ETSI GS NFV-IFA 002 V2.1.1 (2016-03)

Network Functions Virtualisation (NFV); Acceleration Technologies; VNF Interfaces Specification

Disclaimer

The present document has been produced and approved by the Network Functions Virtualisation (NFV) ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.
It does not necessarily represent the views of the entire ETSI membership.

Reference
DGS/NFV-IFA002

Keywords
acceleration, interoperability, NFV, NFVI,
performance, portability

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Contents

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This Group Specification (GS) has been produced by ETSI Industry Specification Group (ISG) Network Functions Virtualisation (NFV).

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document specifies requirements for a set of abstract interfaces enabling a VNF to leverage acceleration services from the infrastructure, regardless of their implementation. The present document also provides an acceleration architectural model to support its deployment model.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

[1] ETSI GS NFV 003: "Network Functions Virtualisation (NFV); Terminology for main concepts in NFV".

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ETSI GS NFV-INF 003: "Network Functions Virtualisation (NFV); Infrastructure; Compute Domain".

[i.2] ETSI GS NFV-SWA 001: "Network Functions Virtualisation (NFV); Virtual Network Functions Architecture".

[i.3] ETSI GS NFV-IFA 003: "Network Functions Virtualisation (NFV); Acceleration Technologies; vSwitch Benchmarking and Acceleration Specification".

[i.4] ETSI GS NFV-IFA 004: "Network Functions Virtualisation (NFV); Acceleration Technologies; Management aspects Specification".

[i.5] NVM Express™ Inc: NVM Express 1.0e, NVM Express 1.1, NVM Express 1.2.

NOTE: Available at http://www.nvmexpress.org/specifications/.

[i.6] ETSI GS NFV-INF 005: "Network Functions Virtualisation (NFV);Infrastructure; Network Domain".

[i.7] ETSI GS NFV-IFA 001: "Network Functions Virtualisation (NFV); Acceleration Technologies; Report on Acceleration Technologies & Use Cases".

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the terms and definitions given in ETSI GS NFV 003 [1] and the following apply:

abstract interface: computer specification and modelling construct. It defines an information model and a way to communicate between two or more entities

NOTE: Computing objects and APIs can be developed for a programming language to implement it.

Application Programming Interface (API): computer programming language construct, composed of a set of functions, methods, objects, structures and constant definitions used to implement an abstract interface

NOTE: This construct is called an abstract interface language binding. There might be multiple bindings to a single abstract interface, one for each computer language (C, C++, Java™, Ruby, PHP, etc.).

dispatching: deterministic method of distributing packets amongst packet queues to be handled by the network stack, exposed as a NIC capability

NOTE: This dispatching can be done by an operating system or a software library such as DPDK or ODP. The criteria for dispatching can be as simple as round robin or as complex as packet hashing on combination of IP address, TCP ports, taking into account tunnelling. The number of processor threads handling the queues can be lower or higher than the number of those queues.

extensible para-virtualised device: para-virtualised device that can execute code and use resources provided by the hypervisor domain at runtime

NOTE: The benefit of the extensibility is to avoid crossing virtualisation boundary whenever it is possible. The resources enable bypassing the hypervisor in a hardware independent manner.

Execution Environment (EE): set of resources and control mechanisms on which application are running

NOTE 1: Examples of Execution Environments include:

- Traditional Operating System.
- RUMP kernels.
- Java™ Virtual Machine (with or without underlying Operating System).
- Xen Minios.
- DPDK.
- Docker.

NOTE 2: An Execution Environment may be virtualised on top of a hypervisor or not.

NOTE 3: Execution Environments may share or not resources such as processor between applications running on top of them.

language binding: API definition for an abstract interface on a particular programming language

library: collection of implementations of behaviour, written in terms of a language that has a well-defined interface by which the behaviour is invoked

NOTE: This means that as long as a higher level program uses a library to make system calls, it does not need to be re-written to implement those system calls over and over again. In addition, the behaviour is provided for reuse by multiple independent programs. A program invokes the library-provided behaviour via a mechanism of the language. For example, in a simple imperative language such as C, the behaviour in a library is invoked by using C's normal function-call. What distinguishes the call as being to a library, versus being to another function in the same program, is the way that the code is organized in the system.

load balancing: dynamic application level traffic distribution function, that distributes traffic amongst VNFC instances within a VNF or amongst VNF instances

NOTE: As defined in ETSI GS NFV-SWA 001 [i.2], clause 5.1.4.

offload: delegate processing (e.g. classification, forwarding, dispatching, load balancing, cryptography, and transcoding) to a different processor or other specialized hardware entity

Real Time Application (RTA): application whose execution can be **guaranteed** to be accomplished under a specific "execution contract"

NOTE: For instance within a bounded delay, for a certain bandwidth. It assumes execution on a Real Time Execution Environment or Operating System (see Real Time Execution Environment).

