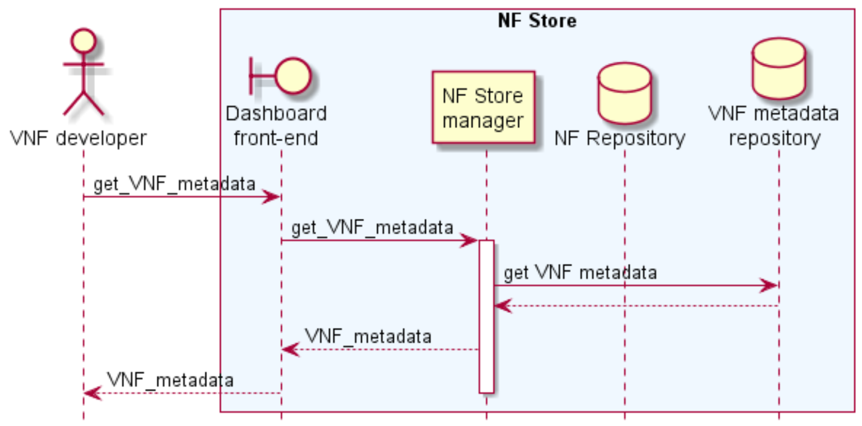

Figure 2-10. Get VNF metadata descriptor from the NF Store (Dashboard front-end)

2.5.1.2. Orchestrator front-end

The T-OR-Nfs interface supports the following operations:

- get_VNF_image
- get_VNF_list
- get_VNF_metadata

The get_VNF_image is the main operation that is expected to be invoked (see Figure 2-10). It can be executed at any time for extracting a VNF VM image from the NF Store. T-NOVA orchestrator was informed about VNF metadata when the VNF was published into the NF Store. Therefore, it is able to directly address the VM images composing the VNF.

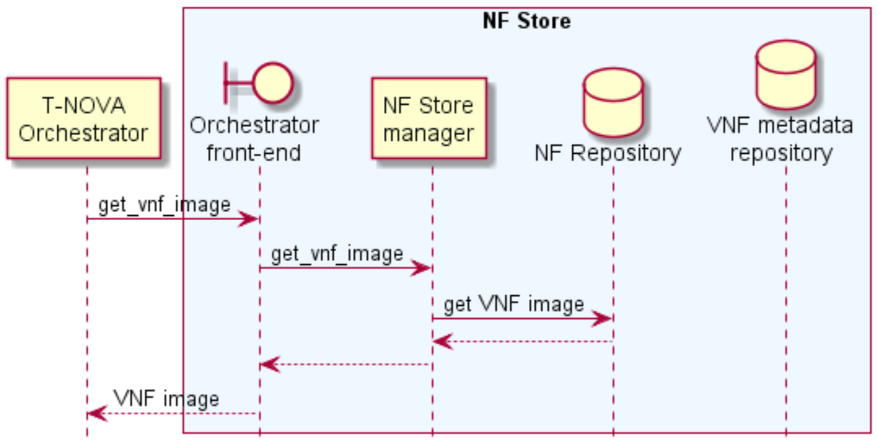

Figure 2-11. Get VNF image

The NF Store supports also scenarios where the orchestrator can ask to refresh the entire list of VNFs available in the NF Store. The get_VNF_list and get_VNF_metadata operations are used to retrieve the list of the available VNFs and then their metadata descriptors. These operations can be invoked in all the cases the orchestrator is not aligned with the SF Store, for example after a restart of the orchestrator due to software update or any other reason.

Operation get_VNF_list, represented in Figure 2-12 is logically equivalent to the same operation executed over the Dashboard front-end interface.

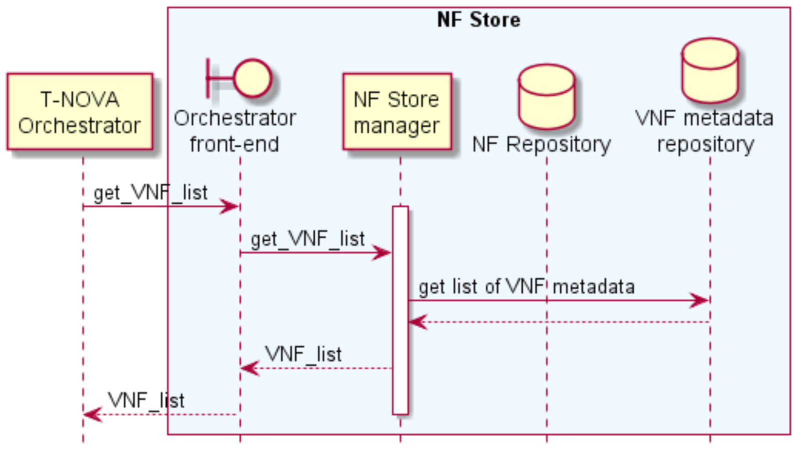

Figure 2-12. Get list of VNFs in the NF Store (orchestrator front-end)

Operation get_VNF_metadata, represented in Figure 2-13 is logically equivalent to the same operation executed over the Dashboard front-end interface.

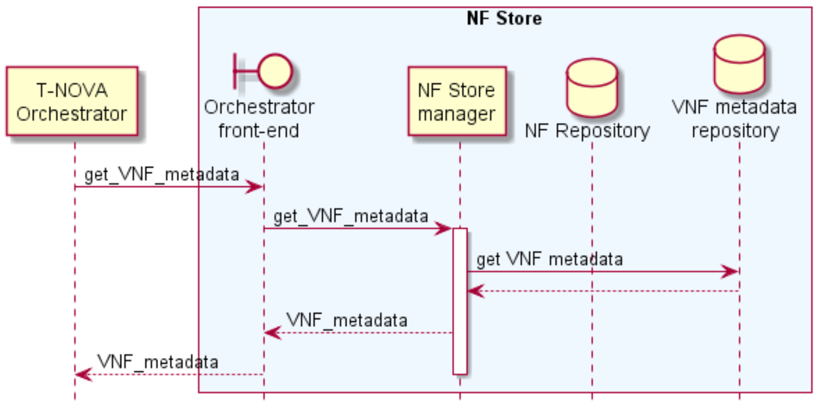

Figure 2-13. Get VNF metadata descriptor from the NF Store (orchestrator front-end)

2.5.2. Lifecycle

The NF Store implements the Publication state of the VNF lifecycle. Extensive description is available in deliverable [D2.41].

2.6. Interfaces

This section summarizes the APIs for interacting with the NF Store for uploading, downloading, and deleting the VNF VM images and the related metadata descriptor.

Interface T-Da-Nfs contains the following operations:

- publish_VNF_metadata, publish_VNF_image
    - Upload into NF Store the VNF image and the VNF metadata descriptor.
- withdraw_VNF

  - Delete from NF Store the VNF image and the VNF metadata descriptor.
- get_VNF_metadata
  - Read from NF Store the VNF metadata descriptor.
- get_VNF_list
  - Get from NF Store the list of stored VNF.

Interface T-Or-Nfs contains the following operations:

- get_VNF_metadata, get_VNF_image
  - Read from NF Store the VNF image and the VNF metadata descriptor.
- get_VNF_list
  - Get from NF Store the list of stored VNF.

The interface is implemented by REST primitives using JSON for data representation.

2.7. Technology

For the NF Store implementation we chose java web service on Tomcat application server. In a second phase we are planning to adopt a Python based web-framework such as web2py [web2py] or django [django], so that we will be able to measure the differences in development effort, performances, reliability of these similar implementations.

Apache Tomcat is an open source software implementation of the Java Servlet and Java Server Pages technologies. For our implementation it will be used version 8, the most recent major version available at the moment.

Database data is described using JPA (Java Persistence API) and the interconnection with DB using JDBC API, giving us the possibility to change it without requiring changes in source code.

The database we are going to use is H2, a pure Java SQL database with small footprint that matches very well with java applications. It can be used in embedded or server mode and give also clustering support if needed.

Data exchange on interfaces T-Da-Nfs and T-Or-Nfs are implemented by REST primitives using the standard JSON (JavaScript Object Notation) format. JSON is a lightweight data-interchange format easy for humans to read and write and also easy for machines to parse and generate a text format that is completely language independent and based on a subset of the JavaScript Programming Language.

2.8. Dimensioning and Performances

The NF Store is mainly constrained by the size of VM images. The bottleneck in term of performances is the network interface used for transmitting this huge amount of data.

In terms of dimensioning the size of VM images is again the main driver. Because each VM can be some GB large we chose to store them directly in a file system. The back-end storage facility should adopt RAID system for extending not only the available space (RAID 1) but also for increasing reliability and performances (RAID 5).

2.9. Future work

We are currently working on the implementation of a first version of NF Store using the java based technologies described in this section. When this functioning NF Store implementation will be available, we will work on an alternative version based on python technology. These two implementations will be compared in terms of performances and reliability. We will also compare the effort that was required by adopting these two different approaches. The results of this activity will be reported and hopefully published for the benefit of the research community.

At the same time we will continue to study OpenStack Murano framework for better understand how it could be enhanced to satisfy T-NOVA needs. Part of this activity will consist in exploring possible ways to collaborate with OpenStack community with the goal to contribute to Murano enhancements if we realize that it could be effectively adopted in T-NOVA.

3. Function Deployment, Configuration and Management

3.1. Introduction

The interactions of the VNFs with T-NOVA orchestrator shall support the VNF lifecycle [D2.41]. A peculiarity of T-NOVA system is the wide set of applications or VNFs developed by different and possibly independent developers. Therefore, the decision was that VNFs shall support the T-NOVA VNF lifecycle but are not imposed to implement a unique standard interface for interacting with the T-NOVA orchestrator. The chosen solution was to introduce a middleware between the T-NOVA orchestrator and the VNFs.

The purpose of the middleware framework is to implement the interface between T-NOVA orchestrator components and the VNFs, identified in Deliverable D2.41 as SWA-3 [D2.41] and represented in Figure 3-1. This framework exposes a common API to the T-NOVA orchestrator which allows triggering configuration and management actions on VNFs.

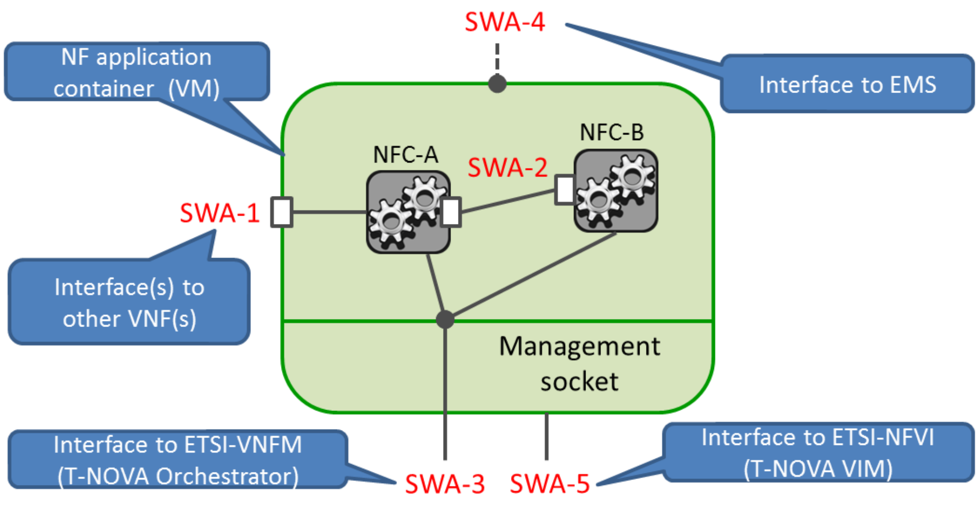
SWA-4
NF application container (VM)
Interface to EMS
NFC-A
NFC-B
SWA-2
SWA-1
Interface(s) to other VNF(s)
Management socket
Interface to ETSI-VNFM (T-NOVA Orchestrator)
SWA-3
SWA-5
Interface to ETSI-NFVI (T-NOVA VIM)

Figure 3-1. T-NOVA NFV structure mapping to ETSI framework

During the Management and Termination states of the VNF lifecycle [D2.41], the T-NOVA orchestrator interacts with VNFs by using the RESTful API in the middleware framework. The API exposes generic methods, which map in the VNF lifecycle.

To enforce the requests on VNFs, the framework uses one of the available drivers, which are mapped in a one-to-one relationship between the latter and distinct VNFs.

Figure 3-2 shows the T-NOVA Architecture with the VNF middleware API, represented by an orange box. Although the middleware API is logically located between the orchestrator and the VNFs, it makes sense to include it within the orchestrator. First, the consumers of the middleware API are the management and orchestration

(MANO) elements, which are all within the T-NOVA orchestrator. Secondly, the middleware API aggregates various mechanisms used to communicate with the VNFs, which makes the middleware API not only an interface but a framework which abstracts the real interface implementation.

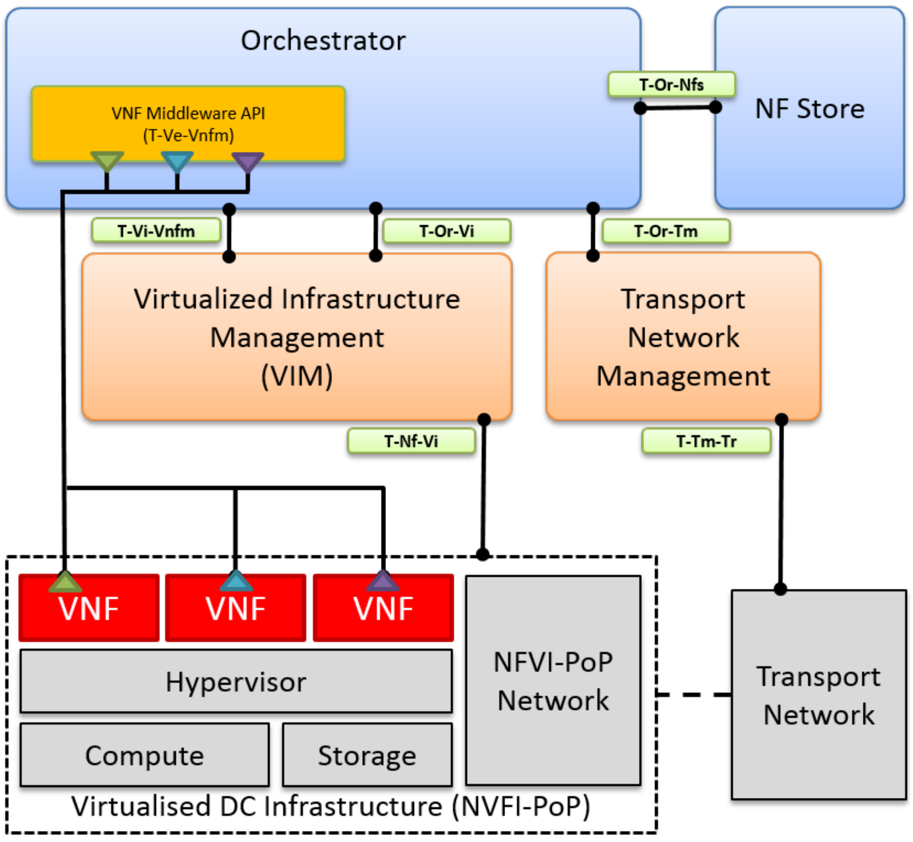

Figure 3-2. T-NOVA Architecture with VNF Middleware API

3.2. State-of-the-art

Traditional network management, in particular, configuration protocols and tools were discussed in RFC 1157. The Simple Network Management Protocol (SNMP) [RFC1157] utilizes Management Information Bases (MIBs) to implement the management functionality. It provides methods for (1) remotely reading and writing the MIBs on the managed resources, and (2) monitoring their state and control their configuration. Even if SNMP is often used for monitoring tasks (thanks to the advanced MIB state notification mechanisms it provides), it is also utilized for other Configuration Management operations. Unfortunately, SNMP heavily depends on standardized MIBs and the fact that some of the parameters that need to be configured are vendor specific and not part of any standardized MIB, leads to some limitations. In other words, these parameters cannot be managed through SNMP and their management relies on other tools, such as the Command Line Interface (CLI) of the device to be configured. The CLI defines a console based interface to the device, through which the state of the resource can be manually requested and re-configured.

A modern protocol for Network Management is reflected by the Network Configuration Protocol (NETCONF) [RFC4741] that is built based on technologies such as XML. NETCONF mainly targets the configuration related tasks in Network Management. It allows a hierarchical structuring of configuration information through XML based NETCONF datastores, and provides operations to remotely modify this configuration data. Typically, the contents of the NETCONF structure are defined using CLI, thus in this case it acts as a protocol to structure and remotely execute CLI configuration requests. Similarly to SNMP and CLI, NETCONF also relies on the involvement of the operator, as it does not provide mechanisms for the auto-discovery of incrementally connected routers.

The Simple Middlebox Configuration (SIMCO) protocol [RFC4540] enables the configuration of firewall and NAT. As known, NAT and firewalls can hinder the communication between end hosts by blocking particular ports or addresses. SIMCO allows end hosts to dynamically configure NATs and firewalls such that packets can go through. SIMCO is a point-to-point protocol with a host being one point and the middleboxes is another point, i.e. to configure all middleboxes on the path, the end host needs to execute multiple separate SIMCO sessions. SIMCO is a binary-based protocol.

Next Steps In Signaling (NSIS) [RFC4080] provides a set of signaling protocols which supports different applications. One of these applications is firewall and NAT control. As with SIMCO, NSIS offers a signaling protocol to dynamically configure NAT and firewall middleboxes by end hosts. In contrast to SIMCO, a host does not need a separate communication session to configure each middleboxes on the traffic path, i.e. NSIS signaling (configuration) messages propagates through all middleboxes on the path from one end host to the other. NSIS is also a binary-based protocol.

Similar to openFlow, OpenNF [OpenNF] offers a set of flow-based APIs to configure and control the internal state of middleboxes. The APIs allows inserting, copying and deleting of middleboxes' state. The APIs was designed by considering different types of middleboxes with different requirements. OpenNF design focuses on facilitating state migration between different instances of NFs (middleboxes). OpenNF APIs are implemented using JSON.