Real Time Execution Environment (RTEE): Execution Environment that offer applications with provisions to meet their execution contract (see Real Time Application)

Real Time Operating System (RTOS): Real Time Execution Environment in the form of an Operating System

NOTE: An RTOS has an advanced algorithm for scheduling. Scheduler flexibility enables a wider, computer-system orchestration of process priorities, but a real-time OS is more frequently dedicated to a narrow set of applications. Key factors in a real-time OS are minimal interrupt latency and minimal thread switching latency; a real-time OS is valued more for how quickly or how predictably it can respond than for the amount of work it can perform in a given period of time.

software framework: abstraction in which software providing generic functionality can be selectively changed by additional user-written code, thus providing application-specific software

NOTE 1: A software framework is a universal, reusable software environment that provides particular functionality as part of a larger software platform to facilitate development of software applications, products and solutions. Software frameworks may include support programs, compilers, code libraries, tool sets, and application programming interfaces (APIs) that bring together all the different components to enable development of a project or solution.

NOTE 2: Frameworks contain key distinguishing features that separate them from normal libraries:

- Inversion of control: In a framework, unlike in libraries or normal user applications, the overall program's flow of control is not dictated by the caller, but by the framework.
- Default behaviour: A framework has a default behaviour. This default behaviour is some useful behaviour and not a series of no-ops.
- Extensibility: A framework can be extended by the user usually by selective overriding or specialized by user code to provide specific functionality.
- Non-modifiable framework code: The framework code, in general, is not supposed to be modified, while accepting user-implemented extensions. In other words, users can extend the framework, but should not modify its code.

3.2 Abbreviations

For the purposes of the present document, the abbreviations given in ETSI GS NFV 003 [1] and the following apply.

API	Application Programming Interface
Encrypt/Decrypt	Encryption and Decryption
Encap/Decap	Encapsulation and Decapsulation
EPD	Extensible Para-virtualised Device
IRQ	Interrupt ReQuest
NAT	Network Address Translation
NUMA	Non Uniform Memory Access
NIC	Network Interface Card
RAM	Random Access Memory
SA	Security Association
SPD	Security Policy Database
UML	Unified Modelling Language

VA	Virtual Accelerator
VNI	VXLAN Network Identifier
VTEP	VXLAN Tunnel End Point
VXLAN	Virtual eXtensible Local Area Network

4 Overview

4.1 Problem Statement

4.1.1 VNF Acceleration goals

The goals of the present document are:

- to identify common design patterns that enable an executable VNFC to leverage, at runtime, accelerators to meet their performance objectives;
- to describe how a VNF Provider might leverage those accelerators in an implementation independent way; and
- to define methods in which all aspects of the VNF (VNFC, VNFD, etc.) could be made independent from accelerator implementations.

VNF providers have to mitigate two goals:

- VNFs might have constraints to perform their function within certain power consumption boundaries, CPU core count, PCI express slot usage and with good price/performance ratio; and
- VNFs should accommodate most if not all deployment possibilities.

Use of accelerators can help meet the constraints but can have an influence on deployment flexibility. VNF acceleration implementations will range from inflexible software that is tightly-coupled to specific software and hardware in the VNF and NFVI (see pass-through model as shown in figure 4.1.1-1), to highly flexible loosely-coupled software that uses common abstractions and interfaces (see abstracted model as shown in figure 4.1.1-2). Further it is understood that virtualisation and acceleration technologies will evolve. Pass-through deployments are expected to exist, and the present document does not intend to preclude any specific acceleration architectures from NFV deployments. However, the focus of the present document is to define and promote abstracted models.

It is desirable that the use of accelerators be implementation independent. There is a slight difference between "implementation independent executable VNFC" and "implementation independent VNF":

- An implementation independent executable VNFC is a software that can leverage a known set of accelerator implementations both in hardware and in software. The part of the VNFD that applies to this VNFC contains information elements that allow the NFV management and orchestration to find a compute node with the requested characteristics or hardware. Should a new hardware become available on the market, a VNF Provider willing to make use of it to accelerate a VNFC has to update it's the VNFC image and the VNFD, the operator has to on-board the new VNF package and redeploy it to make use of the new hardware. This was defined as the pass-through model in ETSI GS NFV-INF 003 [i.1], clause 7.2.2.