3.3. Requirements

The middleware API shall expose methods which guarantee all the orchestrator configuration and management necessities towards the VNFs during the lifecycle stages. The states considered in the VNF lifecycle are the management and termination. The management state can be divided in further states which comprise the extended VNF lifecycle: setup; start; scale; stop. Beside these requirements, there are others which are directly related to this component and how it can fulfil its role:

Scalable - Since this component is a generic configuration and management API for all VNFs deployed in T-NOVA, the number of VNFs may be significant (e.g. vHG). Scaling capabilities are required to keep response times low, especially during peak times.

Modular - To enforce the requests made by the orchestrator several technologies are available, e.g. SSH, SNMP, Telnet. Usually the enforcement mechanism is chosen by the VNF developer, it can support one or more technologies. Again, since this component is generic to all VNFs it needs to support all the necessary technologies and as new VNFs are added it needs to support the adoption of new ones as seamlessly as possible.

Extension Capabilities - Since the framework will have to interact with a myriad of VNFs, it needs to support not only current interfaces but future ones as well. This imposes the need to deploy mechanisms which allows extending the available interfaces.

High Availability - A failure in this component would lead to the disruption of communications between T_NOVA orchestrator components and the VNFs. Due to its crucial role in the configuration and management of VNFs, it's extremely important that the downtime is kept to the absolute minimum, which may imply some redundancy mechanisms.

RESTful API – A RESTful API is a type of interface that is useful to provide a level of abstraction between the interface and the actual implementation. Also, this style of architecture decouples the interface from the architecture which in turns enables independent development.

Driver Abstraction - From the T-NOVA orchestrator perspective there is no need to know which driver is going to be used to enforce a configuration on the VNF. The T-NOVA orchestrator only needs to focus on the requests, e.g. start VNF, stop VNF, etc. Abstracting the southbound drivers not only facilitates the co-existence of different technologies in a very heterogeneous environment but it also eases the integration of new drivers and of the VNFs with the T-NOVA orchestrator.

3.4. Architecture

Figure 3-3 shows the components of the VNF middleware API and their interaction. The Northbound API receives requests sent by the T-NOVA orchestrator, which may hold some input parameters. Afterwards, the northbound API forwards the request to the Service Logic, which will request from the VNF Registry the recipe for the specific VNF. At this moment, the Service Logic holds all the necessary information (recipe plus input parameters) to build the configuration template. After building the configuration template, the Service Logic will trigger the specific VNF driver that will enforce the desired changes on the VNF.

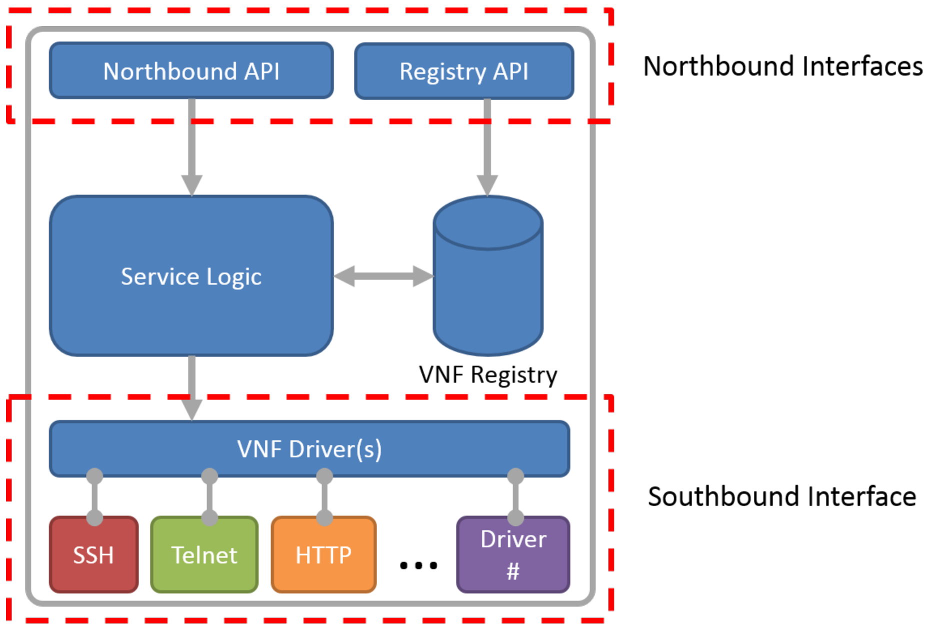

Figure 3-3. VNF Middleware API Architecture

3.4.1. VNF Middleware API Components

Northbound API – component that exposes configuration and management actions to MANO elements within the T-NOVA orchestrator.

VNF Registry – component that stores and provides the recipes that describe how each action exposed on the northbound API is performed on specific VNFs. For scalability reasons, this component may reside outside of the middleware framework to ease horizontal scaling.

Service Logic – component that parses the requests received through the northbound API and uses the information contained in the recipes to build the configuration templates that will be enforced on VNFs.

Driver(s) – enforcement mechanisms which are capable of performing actions on the VM that hold the VNF or the VNF itself. In some cases, it might be necessary to transfer configuration files or to make requests to an API within the VNF.

3.4.2. Non-Functional Elements

In this section non-functional elements within the VNF middleware API are described.

Recipe – file or group of files that contain the necessary information on how to perform specific actions on VNFs. These files are a subset of the VNF Descriptor [D2.41].

Configuration Template – the configuration template is built by the Service Logic by using the data contained in recipes and input parameters received through the northbound API. When ready, the configuration template is sent by the Service Logic to the specific driver for enforcement on VNFs.

3.5. Functional description

3.5.1. Sequence diagrams

In this section the sequence diagrams below will illustrate some of the internal workflows for different methods. Figure 3-4 shows the procedures during a request to retrieve the current configuration of a VNF.

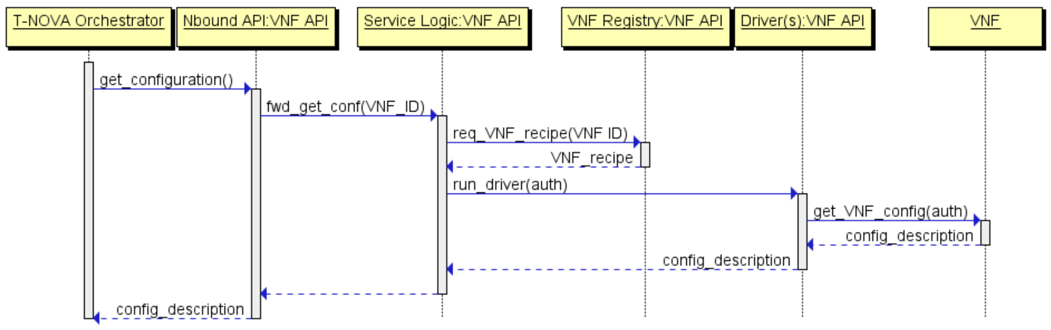

Figure 3-4. VNF Get Configuration Sequence

Setting a configuration in a VNF has an extra step in the workflow in comparison with the get configuration method, see Figure 3-5. The extra step is where the configuration template is built before activating the driver to enforce changes in the configuration.

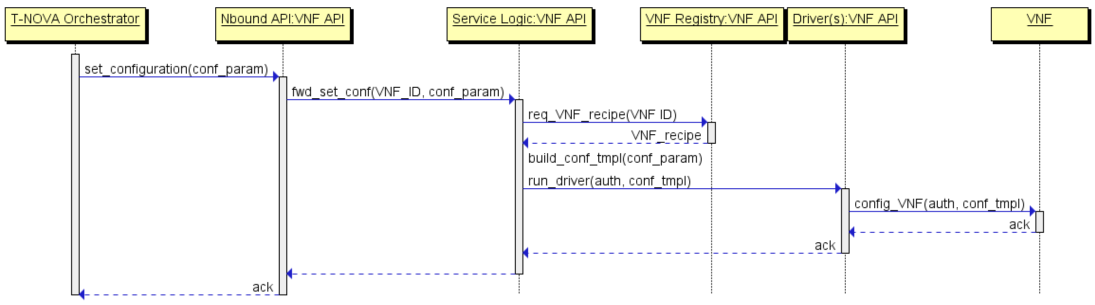
T-NOVA Orchestrator
Nbound API:VNF API
Service Logic:VNF API
VNF Registry:VNF API
Driver(s):VNF API
VNF
set_configuration(conf_param)
fwd_set_conf(VNF_ID, conf_param)
req_VNF_recipe(VNF ID)
VNF_recipe
build_conf_tmpl(conf_param)
run_driver(auth, conf_tmpl)
config_VNF(auth, conf_tmpl)
ack
ack
ack

Figure 3-5. VNF Set Configuration sequence

The start operation sequence diagram is shown in Figure 3-6. The same workflow applies to the stop and termination methods by changing the start function calls to the respective function calls.

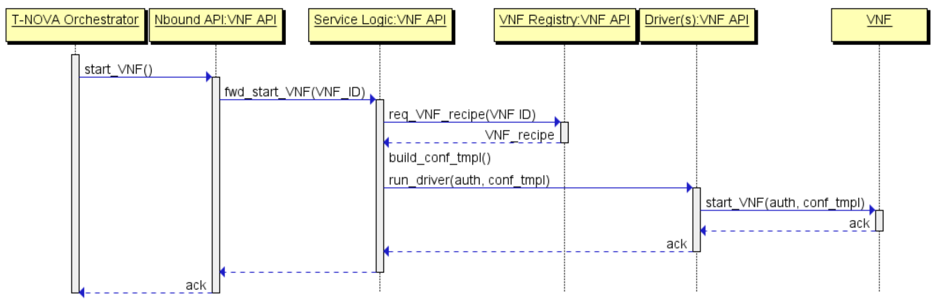

Figure 3-6. VNF Start sequence

3.6. Interfaces

This section covers the interfaces present in the VNF Middleware API. These are divided in two parts, the interfaces towards the T-NOVA orchestrator (northbound) and the interfaces towards the VNFs (southbound).

3.6.1. Northbound interfaces

The T-NOVA orchestrator uses the VNF Middleware API northbound interfaces in two distinct occasions. Firstly, to upload and manage the VNF recipes stored in the repository and secondly to manage VNFs. Although these interfaces can be realized using within a common REST API, their functionality and purpose is distinct and thus are regarded as different interfaces.

3.6.1.1. VNF API

This interface exposes methods which are mapped in the VNF extended lifecycle [D2.41]. These methods enable the T-NOVA orchestrator to control and manage VNFs during the different stages of the VNF lifecycle (see Table 1).

Method	**Description**
Get Configuration	Retrieval of the VNF current configuration
Set Configuration	Enforce specific configuration on the VNF
Start	Start the VNF service
Stop	Stop the VNF service
Terminate	Terminate the VNF service

Table 1. VNF API methods description

The Stop and Terminate methods are used in different situations, the first is used to stop the service without destroying or terminate the VNF and can be resumed afterwards. The second is used to terminate the VNF service before shutting down the resources.

An additional sixth method might be implemented depending on the VNF developer requirements, the "custom command" method. This can be used to trigger an operation on the VNF that doesn't fit into any of the previously mentioned methods, e.g. scaling.

3.6.1.2. Repository Registry

The repository contains the recipes that describe how to perform specific operations on VNFs. These recipes are part of the VNF Descriptor and need to be uploaded and managed by the T-NOVA orchestrator. Table 2 shows the available operations in Repository Registry.

Operations	**Description**
Post	Creates a new VNF API on the middleware framework by uploading a new recipe to the repository and returns the VNF API ID
Update	Updates a VNF recipe in the repository, can be used for VNF versioning
Delete	Removes the VNF recipe from the repository

Table 2. Repository Registry operations description

3.6.2. Southbound interface

The southbound interface will be implemented by one of the available drivers, these enable the enforcement of configurations on VNFs. The drivers are called by the Service Logic component, which uses a configuration template to describe the operations to be performed. These operations can be commands, upload of configurations files or any other necessary action.

The actual drivers consist of implementations of standard network protocols. Although it is intended to support the most common such as SSH or Telnet, it is impossible to support the myriad of network protocols. Therefore, there is a need for some extension mechanism that allows VNF developers to use the driver of their choice. The HTTP driver is a special driver because it allows VNF developers to extend the VNF Middleware API. By using this driver VNF developers can deploy an API within the VNF or in an external module, see Figure 3-7. The latter allows VNF developers to use proprietary or other specific drivers which are not present in the middleware framework.

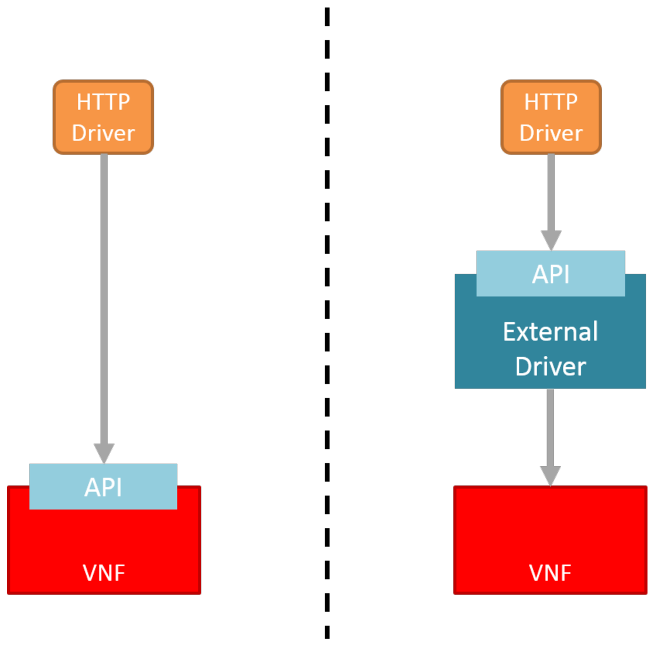

Figure 3-7. Extending available drivers

3.7. Technology

A few technologies are mature and well known in the development world to perform configuration management and remote execution application Salt [Salt], Puppet [Puppet], Chef [Chef], and Ansibl [Ansibl] are the most known tools.

They all propose an abstraction over the operating system and deployed services and applications. They all provide both CLI and Rest APIs to remotely orchestrate a park of machines. They all allow the development for Execution modules, making it possible for developers to write code to pilot the application and services.

The decision about which technology actually adopt for our implementation is still ongoing.

3.8. Future work

For the near-term, the following objectives will be investigated:

- Define the complete process for the onboarding of VNF recipes
- Retrieve requirements from VNF Developers to better align the proposed solution
- Test and evaluate the proposed technologies which can be used to implement the VNF Middleware API
- After choosing one of the technologies, we will analyze the points that need to be explored to fulfill the T-NOVA requirements

4. Development of Network Functions

4.1. Introduction

This section provides the description of the VNFs developed in WP5. They are:

- Security Appliance (SA)
- Session Border Controller (SBC)
- Traffic Classification (DPI)
- Home Gateway (HG)

These virtual network functions will be used for proving the effectiveness of T-NOVA concept. Some of them represent real implementations of first stage prototypes interesting for T-NOVA industrial partners willing to address the NFV market.

Each VNFs is described in self-consistent chapters, while this introduction provides a general description and design choices applicable to any VNF that is part of T-NOVA ecosystem.

4.1.1. Packet handling acceleration frameworks

The ordinary environment where the VNFs will be executed is within virtual machines or containers operating over a host hypervisor that provides the appropriate abstraction mechanisms allowing virtualization of the available hardware resources. The challenge that VNF performance faces is that of the method of communication between the actual network interface card (NIC) hardware and the VM.

Depending on the architecture of the system, resource access from the guest VM can be mediated by the hypervisor, or physical resources may be partially or completely allocated to the guest, which then uses them with no interference from the hypervisor (except when triggering protection mechanisms).

In this context Figure 4-1 illustrates the three cases that can be envisaged. (i) Case A, the hypervisor does a full emulation of the network interfaces (NICs), and intercepts outgoing traffic so that any communication between the virtual machine instances goes through it (-net user mode in QEMU); (ii) Case B the hypervisor still provides NIC emulation, but traffic forwarding is implemented by the host, e.g. through an in-kernel bridge (-net-tap) or another proprietary module; (iii) Case C, the virtual machine has direct access to the NIC (or some of its queues). In this case, the NIC itself can implement packet forwarding between different guest (see C1 line), or traffic is forwarded to an external switch which in turn can bounce it back to the appropriate destination (see C2 line).

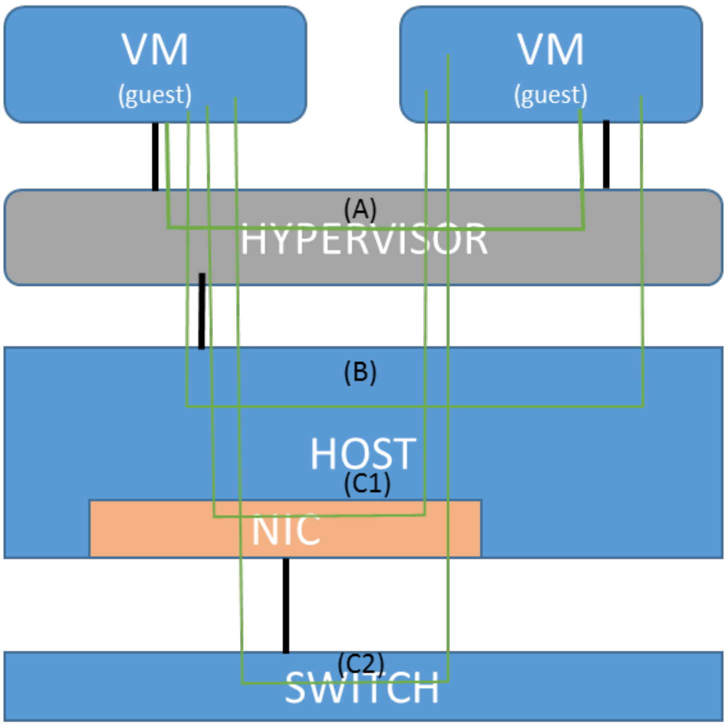

Figure 4-1 NIC access I/O schemes for VM-to-VM communication

It is generally largely understood that NIC emulation is slow, and direct NIC access (as in C1 and C2) is generally necessary for efficient virtual network performance. However, it is most times required some sort of hardware support to make sure that the guest does not interfere with other critical system resources. In this context packet handling acceleration frameworks have been proposed namely:

- DPDK Framework
  Data Plane Development Kit (DPDK) is an Open Source BSD licensed project [DPDK]. DPDK provides a set of libraries and drivers for fast packet processing. These libraries can be used to: i) receive and send packets within the minimum number of CPU cycles (usually less than 80 cycles); ii) develop fast packet capture algorithms (tcpdump-like); iii) run third-party fast path stacks. For example, some packet processing functions have been benchmarked up to 160 Mfps (million frames per second, using 64-byte packets) with a PCIe Gen-2 NIC.
  It was designed to run on any processors knowing Intel x86 has been the first CPU to be supported. It runs mostly in Linux userland. The DPDK is device specific since it required specialized chipset for the support of its enhanced capabilities. A list of supported hardware is provided in [DPDK_NIC].
  Currently a lot of NFV related implementations exploit DPDK for optimized and enhanced performance under utmost traffic conditions. The DPDK development community has provided extensions of OpenVirtualSwitch (OVS) supporting DPDK [vSwitch] and in addition to this specific memory/kernel configurations have been proposed enhancing the VM communication with the hardware NIC [DPDK_RN170].

- PFRing

PFRing is a new type of network socket that dramatically improves the packet capture speed [PFRing]. PFRing is available for Linux Kernels (>2.6.32) and is provided as a loadable module (no need for patching). The capabilities of PFRing support 10Gbit hardware packet filtering using commodity network adapters. Current implementation is device driver independent and also supports for libcap implementations in an effort for backwards compatibility. PFRing supports inherently content inspections and Berkley Packet Filters (BPF) type of rules for creation of matching traffic filters from L2 and above. Additionally it supports plugins for on top functionalities. Latest evolutions are the support of zero copy at the user space level (using Direct NIC access – DNA drivers), thus supporting extreme packet capture/transmission speed as the NIC NPU (Network Process Unit) is pushing/getting packets to/from userland without any kernel intervention. Extensions for supporting VM shared threads is provided by PRRing ZC library.

- NetMap
  netmap / VALE is a framework for high speed packet I/O. Implemented as a kernel module for FreeBSD and Linux, it supports access to network cards (NICs), host stack, virtual ports (the "VALE" switch), and "netmap pipes". netmap can easily do line rate on 10G NICs (14.88 Mpps), moves over 20 Mpps on VALE ports, and over 100 Mpps on netmap pipes [Netmap].

4.1.2. GPU-based acceleration

NFV allows to consolidate a great variety of network functions, which could previously run only on bespoke hardware by specific vendors, as software appliances executed on top of standard commodity servers by any provider. This way, network operators can significantly reduce OPEX and CAPEX, accelerate time-to-market and achieve the high level of automation and flexibility in service provisioning that the cloud computing paradigm can offer.

On the other hand, however, the outburst of innovative services and applications coming from the internet world is generating an ever-increasing demand for low cost and energy-efficient processing power, which can be hardly satisfied only by the technological evolution of general purpose CPU’s.

For this reason, a great interest is now on the use of hardware accelerators, to offload the commodity server general purpose CPU’s of the most intensive workloads. Many different types of specialized devices are available, which can significantly accelerate networking functions, video applications, cryptographic algorithms, audio coding, etc. Among these, general purpose 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, guaranteed by the huge demand of such devices originated by the gaming market [Kirk].

With respect to the extremely complex and closed-source graphic processors available a decade ago, current GPU’s can be programmed with general purpose, high level languages, such as CUDA (by NVIDIA) or OpenCL, coming from the open source community. Moreover, effective development and debugging/profiling tools are also at hand, and are continuously enhanced by device manufacturers, to facilitate the rapidly increasing community of GPU developers [CUDA], [Karimi].

However, to fully exploit the huge potentialities offered by GPU's to the cloud computing context, it is necessary to extend virtualization also to this type of processors. Fortunately, many activities have been ongoing in this field in the last years [Duato], [Walters].

The first attempts to virtualize GPU's have been made in the areas of scientific high performance computing and education, in order to share highly expensive GPU clusters, often under-utilized, among a larger number of users. The next step was moving GPU's to the cloud within the "Infrastructure as a Service" framework. However, even if many research efforts have been spent so far, Amazon is probably the only known example of a major OTT provider offering access to GPU-enabled services to its customers [Walters].

A relevant objective of T-NOVA is to deeply change this scenario, by providing an overall framework which will allow to easily developing and straightforwardly bringing to the market high performance NFV's. In the high performance computing domain, this will mean combining the disruptive performances of GPU-based accelerators with the benefits brought about by the NFaaS (Network Function as a Service) concept. To this end, T-NOVA will contribute with innovative results at all levels of its NFV framework, i.e. in the Marketplace (WP6), in Orchestration (WP3) and at the virtualized infrastructure level (WP4).

The objective of WP5 is to provide the project with a set of innovative VNF's, to demonstrate the results achievable by the T-NOVA framework, using accelerators both in the networking (DPDK) and in the computing domain (GPU). In this section we will briefly summarize the most significant results so far achieved in the field of GPU virtualization, and will describe and motivate the (initial) choices made within the T-NOVA project to approach the use of this type of accelerators.

4.1.2.1. Current GPU market scenario

In order to be able to run GPU-accelerated VNFs in T-NOVA, high-end GPU devices are necessary. High-performance GPUs are nowadays produced by NVIDIA, AMD, and Intel.

NVIDIA was the first to propose, with G80 and CUDA, a family of architectures and a paradigm for exploiting GPUs for general-purpose parallel computing. For this reason NVIDIA's are the most diffused GPUs used in High Performance Computing (HPC) applications. To this day, NVIDIA produces two kinds of GPU boards suitable for datacenter solutions:

– the NVIDIA Tesla family, optimized for maximum computing performance, and

– the NVIDIA GRID platform, optimized for use in virtualized environments.

The top model features 4992 cores and is capable of a peak performance of 8.74 TFLOPS (single precision) and 480 GB/s memory bandwidth.

AMD (formerly ATI) has been, together with NVIDIA, producer of high-end GPUs. AMD's recent GPUs are capable of general-purpose computing, as they can be programmed using OpenCL. AMD also produces devices suitable for datacenters (AMD FirePro S series). Top models are capable of a peak performance of 5.91 TFLOPS (single precision) and 480 GB/s memory bandwidth.

Intel has also produced GPU for long, although not in form of stand-alone chips or boards, but integrated in the Intel CPUs. The recent models (starting from Intel HD 4000, embedded in third-generation i3/i5/i7 CPUs) are capable of running OpenCL code, and are therefore considered GP-GPUs. However, peak performances are currently around 800 GFLOPS, by approximatively 25 GB/s memory bandwidth, still lower than NVIDIA's and AMD's counterparts.

4.1.2.2. GPU Virtualization

In the literature, GPU virtualization is usually addressed in its most simplified form, which can be stated as follows: on a hardware platform, consisting of both general purpose CPU's and specialized GPU-based accelerators, the guest VM's running on the CPU's must be able to concurrently and independently access the GPU computing resources without security issues [Duato],[Walters],[Maurice].

As already discussed in the introduction, in T-NOVA and in general in the NFaaS context, the problem of virtualizing specialized accelerators has a much wider scope: virtualized acceleration resources must be managed at all levels of the T-NOVA stack, that is at the marketplace, at the orchestrator and at the virtualized infrastructure level.

In this document, which is focused on VNF's, for the sake of simplicity we will consider GPU virtualization adopting the narrowest definition. In fact, considerations about the wider NFaaS context are not in the scope of this deliverable.

In the literature, many techniques to achieve GPU virtualization have been proposed. However, all the proposed methods can be divided in two main categories, which are usually referred to as API remoting [Maurice] (also known as split driver model or driver paravirtualization [Maurice]) and PCI pass-through [Walters] (also known as direct device assignment [Maurice]), respectively.

API remoting – Various approaches based on API remoting have been proposed in the literature. The most well-known solutions are rCUDA [Duato], gVirtuS [Lauro], vCUDA [Shi], and GVim [Gupta]. All such approaches adopt the same technique, which consists in splitting the GPU driver into a front-end and a back-end driver. The front-end driver is included in each VM. The back-end driver runs in a privileged domain (like the Dom0 in Xen, [Walters]) and includes the GPU drivers and (in the case of NVIDIA devices) the CUDA library. Calls to the GPU library made within a VM are sent from the front-end to the back-end driver, which executes the CUDA API on the physical GPU on behalf of the VM. This way, many VM's can concurrently interact with one physical GPU; in particular, a software emulation of the GPU can be assigned to each VM. However, the performance of such techniques can be adversely affected by the performance of the transport channel between the front-end and the back-end drivers, as well as by the amount of data (which is application-dependent) that must be moved [Walters].

The open source rCUDA framework seems to be the most popular API-remoting technique to virtualize GPU's. Its main advantages with respect to the other techniques quoted above are its hypervisor-independence, as well as its maintained alignment with the CUDA API's [Duato].

Pass-through - 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 VM's accessing external devices such as accelerators in a secure way [Walters].

In general, the performance that can be achieved by the pass-through approach is higher than the one offered by API-remoting [Walters],[Maurice]. Also, the pass-through method gives immediate access to the latest GPU drivers and development tools [Walters]. A comparison between the performance achievable using different hypervisors (including also Linux Containers) is given in [Walters], 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 container 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 VM's one after the other, in a sort of "time sharing" approach [Maurice]. This limitation can be overcome by a technique also known as Direct Device Assignment with SR-IOV (Single Root I/O Virtualization) [Maurice]. 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, the only now available GPU enabled to this functionality belongs to the recently launched NVIDIA Grid family [Maurice]. Also, the only hypervisors which can currently support this type of hardware virtualization are VMWare Sphere and Citrix XenServer 6.2. However, since now also KVM can support SR-IOV, there is a path towards the use of GPU hardware virtualization also with this hypervisor, though this possibility has not been mentioned so far in NVIDIA documentation.

4.2. Virtual Security Appliance

4.2.1. Introduction

A security Appliance (SA) is simply 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 just detection and reporting. Intrusion Detection Systems are a good example. If the SA is in charge of scanning the network and identifying potential breaches (e.g. penetration testing), the SA can be qualified as preventive. Nowadays, Security Appliances combine various security features including firewalling, content filtering, and intrusion detection. For this reason, they are more known as Unified Threat Management (UTM) systems.

In the context of T-NOVA, we will focus more on the use of firewalls to show how a secure service that the T-NOVA system offers can be deployed. The firewall service will be endorsed by an intrusion detection system that will investigate deeper the suspicious traffic.

Conventional firewalls were deployed at the network border in order to examine the traffic destined to this network. As networks become more complex, it is often necessary to place firewalls between multiple network segments. With virtualization and cloud computing, the situation becomes more challenging as entire networks or network segments could be hosted completely within a virtual environment. As a result, firewalls need also to protect virtual environment in addition to physical networks. The burden of this task can be carried out by firewalls running on virtual machines. A virtual firewall (VF) is a network firewall service running entirely within a virtualized environment. It provides the usual security functionalities offered by a physical network firewall.

To be more concrete, a Firewall (FW) is a program/device that simply filters the network traffic. It controls the traffic (in and out) using one of the following methods,

- Packet filtering
- Proxy service
- Stateful inspection

The vSA can be deployed at the edges of the network close to the customer premises or at other convenient locations within the virtual network slice that has been provisioned for this customer (see Figure 4-2).

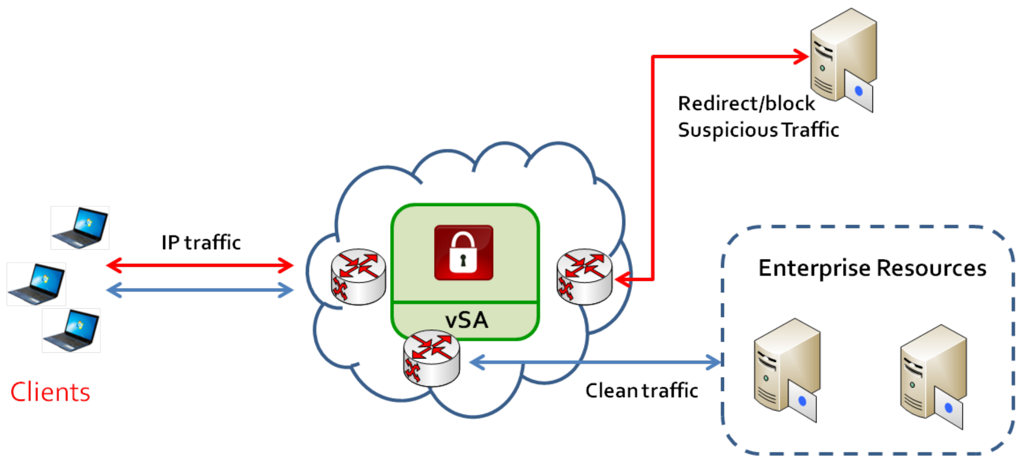

Figure 4-2 Security appliance

The vSA is able to sense potential dangerous or suspicious traffic and to respond appropriately to either block it (for example, in case of DDoS attacks) or redirect it to a traffic analysis/forensics virtual device for deeper attack pattern analysis and recognition.

4.2.2. State-of-the-art

The available technologies considered for the implementation of the vSA are briefly referenced here. The main groups are firewalls, neural networks, virtual switches.

4.2.2.1. Firewalls

In this section, we provide a short evaluation [reddit], [Wfirewalls] of firewalls software that can run in virtual environments. The idea is not to go through all the relevant existing software but just through the most popular ones that could be extended to fulfill the project requirements.

Firewall	Evaluation
Vyatta VyOS	VyOS is a community fork of Vyatta, a Linux based network operating system that provides software-based network routing, firewall, and VPN functionality. [Vyatta] • Runs on both physical and virtual platforms. • Supports paravirtual drivers and integration packages for virtual platforms. • Completely free and open source. pros: open source, large user base, REST APIs, high performance, root shell, support for IDS
pfSense	The pfSense project is a free network firewall distribution, based on the FreeBSD operating system with a custom kernel and including third party free software packages for additional

	functionality. [pfSense] Runs on both physical and virtual platforms pros: Open source, Web User Interface, very easy to use, large community, root shell, integration of external packages cons: incomplete bgp/ospf, xml config, no config cli, no REST APIs
Halon	Halon Virtual Security Router (VSR) is an OpenBSD-based firewall, router, VPN and load balancing appliance focusing on security, flexibility and manageability. [Halon] Runs on both physical and virtual platforms pros: Open source, Web User Interface, SOAP APIs, juniper-style config/rollback/commit, inexpensive, root shell, pkg_add cons: small community, unknown vendor, no IPS functionalities
m0n0wall	m0n0wall is a project aimed at creating a complete, embedded firewall software package that, when used together with an embedded PC, provides all the important features of commercial firewalls. m0n0wall is based on FreeBSD, along with a web server, PHP and a few other utilities. The entire system configuration is stored in one single XML text file to keep things transparent. [m0n0wall] Runs on both physical and virtual platforms Provides packet filtering, VPN, NAT, IPS pros: open source, very small image, root shell, pkg_add cons: less feature complete than pfSense
Vuurmuur	Vuurmuur is a powerful firewall manager built on top of iptables that works with Linux kernels 2.4 and 2.6. It has a simple and easy to learn configuration that allows both simple and complex configurations. [Vuurmuur] Runs on both physical and virtual platforms pros: open source, no iptables knowledge required, human readable rules syntax, Ncurses GUI, no X required, potential integration with IDS/IPS, traffic volume accounting cons: no REST APIs for configuration, less feature complete than pfSense

4.2.2.2. Neural Networks

Data mining techniques will be introduced into the IDS engine in order to support the misuse detection achieved at the protocol analysis. This combination will ensure a better detection. Neural Networks (NNs) are popular methods that have been applied in research and real life circumstances. They have been often proposed for use in the data security area due to their flexibility. To be more precise, a NN is able to analyze the network data even this data is incomplete or distorted. For this reason, it is foreseen in the T-NOVA project to use this technique as well.

Some NNs can exploit known cases of attacks and are trained so as to be capable of identifying analogous attack cases occurring in the future. Such NNs are usually Feed Forward NNs and belong to the general class of supervised NNs. Although useful, such approaches fail to identify malicious activities that are unknown or not identified in the past. Another interesting approach is to focus on the clustering capabilities of a special type of unsupervised NN called Self Organized Map (SOM). SOMs are simple networks that unlike other types of NNs do not use activation functions nor have hidden layers.

4.2.2.3. Open vSwitch

Open vSwitch is a virtual switch that is used to bridge traffic between virtual machines and with the outside word. It is integrated into the latest Linux kernel and comprises in particular the following components,

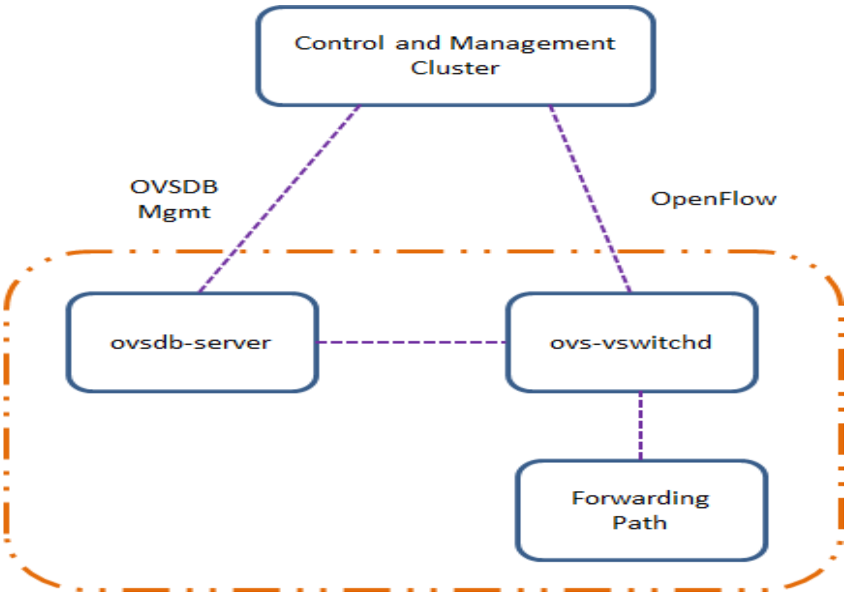

Figure 4-3. Open vSwitch architecture

- *ovs-vswitchd*, A daemon that manages and controls Open vSwitch switches on the local machine. The Openflow protocol is being used to talk to ovs-vswitchd
- *ovsdb-server*, a lightweight database that holds switch-level configuration. External clients can talk to ovsdb-server using ovsdb management protocol. The Open vSwitch Database Management Protocol (OVSDB) is an OpenFlow configuration protocol designed to manage Open vSwitch implementations
- *control and management cluster* contains client tools to talk to ovsdb-server and ovs-vswitchd

4.2.3. Requirements

The following requirements are considered.

1. The vSA shall protect the service from malicious traffic
2. The vSA shall provide simple traffic filtering as well as deep inspection
3. The vSA shall run on virtual machines
4. The vSA shall provide appropriate APIs for configuration
5. The vSA shall provide an acceptable level of performance
6. The vSA shall be flexible enough to enable detection rules revision

4.2.4. Architecture

The architecture of the virtual Security Appliance (vSA) is depicted in Figure 4-4. It is composed from three main components,

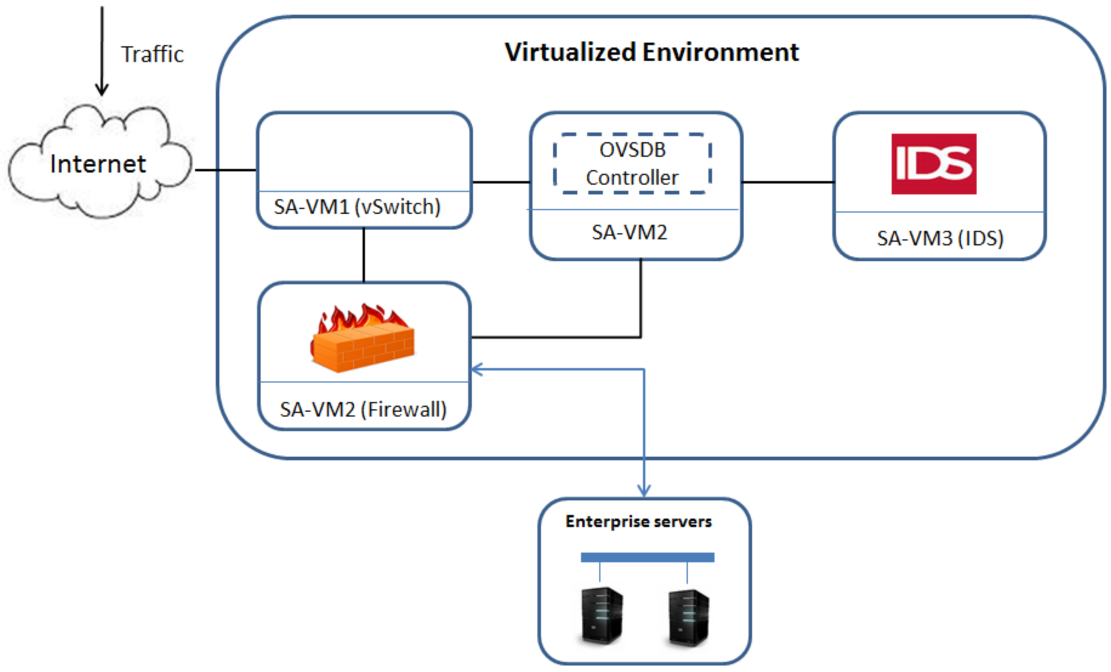

Figure 4-4. vSA high-level architecture

The firewall: this component will be in charge of filtering the traffic towards the service. As mentioned earlier, this component would run pfsense or Vyatta core extended to fulfill the project requirements. It is worth to mention that packet filtering firewalls are often unable to discover packets with malicious payload as they just look at the source address, destination address, protocol, and port number.

The Intrusion Detection System (IDS): In order to improve detecting attacks, 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 have the advantage over signature based IDS in detecting novel attacks for which signatures do not exist. Unfortunately, anomaly detection IDS suffer from high false detection rate. So it is expected that combining both arts of detection will improve the detection and reduce the number of false alarms. In T-NOVA, an appropriate existing signature based IDS (e.g. snort, Bro, Suricata) will be extended to support anomaly detection as well. The mode of operation of the IDS component is depicted in Figure 4-5.

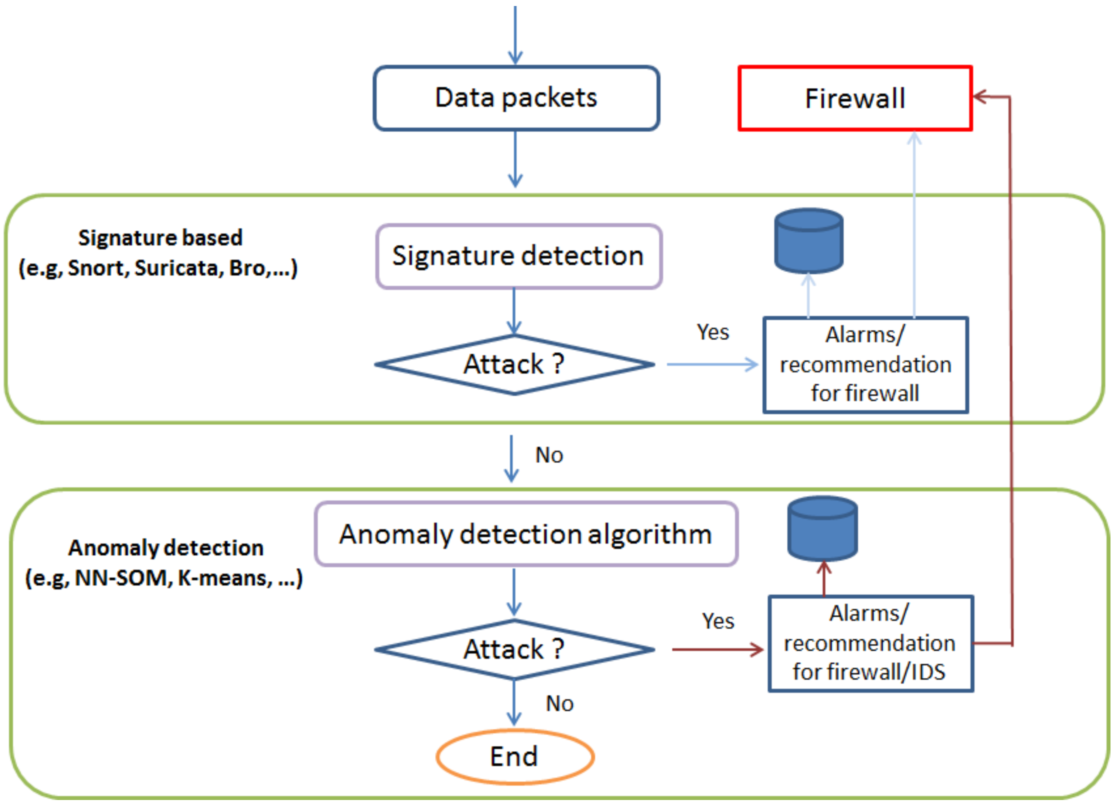
Data packets
Firewall
Signature based
(e.g, Snort, Suricata, Bro,...)
Signature detection
Attack ?
Yes
Alarms/ recommendation for firewall
No
Anomaly detection
(e.g, NN-SOM, K-means, ...)
Anomaly detection algorithm
Attack ?
Yes
Alarms/ recommendation for firewall/IDS
No
End

Figure 4-5. IDS process flow diagram

- The data packets are first of all filtered by the firewall before the analysis by the IDS is conducted
- The IDS will monitor and analyze all the services passing through the network
- As a first step, the 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
- If an attack is detected at this stage, an alarm is generated or the firewall is contacted to revise its rules
- If no attack is detected, the data packets will be passed to an anomaly detection algorithm. In our context, it might be NN-SOM, K-means, or another one
- In the same way, if an attack is detected, an alarm is generated or the firewall and the signature based IDs will be contacted to revise their rules
- If no attack is detected, no further action is required

The OVSDB controller: As the firewall and the IDS run on different virtual machines and need to interact with each other, a third component is needed to facilitate this interaction and forward the traffic between the firewall and the IDS virtual machines. For this purpose, Open vSwitch is going to be used. It is an open source software (client and server) designed to be used as a virtual switch. It can also be extended and controlled using OpenFlow and the OVSDB (Open vSwitch Database) management protocol [RFC7047]. Alternatively OpenFlow could be used in place, along with a simple controller instance. However in order not to complicate our implementation OVSDB solution is selected.

4.2.5. Lifecycle

With respect to the general VNF lifecycle [D2.41], the lifecycle of the vSA includes: Start, Stop, Monitor, and Configure.

4.2.6. Technology

In the context of T-NOVA, open source firewalls will be extended to fulfill the related requirements and used. So far, two candidates seem to be more appropriate,

- Vyatta Core [Vyatta], which is the open source version of Vyatta. Its main offerings are IPv4 and IPv6 routing, stateful firewall, IPSec and SSL VPN, and intrusion prevention. It seems it also support REST APIs for configuration
- pfsense. The pfSense project [pfSense] is also an open source network firewall distribution. It also inludes third party free software packages for additional functionality. Through this package system, pfSense is able to provide most of the functionality of common commercial firewalls. pfSense software includes a web interface for the configuration of all included components. Unfortunately, no REST APIs for configuration exist so far

4.2.7. Dimensioning and Performances

Firewalls are often implemented in routers to control packet flows. If the packet filtering process generates an extra overhead, this will, certainly, affect the performance of the system and lead to degradation in its time response.

To study the performance of firewalls, benchmarking techniques are needed. Unfortunately, activities in this area are very scarce. As an example, the IETF Benchmarking Methodology Working Group produced several Request for Comments (RFCs) describing benchmarking terminology and methodology for a wide range of networking devices. Performance benchmarks related to firewalls are discussed in [RFC2647] and [RFC3511]. The suggested methodologies are intended to be standard benchmarking for all classes of firewalls. Unfortunately, this makes them too general to be applied to a particular class of firewall. So far, it seems to us that the methodology suggested by Kean and Mohd [KeanMohd] for evaluating firewalls performance. This methodology suggests the following metrics:

- <u>Throughput:</u> The maximum rate at network layer which none of the received packet is dropped by the firewall without activating filtering rules. In [RFC2647], the throughput is defined as the actual payload that is received per unit of time
- <u>Latency:</u> The time interval starting when the last bit of input frame reaches at the input interface of the firewall, and ending when the first bit of the output frame is observed at the output interface of the firewall
- <u>Jitter:</u> measures the variation in delay of the received packet

- Goodput: The rate at which packets are forwarded to the correct destination interfaces of the firewall, excluding any packets dropped due to the rule set definition. The goodput could be seen as the opposite of the Packet Loss Rate (PLR) which is the ratio of the lost packets to the total transmitted packets

When testing the firewall components, different related specifications (that will also be used when defining SLAs) will be used. These specifications include,

- Firewall model
- Operating System
- Memory
- Processing Speed
- Interfaces
- Connections/second
- Throughput
- Concurrent sessions

The testing activities will be using IP traffic generators such as D-ITG [ITG], [ostinato] and [IPTraf]. The latters will mainly generate TCP and UDP traffic at different rates. Diverse loads (light, medium, heavy) and different packet sizes will also be considered.

4.2.8. Future work

At present the general architecture of the vSA is clear and the main components are identified. To implement the vSA, different tasks are planned:

- Setup the selected firewalls on virtual machines and investigate their functionalities (configuration, Web interface, APIs if any, etc). Different virtualization technologies might be used here.
- Check their performance based on the metrics discussed earlier.
- Identify the extensions needed to fulfill the project requirements
  - interfaces to the other components within the vSA
  - metadata for the NF store
  - configuration format and packaging
  - APIs to the orchestrator and potentially to other component in the T-NOVA system
  - vSA manager implementation

4.3. Session Border Controller

4.3.1. Introduction

A Session Border Controller (SBC) is a device used in multi-media telecommunication for providing network interconnection and security services between two IP networks whenever multi-media sessions involve two different IP network domains. Main logical components of an SBC are the Interconnection Border Control Function (IBCF) for the signalling procedures and the Border Gateway Function (BGF) for user data plane processing. Signalling procedures are usually implemented by the Session Initiation Protocol (SIP), while the data or use plane usually adopts Real-time Transport Protocol (RTP) for multimedia content delivery.

As already introduced in T-NOVA "System Use Cases and Requirements" deliverable [D2.1], we will focus on the virtualized implementation of such device.

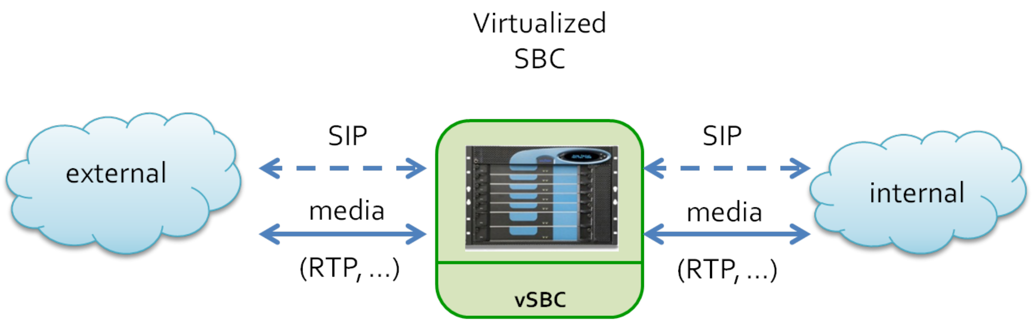

Figure 4-6 Virtualized SBC high level model

The virtualized SBC (vSBC) is the VNF implementing the SBC service in T-NOVA virtualized environment. As analyzed in [D2.41] the same functionality is compatible with ETSI NFV apart from the details of the actual implementation of the interfaces to the virtualized environment. For instance the ETSI Ve-Vnfm interface corresponds to T-NOVA T-Ve-Vnfm. On the other hand, however, applying ETSI NFV paradigm to the existing Telco infrastructures can require challenging architectural upgrades, as well as radical changes in service models and operating procedures. Moreover, the services provided by telecommunication networks greatly differ from the standard IT applications running in the cloud, in terms of "carrier grade availability" and "high processing throughputs", achieved with specialized devices such as Network Processor Units (NPU) and/or Digital Signal Processors (DSP). For this reasons the "pure" software implementation shall be complemented by acceleration technologies such as DPDK that can be available also in cloud environments [DPDK].

The vSBC implementation in T-NOVA project is a feature-limited or prototyped version of the commercial SBC Italtel is developing for NFV market.

4.3.2. State-of-the-art

Session Border Controller is well known component in IP multi-media telecommunication networks. Its logical architecture, functional and non-functional requirements are described in different standard specifications by ETSI, 3GPP and others. Some non-exhaustive references are: [ES282.001], [TS23.228], [TS29.238].

Many vendors are active in Session Border Controllers market. Among them ACME Packet (now Oracle) [SBC_ACME], Sonus [SBC_Sonus], Alcatel-Lucent [SBC_ALU], Audiocodes [SBC_Audiocodes], Metaswitch [SBC_Metaswitch], and finally Italtel [SBC_Italtel] are the most representative. For the objective of this document we want just to outline current commercial offers in SBC market and not providing detailed company-sensitive information.

Typically an SBC is a stand-alone device integrating signaling and media capabilities. Distributed deployment is also possible but not considered here. Any SBC basically supports NAT (Network Address Translation) service. Some vendor implements also media processing service also known as transcoding for adapting media streams in interconnections between two different non-compatible codecs. Finally, some vendor offers a virtualized version of this device.

Research and development departments of SBC manufacturers are now spending efforts in the design and implementation of ETSI NFV version of such devices. This is actually in line with the objective of T-NOVA project.

4.3.3. Requirements

General requirements for SBCs are summarized here.

1. IP to IP network interconnection
2. SIP signalling proxy
3. SIP signalling manipulation and interworking
4. Media flow NAT
5. RTP media support
6. Real-time audio and/or video transcoding
7. Topology hiding
8. Security gateway
9. IPv4-IPv6 gateway
10. High-availability (99,999%)

This list catches not only essential but also some advanced requirements expected by an SBC. For example media manipulation and advanced security are distinctive features supported by advanced version of commercial products.

For the objectives of T-NOVA project we will focus on a subset of these requirements with the objective to provide a virtualized implementation of an SBC according to the VNF paradigm and compliant with T-NOVA framework.

The requirements addressed in T-NOVA are then:

1. IP to IP network interconnection
2. SIP signalling proxy
3. Basic SIP signalling manipulation and interworking
4. Media flow NAT
5. RTP media support
6. Real-time audio and/or video transcoding (optional)

4.3.4. Architecture

The high level architecture of the virtualized SBC consists of specialized components represented in Figure 4-7. Each component could be deployed as a dedicated VM.

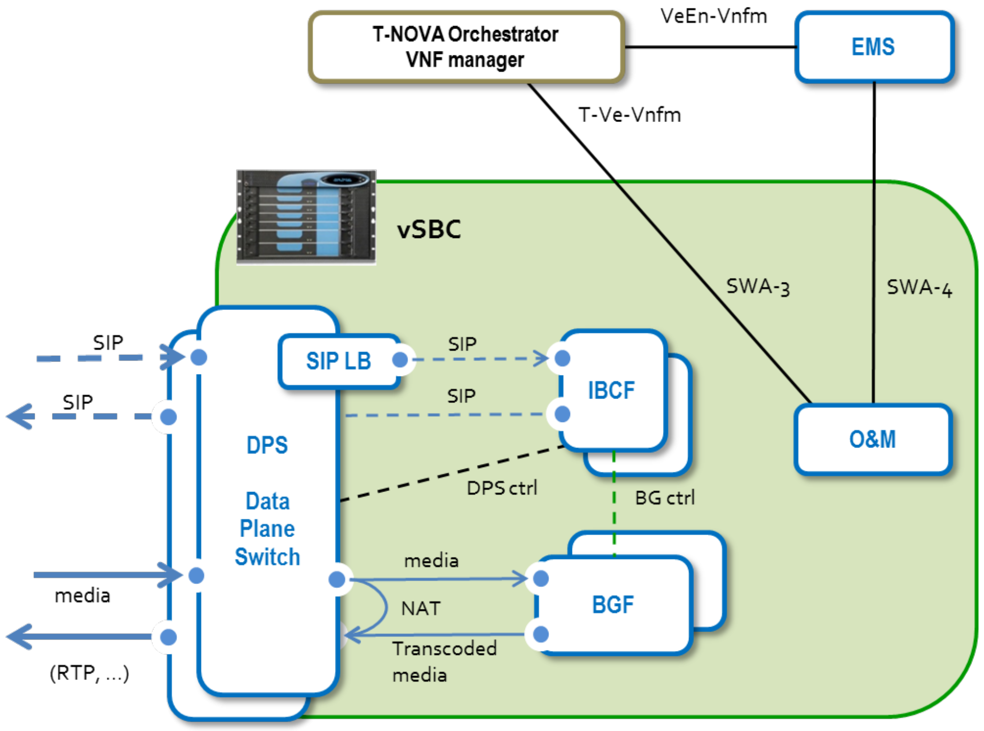

Figure 4-7 vSBC internal architecture

DPS (Data Plane Switch) manages the IP network interface using DPDK technology [DPDK]. This component implements the single ingress or egress point of both the signaling and media flows. It is controlled by a dedicated interface (DPS ctrl) to provide packet forward or NAT function.

IBCF (Interconnection Border Control Function) implements the control function of the SBC. It analyzes the SIP messages supporting the establishment, modification and termination of the communication between disparate SIP end-point applications. The IBCF can provide SIP message adaptation or modification enabling interoperability and also enforcing topology hiding and other security features. The IBCF extracts from SIP messages the information about the media streams associated to the SIP dialog and instructs media plane components to process them.

SIP LB (Load Balancer) balances incoming SIP messages traffic, forwarding them to the appropriate IBCF instance.

BGF (Border Gateway Function) processes media streams applying transcoding algorithms. This is used whenever the endpoints of the media connection support different codecs. Transcoding function is implemented by a pure software transcoder, that can dramatically increase its performance whenever GPU acceleration hardware is available in the virtual execution environment. The BGF is controlled by the BGF ctrl interface.

O&M (Operating and Maintenance) module supervises the operating and maintenance functions of the network function. In the cloud environment the O&M module extends the traditional management operations with the internal orchestration function. This orchestrator knows how to instruct the internal component of the VNF to accomplish scaling procedures and to support the pay-as-you-go paradigm. The O&M orchestrator interfaces the VNF manager that instruct it for applying the VNF lifecycle.

4.3.5. Functional description

The most innovative components of the vSBC are the Data Plane Switch and the Border Gateway Function implemented in a virtualized environment. The following chapters provide some insight about them and also outlines the research activities we are doing.

4.3.5.1. Data Plane Switch

The Data Plane Switch (DPS) is the front-end component dedicated to IP network interface management. Its goal is to provide high speed packet processing for the IP packets destined or originated by a single IP address. The same IP address is used for the signaling and the media flows. Many signaling flows can share the same port while each media flow is associated to a single port.

The DPS processes the addressing information in the header of IP packet leaving untouched the payload. Therefore, it can provide packet forwarding, packet redirection, Network Address Translation (NAT), port translation. The DPS is instructed how to manage the IP packets by an external controller. For the vSBC the controller is the IBCF component.

Besides these basic packet processing functions, the DPS can be optionally extended with firewall and security gateway components. The front-end role of DPS has been implemented by dedicated hardware components such as Network Processing Unites (NPUs) in traditional SBC products, performing wire-line speed processing rate.

The challenge of pure software implementation is to reach a performance level comparable to the hardware based version. The approach we are following is to adopt acceleration technologies such as DPDK that is available in Intel x86 architectures. Whenever DPS virtual machine is executed on a DPDK enabled CPU, it considerably increases its performances, while not being constrained to run only on such type of hardware platform. In the latter case of course we will experience poorer performances.

4.3.5.2. Border Gateway Function

The Border Gateway Function (BGF) provides real-time media stream adaptation when media transcoding is required. This 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.

For sake of simplicity we use the term transcoding for indicating also the transrating function. While transcoding transforms the algorithm used for coding the media stream, transrating changes the sending rate of IP packets carrying media content. Both transcoding and transrating are key functions for allowing interoperability between diverse media endpoints. Transcoding is a highly intensive computation process implemented by dedicated Digital Signal Processors (DSPs) hardware devices.

Moving BGF to pure software implementation suffers from dramatic decrease of the number of simultaneously processed media streams. By embracing cloud elasticity paradigm, this issue could be mitigated by running more BGF instances. This is in fact the first strategy we are going to adopt. However, this approach could be not appropriate to all situations. To improve BGF performances we shall move to hardware acceleration techniques. The DSP technology is well known in telco field but is not usually adopted in IT domains where however similar problems exists about efficient graphical processing. Graphical Processing Units (GPUs) has been developed and can now be considered commodity hardware almost easily available in datacenters. The BGF can benefit from GPU technology using it to run transcoding and transrating algorithms. The BGF we are studying and implementing has the capability to use GPUs whenever they are available in the execution environment of the VMs hosting the BGF instances.

Summarizing, the BGF will implement pure software transcoding and transrating algorithms. It can be instantiated multiple times to provide a pool of resources according to the actual need or real-time media processing. If GPUs are available, they can provide hardware acceleration to increase BGF performances.

4.3.5.3. Lifecycle

The network function lifecycle was discussed in a general way in [D2.41].

The lifecycle of the SBC includes: Start, Stop, Monitor, and Configure.

4.3.6. Interfaces

The most interesting interfaces are the ones used for controlling the internal components.

DSP ctrl interface instructs the Data Plane Switch to perform forward, NAT, redirect of IP packets.

BG ctrl interface instruct the Border Gateway Function to perform media stream transcoding and transrating between two codecs.

T-Ve-Vnfm interface instructs the O&M component to implement VNF lifecycle.

Detailed description of these interfaces is ongoing. It will be reported in future deliverables.

4.3.7. Technology

The technology we are going to use for vSBC implementation derives from NetMatch-S CI product by Italtel S.p.A, a pure software edition of Italtel SBC based on virtual platforms, specifically designed for deployment in Data Center/Cloud Environments. [SBC_Italtel]

The transcoding feature provided by the BGF module benefits from GPU-acceleration. The overall system consists of a standard commodity server, with a commercial GPU board hosted in a PCIe bay. The GPU board is the Nvidia Tesla k20x platform; the initial development will be based on the use of the Linux operating system, and the

KVM hypervisor. This way, both the API remoting and the GPU passthtrough approaches can be adopted. The limitation of this approach, due to the lack of support of SR-IOV by the Nvidia Tesla architecture, is that in the pass-through case the entire GPU will have to be assigned to a single VM. However, in this initial stage of the development, this is not felt as a major issue; also, it will be easily removed by upgrading the GPU board to the Grid family. The adopted GPU programming language is CUDA. The developed virtual function is a video transcoding unit, that will represent one of the VNF component of the virtual SBC use case. The transcoding will be performed between domains using the Google VP8 coding scheme, and the ITU-T H.264.

4.3.8. Dimensioning and Performances

At present, we don't have absolute numbers related to the expected performances of the vSBC. Nevertheless, we have target requirements about the expected behavior of its internal components. They are summarized here:

- The DPS component shall be able to process all the traffic offered to and originated by a single network interface.
- The SIP LB component shall be able to process all the ingress SIP messages.
- Both IBCF and BGF components shall provide market sensitive performances. For instances it shall support 10, 25, 50, 100, 500, 1000 simultaneous instances. For the commercial product, each component size will be associated to a license fee.

4.3.9. Future work

The work done in this initial stage brought to clear identification of the system architecture and the acceleration technologies that are necessary to provide a Session Border Controller with performances comparable to the legacy hardware versions available in the market.

The backlog of our work can be summarized in the following list:

- Continue studying acceleration technologies and their applicability to SBC context.
- Define detailed description of internal interfaces.
- Complete the implementation of vSBC internal components.
- Test in particular the most innovative components such as DPS and BGF in a deployment scenario with and without hardware accelerators available.

4.4. Traffic Classification

4.4.1. Introduction

Deep Packet Inspection (DPI) is a technology that inspects IP packets at Layer 2 through Layer 7. This includes headers and data protocol structures as well as the actual payload of the message. DPI is used to prevent attacks from viruses and worms at wire line speeds. More specifically, DPI can be effective against buffer overflow attacks, Denial of Service (DoS) attacks, sophisticated intrusions, and a small percentage of worms that fit within a single packet. A classified packet can be redirected, marked/tagged, blocked, rate limited, and of course reported to a reporting agent in the network.

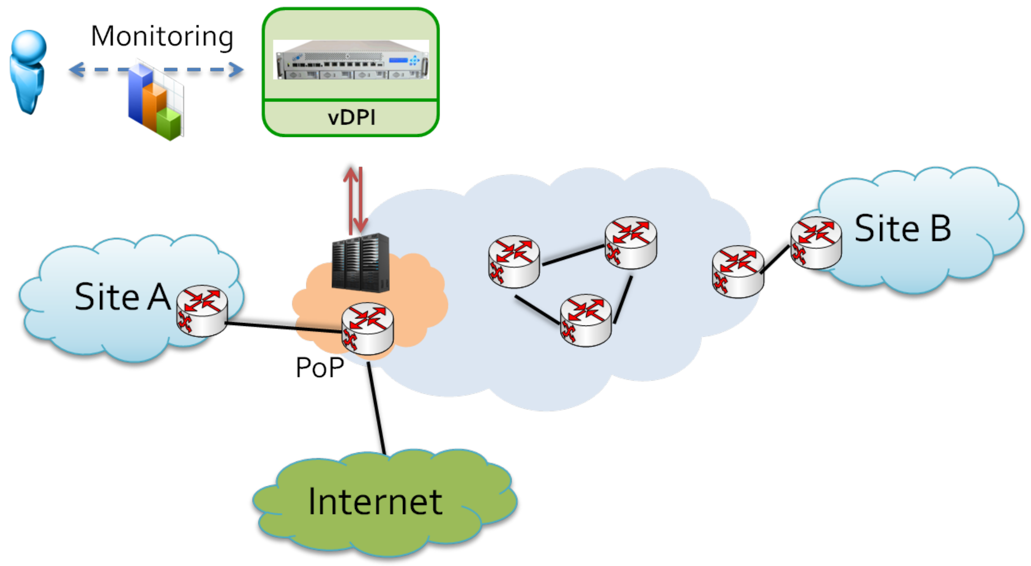

Figure 4-8 DPI used for Monitoring and Statistics for Enterprise Customers

The exploitation of DPI methods for traffic classification is built around two basic assumptions: (i) third parties unaffiliated with either source or recipient are able to inspect each IP packet's payload and (ii) the classifier knows the relevant syntax of each application's packet payloads (protocol signatures, data patterns, etc.).

The proposed DPI based approach will only use an indicative, small number of the initial packets from each flow in order to identify the content and not inspect each packet. In this respect it follows the Packet Based per Flow State (PBFS). 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.

For the second assumption, a library of protocol signatures and filter strings has to be built. This library may be consulted, so the protocol can be accurately detected. Nevertheless, the larger it gets the more resources for the classification procedures are needed. The signature library needs to be constantly updated as application protocols evolve or as new protocols emerge.

The Traffic Classification VNF implementation for T-NOVA will build on top of readily available Opensource libraries providing protocol signatures (i.e. OpenDPI, nDPI [REFnDPI]) to be exploited by the Inspection Engine. Additionally the protocol signature library will be expanded in order to accommodate video delivery specific protocols.

4.4.2. State-of-the-art

4.4.2.1. Inspection Algorithms

The most popular algorithms exploited identification through DPI for signature analysis and mapping can be summarized as:

- Automaton (Regular expression matching) – Tracks partially matched patterns in the data string by transition in either a Deterministic Finite Automaton (DFA) or Non-DFA implementation that accepts strings in the pattern set. The main drawback of this approach is that either the memory space needed to keep the patterns is big in expense to the lower time complexity (Non-DFA) or the opposite for the DFA method. However it is the most broadly used method for pattern matching.
- Heuristics – A heuristic can check a block of characters in the window suffix for its appearance in the patterns. It determines whether a match occurs and moves to the next window position if not. This approach provides the ability to skip characters not in a match to accelerate the search.
- Filtering Based – This approach searches the data string for necessary pattern features and quickly excludes the content not containing those features. A very common way of applying this approach for text filtering is using well-known Bloom filters. This method is useful for the extraction of substrings from regular expressions, and filtering text with them or checking for different pattern lengths.

4.4.2.2. vDPI Implementations

The possible reference implementations for vDPI are described here.

- QOSMOS

  Qosmos DPI as a Virtual Network Function Component (VNFC) complies with an official use case standardized by ETSI in July 2013. This new Qosmos product runs in a virtual machine and uses optimized interface to feed application information and metadata to other integrated components, together forming virtual networking equipment (VNFs) such as Service Routers, GGSN, PCEF, BRAS, ADC/Load Balancers, Network Analytics, NG Firewalls, WAN optimization, etc.

  Qosmos DPI VNFC is based on Qosmos' flagship product ixEngine, which is already established as the de facto industry-standard DPI engine for developers of telecoms and enterprise solutions. ixEngine identifies and extracts information traveling over networks in real time, providing a true picture of the traffic by identifying protocols, types of application, and extracting additional information in the form of metadata. Equipment makers, telco and enterprise software vendors, and cloud service providers use Qosmos to gain application awareness, accelerate time to market and benefit from continuous signature updates

- Wind River

Wind River Content Inspection Engine, a high-speed pattern-matching DPI solution that can match large groups of regular expressions against blocks or streams of data, is a scalable, cost-effective approach that runs entirely in software. It is designed for both single-core and multi-core processors. Content Inspection Engine is ideal for applications that need to scan large amounts of data at line rate, such as intrusion prevention (IPS), antivirus (AV), unified threat management (UTM), and other DPI systems. [REFWind]

- IPOQUE Protocol and Application Classification with Metadata Extraction (PACE) 2.0 (was OpenDPI)
  PACE 2.0 is ipoque's next generation software that identifies thousands of applications and services and provides deeper insight on application attributes such as real-time performance metrics, all from a single solution. PACE 2.0 combines the power of our application classification (PACE) and decoding engine (PADE) into an all-in-one protocol and application classification engine also capable of advanced metadata extraction. This integrated solution makes it more efficient for customers to get deeper application insight for troubleshooting, security or subscriber analytics purposes. The granularity of detail is 100% configurable providing a completely scalable solution. [REFPACE]

- nDPI
  nDPI is a ntop-maintained superset of the popular OpenDPI library. Released under the LGPL license, its goal is to extend the original library by adding new protocols that are otherwise available only on the paid version of OpenDPI. In addition to Unix platforms, it also supports Windows, in order to provide a cross-platform DPI experience. Furthermore, they have modified nDPI do be more suitable for traffic monitoring applications, by disabling specific features that slow down the DPI engine while being them un-necessary for network traffic monitoring. [REFnDPI]

4.4.3. Requirements

This subsection provides insight to the most important requirements that this VNF will need to fulfil. The intention of the development of the Traffic Classification is not to build a ready to market VNF rather than have a VNF with enough functionalities and complexity in order to support the T-NOVA proof of concept validation. Performance related non-functional requirements will be fine grained as soon as the initial validation steps have concluded.

4.4.3.1. Functional Requirements

The traffic Classification VNF:

- Shall be able to analyze incoming traffic exploiting DPI methodologies
- Shall be able to provide precise identification of RTP flows
- Shall be able to prioritize and differentiate traffic, according to configured policies

- Shall provide appropriate functional APIs, for statistics retrieval and configuration.
- Shall be modular and configurable upon service requirements.
- Shall be able to support QoS features
- Shall be able to provide analytics of the available monitored flows information
- Shall provide internal monitoring (VNF specific) information to VNFM in order to allow efficient scaling decision taking.

4.4.3.2. Non-Functional Requirements

The Traffic Classification VNF:

- Shall function at an acceptable and scalable level of performance.
- Shall not introduce latency to the inspected flows; latency as result of the queuing based on the prioritisation does not apply here.
- Shall be able to operate near line rates for simple inspection rules and long living flows.

Requirements related to the performance of this VNF will be updated and refined as soon as the initial implementations will be thoroughly tested in the actual deployment environment.

4.4.4. Architecture

Figure 4-9 provides an overview of the high-level Traffic Classification VNF architecture. The VNF comprises three VNF components namely the DPI VNFC; the Classification VNFC; and the Statistics VNFC. The role of each VNFC is explained summarized below.

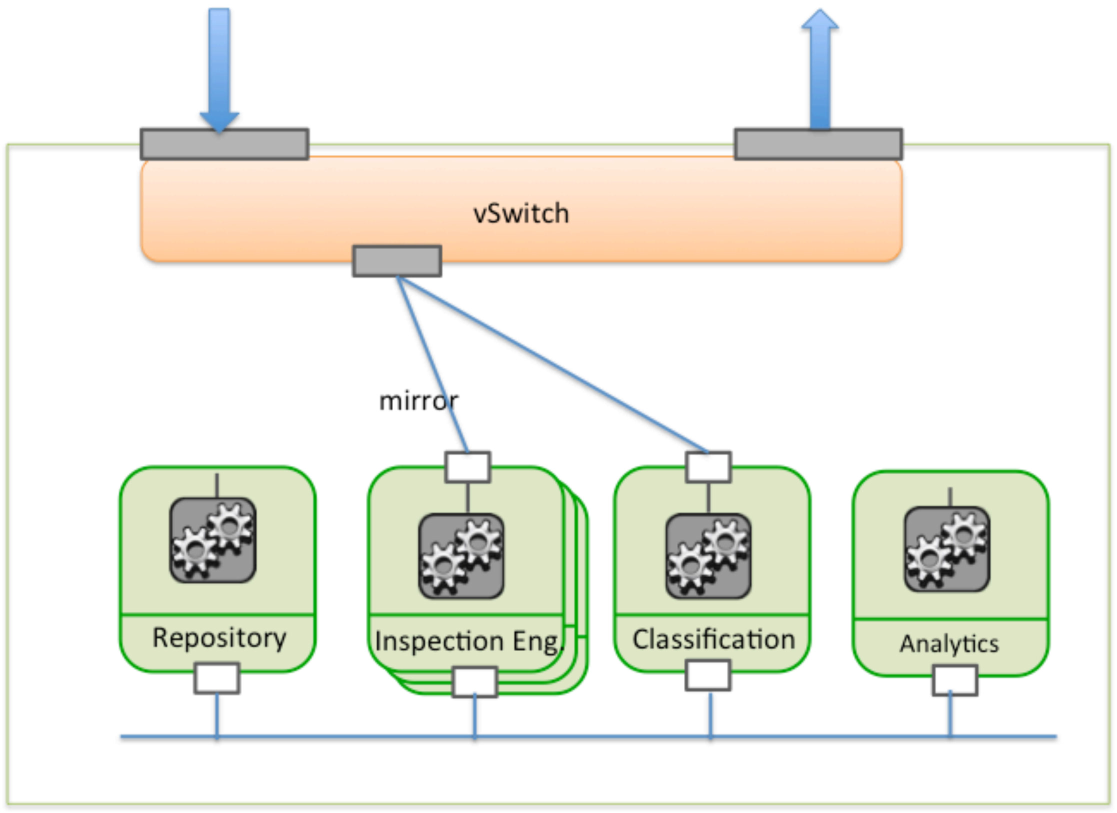

Figure 4-9 vDPI VNF Architecture

Inspection Engine VNFC, this component is responsible for the most processing intensive functionality of the VNF. The component will implement the filtering and packet matching algorithms in order to support the traffic classification. This component support a flow table (exploiting hashing algorithms for fast indexing of flows) and an inspection engine for traffic classification. As soon as a new flow has been identified by the DPI engine, the flow register will be updated with the appropriate information and communicated to the Classification VNFC.

Classification VNFC, this component is responsible for routing and packet forwarding process. The component will accept the incoming traffic, consult the Flow Table for classification information for each incoming flow and then apply mark the traffic or apply QoS policies accordingly. It is noted that traffic is mirrored to the DPI VNFC in order these modules to work in parallel. In case that the VNFCs are not executed on the same host, thus mirroring the traffic might include additional overhead, other more complex and less efficient VNF graphs should be implemented (see VNF Consideration section).

Repository VNFC, this component is responsible for storing the raw information of the monitored and classified flows. This component will be implemented as a round robin database that will be possible to store monitoring and classification information for a specific time window.

Analytics VNFC, this component is responsible for the analysis and statistical processing of the classified traffic information. The component will also expose Web based API allowing the access to this information from the VNF tenants (external to the T-NOVA SP). This component will fulfill the requirement for passive operation of the VNF allowing collection of historical traffic information by the customer.

4.4.4.1. VNF Graph considerations

Depending on the anticipated performance yield by the VNF, there are different deployment scenarios that can be supported for these NVF components, yielding different VNF graphs and restrictions in the VNF management and lifecycle. The following cases may be considered:

- Deployment on the same host without packet acceleration support

  In this case, the traffic mirroring needs to be achieved at the network hypervisor level (i.e. OpenVirtual Switch) with the processing cost of the packet duplication. The VNF graph in this case has two brunches on towards the Classification VNFC and another one towards the DPI VNFC

- Deployment on the same host with packet acceleration support
  In this case, the traffic mirroring is achieved without packet copy penalty as the DPDK support zero copy method and both VM access the data directly on the share memory pool (hugepages) available at the host. This method is the most efficient performance wise. The VNF graph is the same as previously.
- Deployment on different hosts.
  In this case, the traffic needs to be switched to different hosts running the VNFCs. We can have two variations (i) traffic is firstly forwarded to the DPI and then to the Classification VNFC (in sequence) or (ii) traffic is mirrored on the TOR switch supporting the NFVI, hence the same VNG graph as above is preserved. In case the variation (i) is selected, the VNF graph is altered in a way that the traffic parses the two VNFCs in sequence. If no identification has yet occurred, the classification VNFC forwards the traffic with no policies applied. As soon as the traffic is identified the classification VNFC will apply the policies instantly. This method is considered as the indicative method in order not to cause latency to the forwarded traffic due to delays in the packet inspection process.

4.4.5. Functional description

Given the nature of this VNF it is important that the packet handling mechanisms used are optimized in order to guarantee performance and stability with the minimum cost in resource utilization. For this reason various mechanisms are being considered in order not only to allow the development of a VNF that is performant under stringent platform capabilities but also develop a version that is more open and generic. The most notable functional blocks of the VNF at this stage is the Packet Handling block, actually putting the platform requirements.

4.4.5.1. Packet Handling

Currently one of the most promising frameworks, widely used by the industry is the Intel DPDK framework, which we discuss in previous sections. This framework supports the IVSHMEM memory handling mechanism available in Linux OSes. IVSHMEM is a mechanism for sharing host memory with VMs running on that host, providing: i) zero-copy access to data; ii) interrupt/signaling mechanisms; iii) optimizing guest/guest and host/guest communications. The initial implementation of the platform where the Traffic Classification will developed, tested and validated will exploit this mechanism along with DPDK. However in the future a more native to Linux kernel implementation of DPDK (i.e. DPDK-netdev) will be considered in the place of the OVS-DPDK that is used now.

The functional description of this platform although not strictly related to this VNF is important in order to understand the bottlenecks of the implementation. Initially a specific memory address space in the host is allocated as a buffer that will store the incoming packets received by the physical interface of the host compute node. IVSHMEM is used in order to allow VMs that are operating on the same host to have access at the same time to this memory space. In this case no additional configuration needs to be performed to the OVS instance that controls the networking of the VMs. In the alternative case were mirroring would be required an the memory sharing mechanism was not supported, configuration and packet duplication at the OVS would be needed thus decreasing the performance of the platform.

IVSHMEM mechanisms provides zero copy access to the packets between the host and the guest VMs, meaning very low to zero latency, in copying the packet data from the physical network interfaces to the VNFs. Furthermore, IVSHMEM offers the option to have higher packet throughput, when the application is trusted, thus providing a basic service layer scheme, correlating trust and packet throughput. Another option available is the VM-VM security capability, enhancing the inter VM packet buffer security sharing by an additional buffer allocation (via. memcpy). The latter option enables our system to apply restrictions on the packet buffer that each VNF can read and modify, so as to prevent overwriting, or modifying packets of another VNF without permission. Also VNF packet buffer restriction can provide concurrency control among different VNF operations. Last but not least, DPDK's feature IVSHMEM gives the opportunity to work and process data packets without using the Linux network stack, giving more flexibility and more speed on how to handle the incoming packet queues.

4.4.5.2. Classification

This VNFC controls the packet policing, prioritisation and forwarding for the traffic that passes through this VNF. The implementation of this VNF will be based on Linux OS, using native Linux Kernel capabilities and user space control software and libraries. The role of this VNFC is to be able to forward the traffic received by the ingress interface of the VNFC, and as soon as identification information for a particular flow is signalled to this VNFC to apply certain policies and prioritisation. The policies will be signalled via the VNFM according to tenant specification and needs. In order to support various QoS schemes, the VNFC will be able to apply DSCP marking on the forwarded flows as well as support traffic queuing and prioritisation specifically for the tenant flows. Given the application and context awareness that is provided via DPI methods, classification rules can be automatically applied based on more abstracted policies.

4.4.6. Lifecycle

The Traffic Classification function lifecycle follows the T-NOVA lifecycle template with the basic functions of Start, Stop, and Configure. As previously mentioned the Traffic Classification function will take advantage of Intel's DPDK libraries in order to accelerate the packet processing speed. This creates the need to identify in advance if the deploy environment is DPDK-enabled. The concurrent work on the Enhanced Platform Awareness under T-NOVA WP4 [D4.01] is very much related to this. In the case that a DPDK enhancement does not exist, a non-DPDK solution should be applied.

The basic lifecycle stages that are considered at the moment are start, configure, monitor and stop. Depending on the platform capabilities we can currently see to cases (i) the DPDK is not supported – non-DPDK version of Traffic Classification is on-boarded and (ii) DPDK is supported – DPDK version is on-boarded.

Especially for the second case (i.e. case ii) there are some steps that need to prepend the bootstrapping of the VNF VMs. The critical configuration items are:

- DPDK network driver loading, on the required NICs. (this is assumed that is taken care by the VIM and the NFVI)
- The memory Hugepages setting e.g. number of Hugepages, size of a Hugepage.

As soon as the VMs are bootstrapped OVS-DPDK needs to be configured in order to allow shared access and/or port mirroring.

The generic framework for VNF lifecycle stages has been discussed in D2.41, and is considered also for this VNF [D2.41]. As soon implementation proceeds, each stage will be documented and refined.

4.4.7. Interfaces

The most important interfaces for this VNF are identifies as:

i) internal – used for conveying information between the VNFCs
   a. TC-flow, the interface that signals flow identification information between the Inspection Engine and Classification VNFCs
   b. TC-monitoring, the interface that conveys flow monitoring information (i.e pps, errors, identification, TTL etc) that from the inspection engine to the repository.
   c. TC-an-rep, the interface the conveys information between the analytics VNFC and the repository

ii) external – used for conveying signaling and information between the external to VNF entities (i.e. the customer, the monitoring, the VNMF).
   a. T-Ve-VNFM for control of the VNF lifecycle.
   b. TC-rmon allowing access to the analytics by the customer

Detailed description of these interfaces is ongoing. It will be reported in future deliverables.

4.4.8. Technology

The proposed DPI solution is based on the popular and ntop-maintained super-enhanced OpenDPI library nDPI, released under LGPL license. However, further modifications and additions will be added in order to capture specific cases that the original filters and library do not support, mainly with respect to classification and identification of media flows delivered over RTP protocol.

4.4.9. Dimensioning and Performances

In order to initially address the dimensioning assessment required for this VNF we need to establish some baseline tests for the packet forwarding capabilities and introduced latency related to the packet handling framework used at the host where the Traffic Classification VNF is running. In the future a further measurement on the VNFs resource utilization and performance on different type of traffic workloads will be conducted. The traffic forwarding results provided are based upon different system environment settings. The general overview of the setup can be described as a traffic loop, between two individual machines, the traffic generator and the host where the Traffic Classification VNF is running.

Experimental Setup

The software used for the traffic generation was based on NetMap library for optimized traffic generation on our available hardware. The traffic generator was able to approximate the 1Gbps line rate generation on supported network cards. On the same node the generated packets after passing through the Device Under Test (DUT) (compute node, VNFCs) are arriving. The received packets are then analyzed for a various metrics i.e. forwarding rate, one-way delay, jitter and loss.

On the compute node side DUT side, where the vDPI was running different configurations and packet-handling frameworks were used i.e. PF_RING, DPDK, OVS_DPDK (with 1 and 2 VM configuration). The experimental setup is depicted in Figure 4-11.

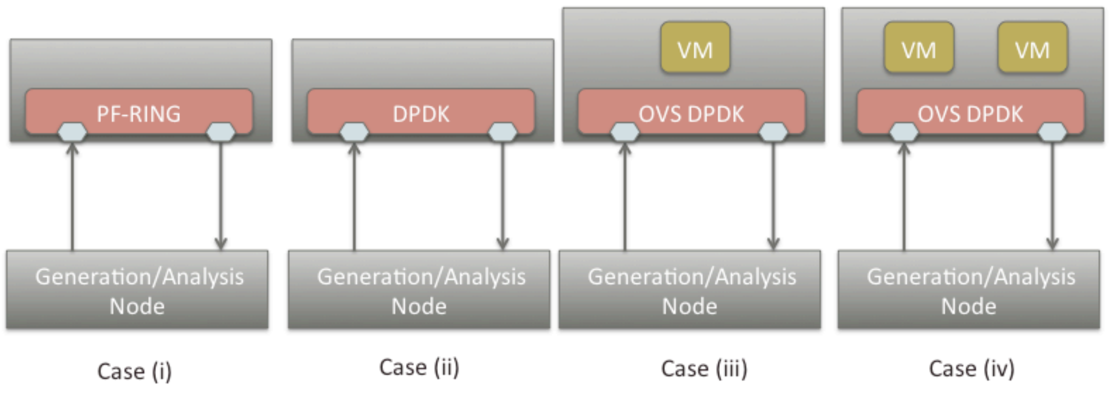

Figure 4-10 vDPI experimental Setup

In detail the following test scenarios were performed:

i) PF_RING - the PF_RING bridge, with the PF_RING module, was used, on top of the native network drivers

ii) DPDK – baseline, everything running on the host (no VMs), i.e the host forwards directly the packets to the second "exit" physical network interface, after performing an LPM (Longest Prefix Matching) lookup.

iii) DPDK (single VM) – Single VM providing DPI and classification over OpenVSwitch DPDK with IVSHMEM functionality.

iv) DPDK (two VMs) – Two VMs distributing the functionalities of the vDPI (one running the DPI and the other running the classification/forwarding)

The case where the forwarding is based on native Linux kernel (bridge or L3) is not considered at all, as it is well established that its performance is quite poor when the packet size is less than 512bytes.

The retrieved results are presented in Table 3 and illustrated in Figure 4-11. The Packet forwarding capability can be easily deducted by these statistics. It is important to notice the improvement in the performance that DPDK provides in the baseline case (i.e. PF_RING vs DPDK), which is comparable. Additionally the introduction of OVS with DPDK improves furthermore the performance.

Table 3 Loss ratio (%) Packet size vs packet handling framework

Loss %				
Packet Size	**PF_RING**	**DPDK**	**OVS DPDK 1 VM**	**OVS DPDK 2 VMs**
64	53.93	30.00	16.23	18.45
128	54.88	2.47	9.56	2.5
256	43.25	4.40	7.78	1.3
512	5.35	2.77	5.51	13.1
1024	6.01	2.94	2.40	5.5
1280	2.53	4.80	5.26	0.4
1518	2.97	4.81	1.90	0.39

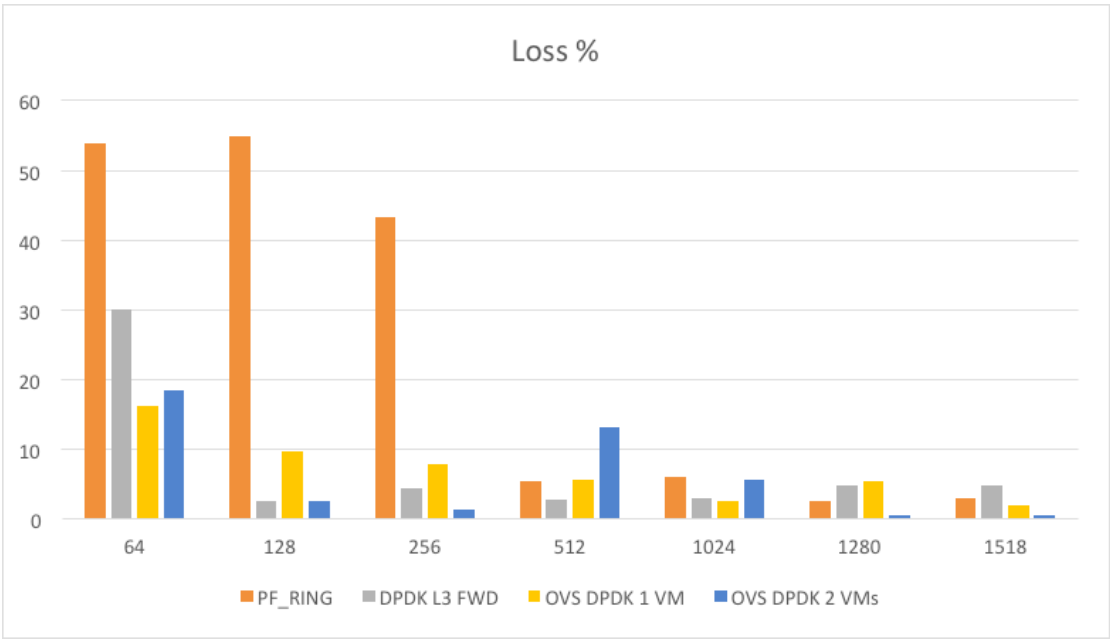

Figure 4-11 Loss(%): Packet Size vs Packet handling framework

Next table (Table 4), provides the measured delay end-to-end through the DUT. Comparing the first two cases the delay is fairly improved, when using the DPDK. However the delay is slightly increased when introducing the OVS in the packet path. Concluding, the tradeoff regarding the packet loss is in favor of selecting the combination of OVS and DPDK.

Table 4 Delay (ms) Packet Size vs packet handling framework

Delay ms				
Packet Size	**PF_RING**	**DPDK L3 Forwarding**	**OVS DPDK 1 VM**	**OVS DPDK 2 VMs**
64	1.5	1.1	1.12	1.2
128	0.9	0.52	0.63	0.97
256	0.848	0.12	0.27	0.86
512	0.99	0.299	1.23	1.21
1024	1.04	0.75	1.21	0.3
1280	1.71	0.89	0.87	0.76
1518	1.07	1.53	1.57	1.15

The above provided results are illustrated in the following Figure 4-12.

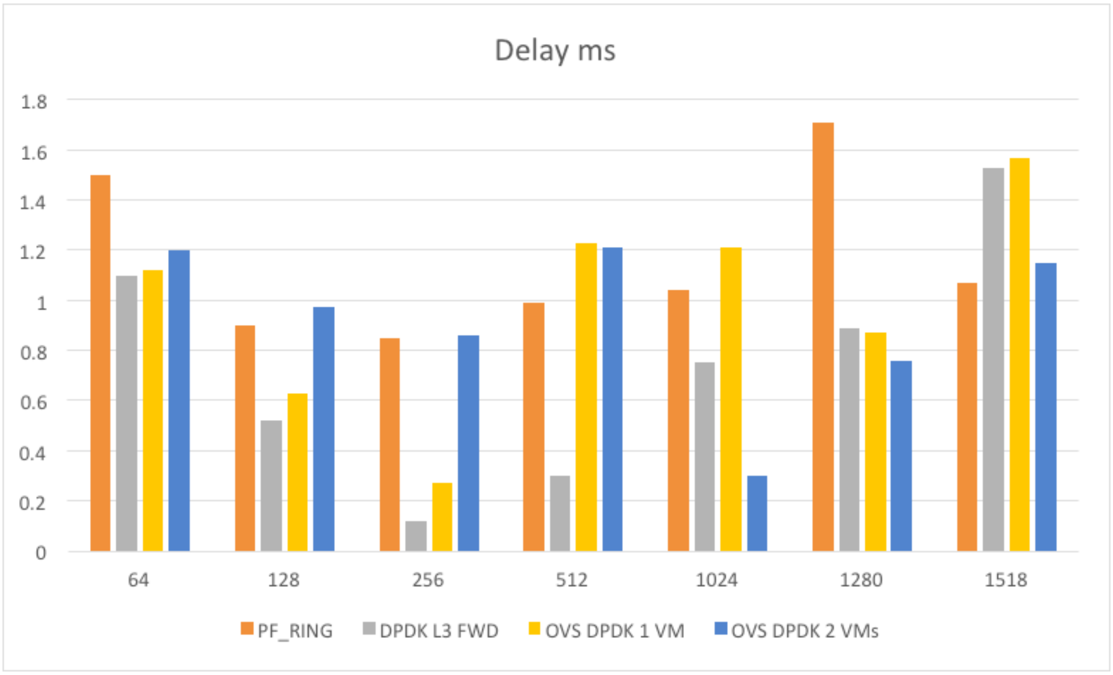

Figure 4-12 Delay (ms): Packet Size vs Packet handling framework

The main scope of these performance tests is to demonstrate the significant improvement not only in packet processing and forwarding times, but also in the total packet number forwarding performance, we can achieve using the DPDK framework. On parallel additional similar tests are provided for various more fine-grained scenarios and host compute node configurations in Deliverable D4.01 [D4.01].

4.4.10. Initial Implementation

The currently implemented Traffic Classification VNF comprises two VMs connected to the deployed Open Virtual Switch (OVS), all traffic will be sent to both VMs simultaneously, in other words a traffic mirroring. One VM (i.e. Inspection Engine) will perform the flow inspection, assortment, and application detection and will send a recap of the collected data to the other VM (i.e. Classification/Forwarding), which will correspondingly decide on the proper classification based on its configured policies. The inter-VM communication is achieved through their TAP interfaces. This upper layer of TAP communication provides an efficient and stable communication channel for the different VMs, as it does not interfere with the main network traffic that is inspected by the VNF, and establishes an independent path for packets related to update and communication messages. This fast, reliable, and independent from the traffic workload communication will guarantee a real-time update of the system. It is obvious that this method of communication applies for VMs that are placed within the same compute node. In the case were the VMs are placed in different hosts the communication will need to be supported by a separate network in order to avoid mixing intra-VNF communication data with the actual forwarded traffic.

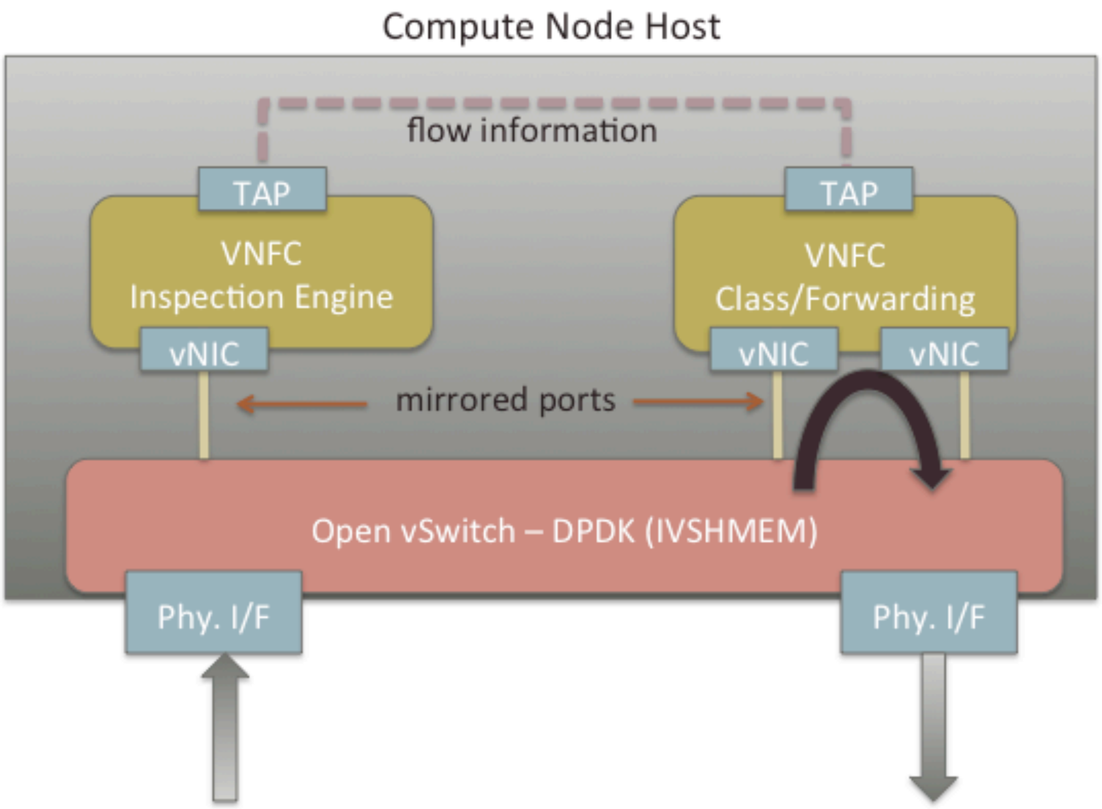

Figure 4-13 Traffic classification VNF initial implementation

4.4.11. Future work

4.4.11.1. Directions in the exploitation of vDPI in 5G and FI

The provision of multimedia content and services has been recently increased significantly, creating the need for both better quality and better exploitation of network resources. QoS is an aspect addressed in 5G. Another issue addressed in 5G is the aspect of intelligent environment-gnostic services. A vDPI function with traffic classification and upper-layer application identification is extremely vital to the issues addressed by 5G. The vDPI will not only address the matter of intelligent services, as it will offer the capability to make decisions upon the traffic application type, but will also be able to offer an efficient QoS mechanism for high network performance.

4.4.11.2. Development Time plan and Features

The first of developing the vDPI is to benchmark the capabilities and performance of the environment on which the vDPI will be deployed. This includes implementing and comparing different setup implementations, basically a DPDK-enabled versus a standard linux environment, for processing packets. The second stage includes the implementation of the vDPI function on both environment options and benchmark the performance of the 2 individual settings. Furthermore, the difference in performance and configuration of the 2 setups should be analyzed. The last step of the process is the implementation of a system capable of deciding based on the deploy parameters which option to prefer and proceed to initiate.

4.5. Virtual Home Gateway

4.5.1. Introduction

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 (VGH), Virtual Residential Gateway, Virtual Set-Top Box or Virtual User Premise Equipment. The following sections aim to provide a brief description of the proposed virtual function along with the requirements, the architecture design, functional description, technology etc.

4.5.2. Scope & Intentions

By Creating a VGH this will aim at putting parts or all of functional aspects usually implemented by a regular residential gateway, at the end user' s premises to another place within the network, closer to the core.

The foreseen benefits include:

- **Lower capital** for deploying the same installed base (assuming increasing returns to scale for all networking operations by using big servers in place of small devices)
- **Lower operational costs** by limiting the need to provide customer support. The end user will not have to interfere with the equipment or update any software
- **Less time to market** by having the possibility to deploy new functionalities by pushing software and scaling up compute power and network capabilities.
- **Reduce power consumption**. It also opens up some green perspective by reducing the overall power.

Although Residential Gateway Specifications are thoroughly described in TR-124 [TR-124], the current contribution to the T-NOVA project does not aim at supporting the whole stack of protocols. 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.

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.

4.5.3. State of the art

Several academic and industrial initiatives have decided to study or implement part of the VHG scenario whilst standardization bodies are also manifested and interested in the proposed solution.

4.5.3.1. Standardization bodies and Forums

ETSI mentioned the VHG Use Case in [NFV001] under the chapter "Virtualization of the Home Environment". It especially describes how Residential Gateway (RGW) and Home Set Top Box (STB) merge into a unified virtualized device providing both connectivity and value-added media services.

The Broadband Forum has been working since 2012 on a couple of studies and projects addressing virtualization opportunities in home and business gateway architectures. Their ongoing works on Network Enhanced Residential Gateway (NERG) WT-317 recall the IETF point of view. Both organizations agreed a formal liaison relationship in 2013 [IETF2013].

Albeit the Home Gateway Initiative (HGI) has not yet published any document in virtualizing with the NFV approach, they have included Java Virtual Machine in the specification for the HGI Open Platform (3), which is a step towards a modular home gateway. This could be viewed as an alternative or complementary vision to the NVF based one.

EURESCOM also released a study describing how the virtual home gateway could be achieved. Although it's not embracing the NFV model, it presents gaps and challenges for this approach.

4.5.3.2. Academic Research

Insufficient academic research has been conducted focusing on the specifics of virtual home gateway virtualization in the context of NFV.

In their paper [Silva], Lopez Da Silva and al. describe alternative architectures to implement virtualization of the routing part of the Home Gateway, along with the deployment of fiber connectivity.

Cruz and al. in [Cruz], propose architectures for both the network part of the access network, the virtualization of the gateway and virtualization specific issues. They explain that part of the functional stack of a home gateway could be mutualized (mentioning the collocation of services like DHCP or NAT) instead of being deployed as a standalone VHG. Parts of the open issues regarding virtualization (VM migration and vHG Pool management) are addressed in the NVF approach and by T-NOVA.

Research has been carried out in order to demonstrate that virtualizing the home gateway can be a good approach to save energy consumption [Gelas] by leveraging the easier energy savings in data centers.

More recent research [Mikityuk] has been conducted to study how a set top box could be virtualized by detailing the Thin Client and Zero Client approach.

4.5.3.3. Industrial projects and products

Other research is currently being carried out by industrials. At least 3 initiatives are currently rolling out the first virtual home gateway with the common goal: being the first operator to deploy vCPE over NVF.

- **Telefónica and NEC in Brazil:** Telefonica presented a proposal for virtualizing customer premises equipment (vCPE) in a demo that illustrated how the equipment at the customer's residence can be simplified and how the management of the resources and services will improve [Nec].
- **Telstra and Ericsson in Australia:** Telstra and Ericsson have undertaken some trials of new technology being that could enable pay TV, broadband internet and other services to be delivered without customers needing a set-top box at home. The concept will also allow subscribers to access their pay TV subscription from any location via the Telstra networks making it possible for people to have one subscription for multiple services [TheAge].
- **China Telecom and Huawei in China**: These two telecos are working in the completion of virtualizing the CPR which aims to simplify deployment for enterprise customers [LR_CPE].

Another thing that should be noticed is the fact that all technology providers and telecom operators who are platinum sponsors of the OPNFV project are located outside the US and EU. The goal of this project is to make the OpenStack project fit the need for carrier grade cloud operation, which is basically the technology beneath the brand name NFV [OPNFV].

4.5.4. Requirements

4.5.4.1. NVF centric

1. vHG NFV MUST propose a technical way to migrate from existing deployed solution to fully NFV one.
2. vHG NVF MUST bring clear added value to the media experience of end user compared to existing solutions
3. vHG SHOULD use standards for its operation as much as possible.
4. vHG SHOULD use libre or open source software for its operation as much as possible.

4.5.4.2. Physical Gateway Centric

1. Applications deployed on the gateway MUST be compliant with a standardisation recommendation, or a de facto standard.
2. Applications deployed on the gateway MUST have the lowest impact possible on latency, throughput and jitter.
3. Applications deployed SHOULD be programmable through a standard and open interface.

4.5.5. Architecture

4.5.5.1. High level

Network perspective

Even if T-NOVA doesn't involve altering the current access network architecture, we present a Bird's-eye view (Figure 4-14) of the network where the box is usually located in today's configuration. A more complete description can be found in [Abgrall].

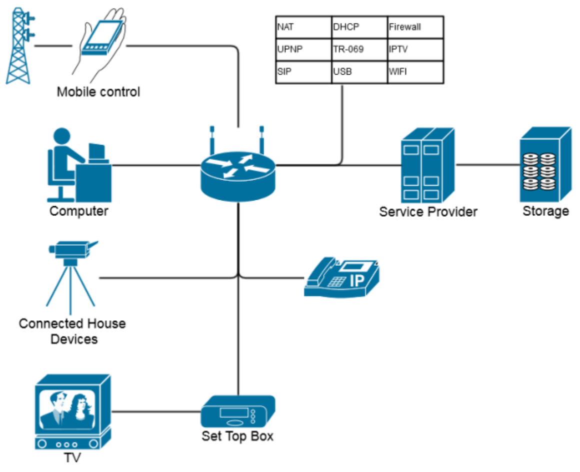

Figure 4-14 high level architecture for field deployment

With the novel vHG approach, we aim at replacing legacy devices like Set Top Box and Home Gateway with smaller devices with limited capabilities, with a vision to reduce both OPEX, CAPEX and devices fragmentation for the Service Provider. All the operations that were supported by the old devices will be relocated into the service provider datacenters, possibly as NVF.

Note that for a first approach, we will have the home gateway replaced by a new modular version promoted by [RD048] that supports deployment of vendor specific modules. Those modules will be used to deploy applications that will delegate the operations to VNF deployed on Service Providers datacenters.

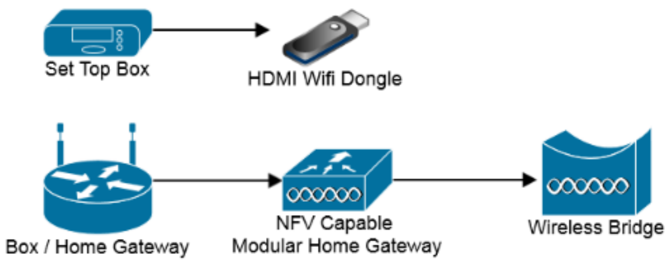

Figure 4-15 proposed hardware replacement

Box-side architecture

Regarding the boxes, some services may be kept for performance and compatibility reasons with existing deployment solutions. For example, from a deployment perspective, it's not clear yet if DHCP should be performed on the box like it's done now or remotely, leveraging DHCP option 82. On the other hand some services may be virtualized locally using for example Java/OSGI or remotely using VNFs.

Since we need a way to demonstrate our development in real world application scenarios, we will assume that boxes are able to run custom external modules, like boxes that are compliant with [RD048] as shown on Figure 4-16.

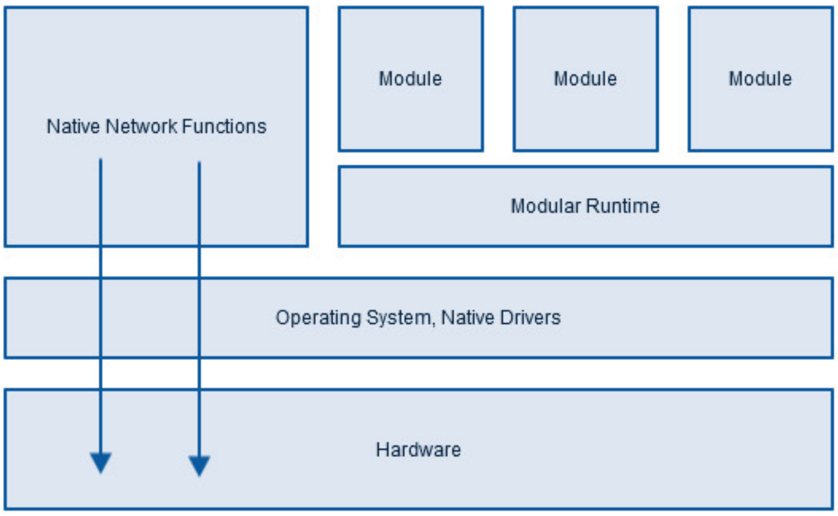

Figure 4-16 modular box architecture

Server Side Architecture

Several server architectures can be envisaged depending on the deployable service and on the operational constraints of the target infrastructure. Some open questions remain, and alternatives are exposed in [Cruz] as reported in Figure 4-17.

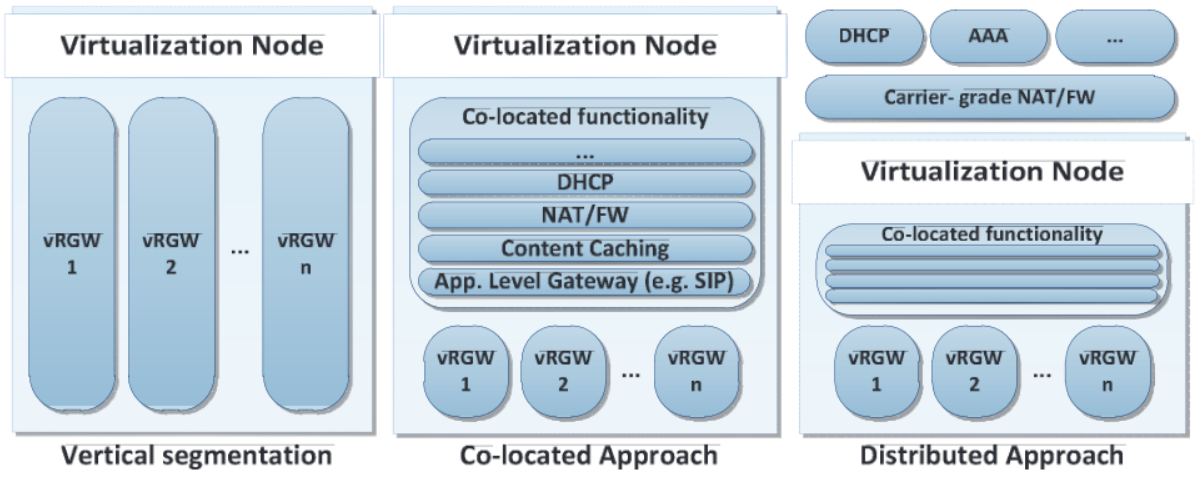

Figure 4-17 Architectural Optimizations for vHG

Vertical Segmentation.

The most straightforward scenario is having the vHG deployed in an infrastructure where there's a 1:1 mapping between the customer and vHG. It brings immediate transposition between the physical world and the virtual world, intrinsic privacy, and easy reuse of existing BSS modules. This approach however has certainly the most important footprint.

Co-located approach & Distributed approach

This approach is more modular as central servers can be optimized through their placement or provisioning. The approach could also have a distributed flavor, for example, DHCP relays could be installed in the access network.

Overall architecture

Next, we present how the box and the server will connect to perform network operations. Figure 4-18 shows the high level architecture as presented by ETSI in [NFV001]. This vision claims supporting vertical segmentation for the sake of simplicity of migration.

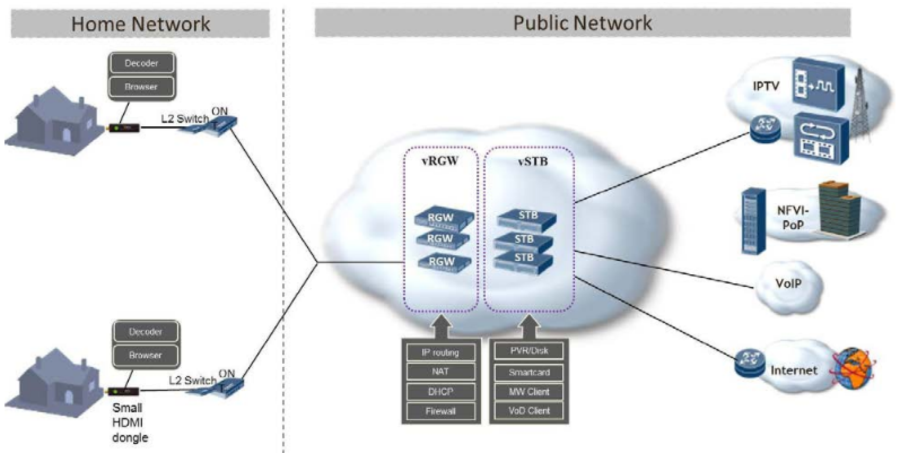

Figure 4-18 ETSI home virtualization functionality

A Transition architecture.

A transition architecture is needed (presented in Figure 4-19) to demonstrate the capabilities on NFV to be able to suit the need for carrier grade services over IP. To reach this point, we need a way to deploy part of the final scenario for selected network functions to prove that the scenario is viable.

This transition architecture, built on ETSI and HGI vision will allow us to develop and deploy a single NVF and plug it into the home gateway by deploying a small bridging application that will be responsible for forwarding to the appropriate VNF the user's request.

Another advantage of being able to deploy selected VNF resides in the fact that we will choose network functions that present the most added values for being deployed into an NVF backend, given the physical HG inherent constraints.

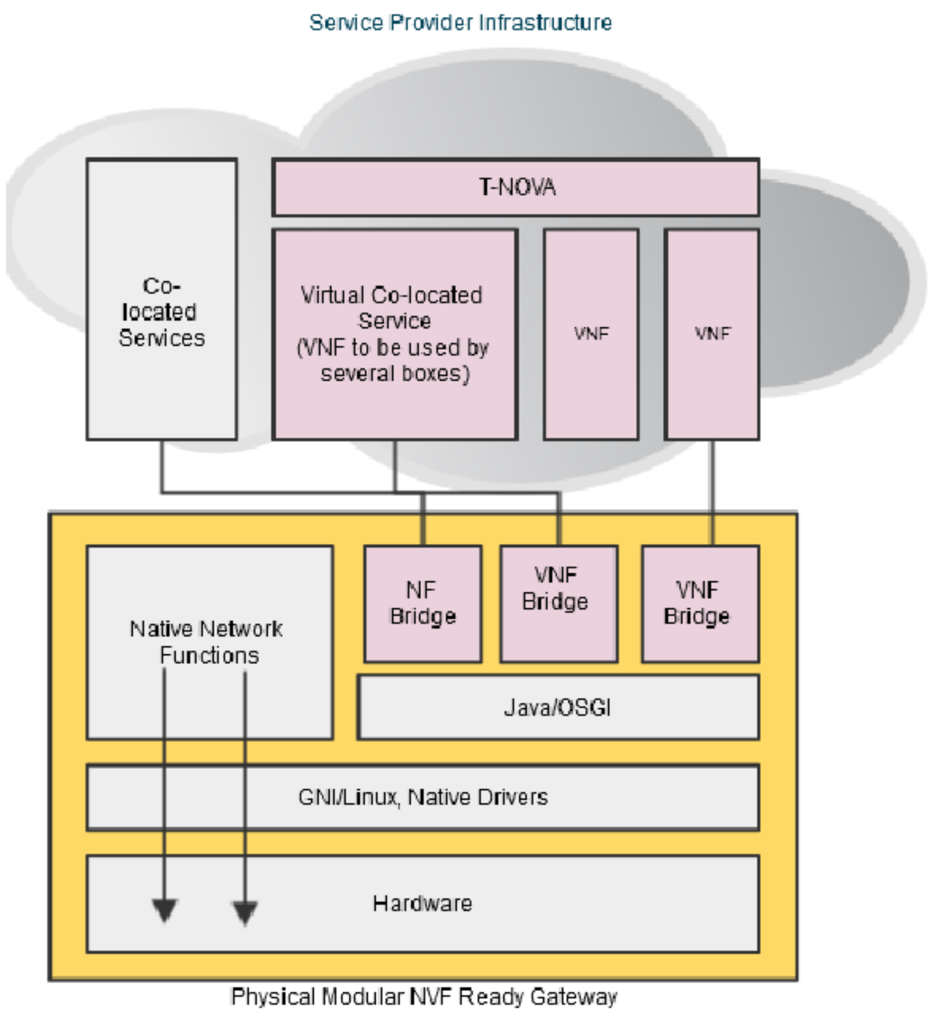
Service Provider Infrastructure
T-NOVA
Co-located Services
Virtual Co-located Service (VNF to be used by several boxes)
VNF
VNF
NF Bridge
VNF Bridge
VNF Bridge
Native Network Functions
Java/OSGI
GNI/Linux, Native Drivers
Hardware
Physical Modular NVF Ready Gateway

Figure 4-19 Transition Architecture

4.5.5.2. Low Level

To illustrate low level aspects of the VNFs that will be deployed, we will take a specific example of VNF which caches, transcodes and streams the most requested videos by gateway users.

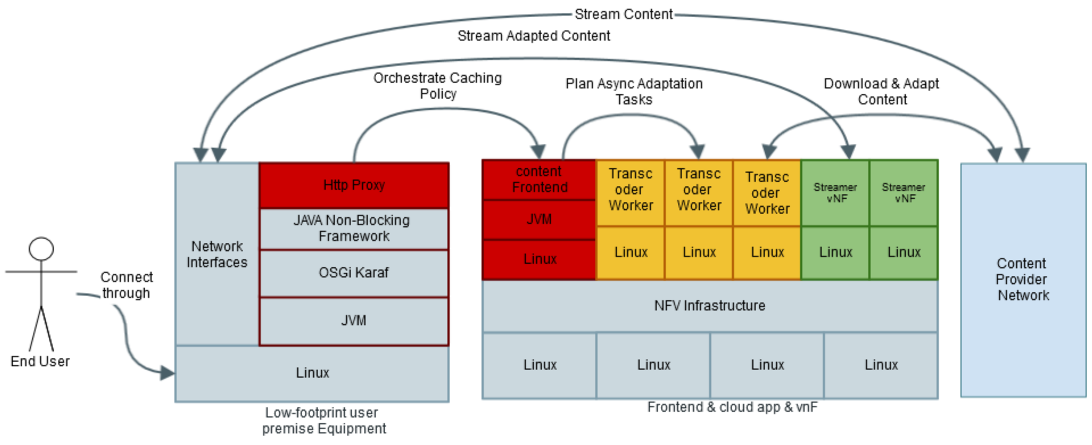

Figure 4-20 Low level architecture diagram for transcode/stream VNF

Figure 4-20 illustrates a modular gateway which acts as a 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's transcoded and moved to the storage shared by the streamers.

Once the video resource is available to the end user, the gateway redirects the user's request to the streamer, ensuring the best QoE possible.

4.5.6. Virtualization targets

Here's a list virtualization targets published by ETSI in [RD048]. We consider implementing part of this list, with a particular focus given to media aspects.

4.5.6.1. Main objective: Media virtualization Target

- **Streaming:** using methods such as HTTP Streaming and Zero Client.
- **Media Cache:** Support caching of different content types and formats
- **Content Sharing**: Possibility for the end user to be able to see their contents over any virtualized Home

4.5.7. Sequence diagrams

The sequence diagram presented in Figure 4-18 is associated with the example presented in section 4.5.5.2.

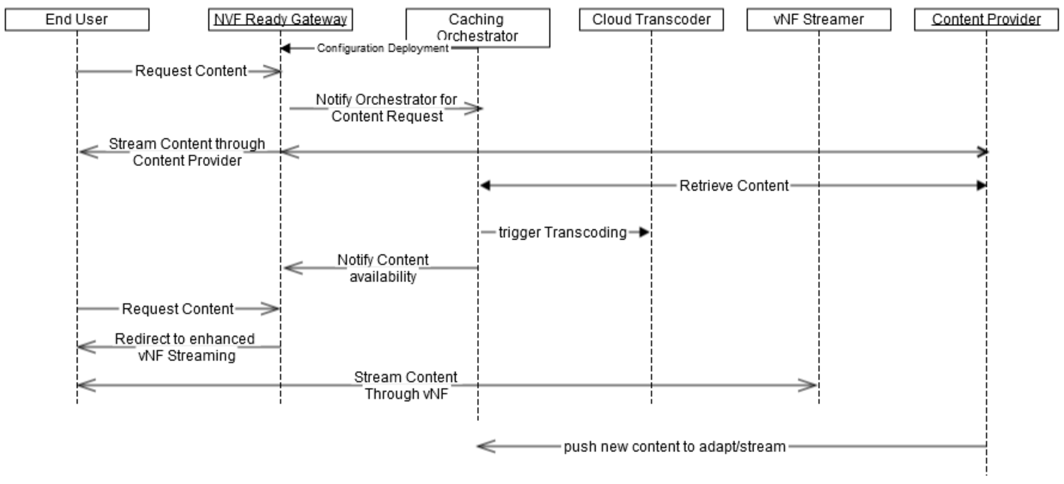
End User
NVF Ready Gateway
Caching Orchestrator
Cloud Transcoder
vNF Streamer
Content Provider
Configuration Deployment
Request Content
Notify Orchestrator for Content Request
Stream Content through Content Provider
Retrieve Content
trigger Transcoding
Notify Content availability
Request Content
Redirect to enhanced vNF Streaming
Stream Content Through vNF
push new content to adapt/stream

Figure 4-21 Sequence diagram for the transcode/stream VNF example

4.5.8. Technology

4.5.8.1. Netty: a Java Non-Blocking Network Framework

Netty is an asynchronous event-driven network application framework 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 to be used to implement the OSGi bundle Proxy.

4.5.8.2. 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 POC.

4.5.8.3. 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 worker 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 of.

4.5.9. Dimensioning and Performances

This chapter identifies the pieces of information related to dimensioning and performance.

Processing Power: The vHG system's resources that tend to become the bottleneck are the processor and the processing power of the virtualization node. [Sigcomm] reports that the key factor of performance in virtual residential gateways is able to deliver **throughput** at the lowest cost of resource usage as possible.

CPU Utilization refers to the the time spent to complete a function and the amount of CPU which is used during this interval by the VNF.

Memory Consumption can be defined as the amount of physical memory consumed by the VNF

High Availability is concerned with the vHG function failure and the consequences associated with these failures. This needs to be considered and specified including the system's response when the function fails, the failure recovery time and the function downtime for upgrade. This will be measured by the probability that the function will be operational when required.

Uptime refers to the level of the success provided by the service. This is measured based on the time that the service has been consistently running for a certain period of time.

Scalability is concerned with the ability of the network function to increase the number of successful operations completed over a given period of time.

Provisioning Time is the time required for the service to become operational.

Access Control is concerned with the system's ability to resist unauthorized usage, while providing users with access to the service such as authenticating and authorizing a user or encrypting data.

Network outage loss of virtual network connectivity will directly impact the service latency, quality and availability experienced by the end user.

Since the proposed vHG is related to video streaming, it could be seen as a media server for the end-user. Video streaming will be used to evaluate the performance of the vHG network function based on encoding and decoding also as QoS and QoE offered to the end user [Mei].

Peak Signal-to-Noise Ratio (PSNR): The objective peak signal-to-noise ratio calculation is the best known evaluation metric. It represents the fidelity of an image i.e. by comparing the image to its original form.

Video Quality Metric (VQM) is a standardized method of measuring video quality by making a comparison between the original and the distorted video sequences based on a set of features extracted from each video.

VQM takes the original video and the processed video and computes the quality based on:

- **Calibration:** the sampled video is calibrated in preparation for feature extraction. It estimates and corrects the spatial and temporal shift as well as the contrast and brightness offset of the processed video sequence compared to the original video sequence.
- **Quality Features Extraction**: This consists of extracting the set of quality features characterizing perceptual changes in the spatial, temporal and chrominance properties.
- **Quality Parameters Calculation** computes a set of quality parameters which describe perceptual changes in video quality by comparing features from the processed video with the original video
- **Quality estimation** computes the overall quality metrics using a linear combination of parameters calculated based on the above steps

4.5.10. Future Work

The following present potential features for the vHG that may be necessary to implement in order achieve more objectives on the vHG in the future.

- **Parental Lock** allows users to block access to specific content which not appropriate to users under the age of 18 using a PIN.
- **Configuration Management** includes functions that aim to manage the HG components including the firewall, security settings.
- **QoS management** includes functions that aim to manage the QoS of the HG in order to ensure the reliable delivery of services.
- **Monitoring** includes functions that will detect any issues with the service delivery also as provide statistics regarding the usage.
- **Supporting Media protocols** such for VOD, NPVR, TSTV, OTT clients and provide interfaces to existing content platforms
- **Multi-screen,** support various, simultaneous, screens of varying resolution and formats
- **Encryption**, to support different encryption schemes for cached content e.g. scrambling.
- **Firewall**: this should be deployed to filter and validate all requests.

5. Conclusions and Future Work

5.1. Conclusions

In this deliverable we reported the present status of research, design and implementation we have done in WP5 "Network Functions" work-package of T-NOVA project.

This work-package contributes to the project both in general framework aspects (T5.1, T5.2) and in the actual implementation of a set of VNFs we will use for proving T-NOVA system (T5.3).

It is worth noting that some of the VNFs we are developing in T5.3 will stand as the "Proof-of-Concept" versions of prototypes with limited features focusing on the innovation aspects. The industrial partners in charge of them aim to go further after the project by implementing the real products out of them. This is especially true for the vSBC and vHG.

The research and innovation activities consist of the optimization of VNF execution in a cloud environment provided by acceleration techniques through commodity hardware, in particular DPDK and GPU-based. The objective is to include them in a datacenter deployment scenario similar to a production environment and prove their efficiency.

For the development of T-NOVA system components in this stage we privileged the objective to provide a full functioning version in the shortest time as possible instead of working to the most appealing solution in terms of advanced research. The rationale is to limit delay and risks for the implementation of the entire T-NOVA system. Nonetheless, we realized that enough effort and time is available to work enhancements towards to most advanced research solution by the end of the project.

5.2. Future Work

In each of the sections of this document we detailed the backlog of activities for each topic, that are managed in the tasks composing WP5.

In general, since we designed the architecture and identified the technologies for the first stage of implementation, the effort is now be in the implementation itself. Research activity proceeds in parallel investigating more in depth the available technologies, such as hardware accelerators that can be available in datacenters.

Extensive cross checking activities will be activated to harmonize details about interfaces and information models among all the components of the project. We have created a dedicated section in the project wiki for describing all the information related to interfaces among T-NOVA system components. We plan to export this knowledge in a public web site when it will be stable enough.

Detailed specification of VNF metadata descriptor is key for the actual implementation of T-NOVA system. This descriptor will be aligned to the work in

WP3 regarding VNF Descriptors at Orchestration level as well as ETSI NFV ISG recommendations. Moreover, it will be the result of extensive discussion and research activities of all the technical workpackages (WP3 to WP6, not forgetting WP2) that will be reported in future deliverables. Intermediate results will be reported in the project wiki as per interface descriptions.

6. 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

7. 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

[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.

[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

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

[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.

[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/

[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

[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/

[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/

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

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

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

[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.

[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

[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