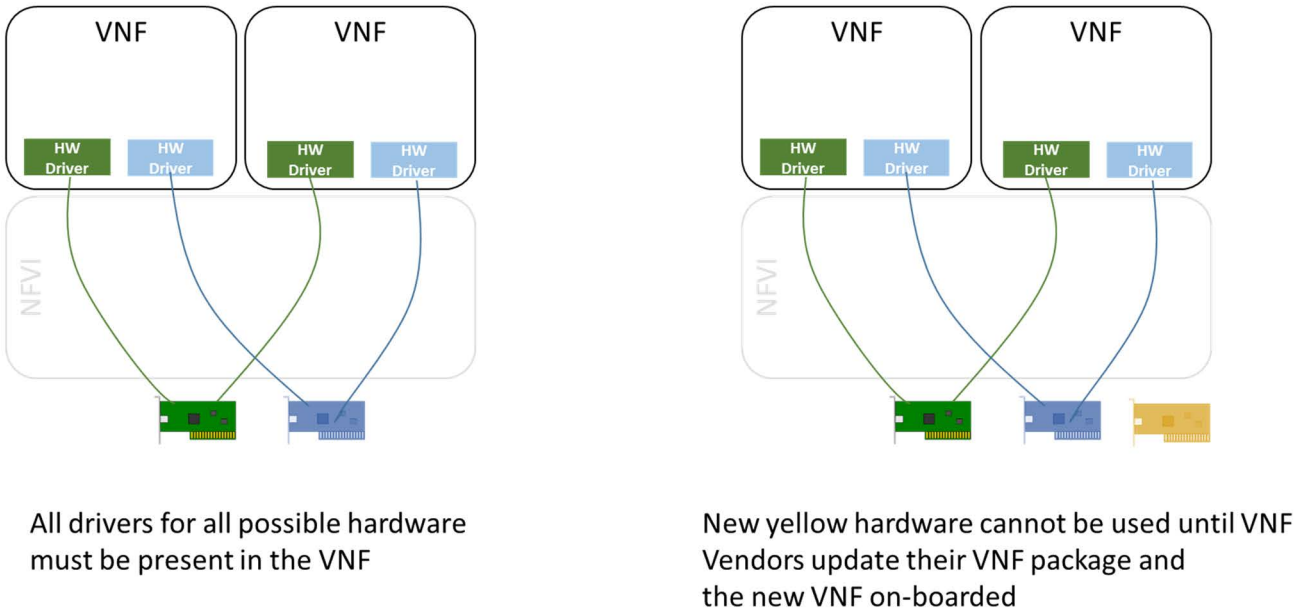


Figure 4.1.1-1: Pass-through model

- An implementation independent VNF is a VNF that makes no assumption whatsoever on the underlying NFVI. Its VNFD does not contain any accelerator specific information elements. Should a new hardware become available on the market, the operator will update its NFVI to allow the VNF to make use of the new hardware. An implementation independent VNF is thus based on implementation independent VNF software that makes use of a functional abstraction of an accelerator supported by an adaptation layer in the NFVI. This model is close to the abstracted model defined in ETSI GS NFV-INF 003 [i.1], clause 7.2.2.

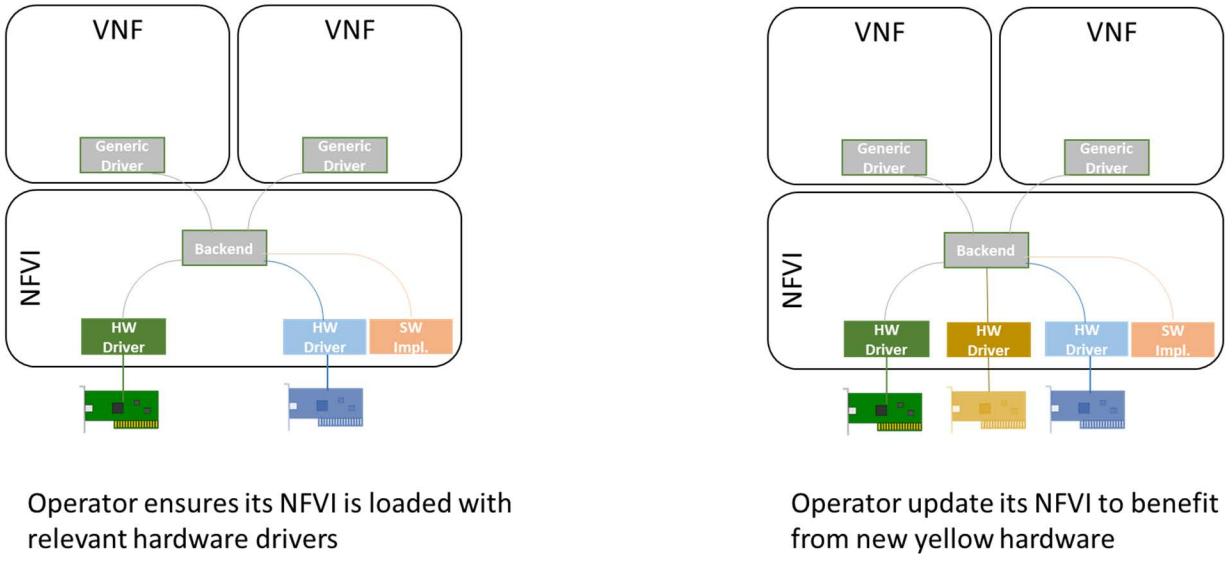


Figure 4.1.1-2: Abstracted model

NOTE 1: An implementation independent executable VNFC allows for VNF deployment in both hypervisor based and non-hypervisor based environments. The latter configuration is outside the scope of the present document.

Live migration of such hardware independent accelerated VNF may be possible if any associated acceleration state information required can also be migrated.

NOTE 2: Live migration from a compute node with accelerator to a compute node without accelerator or with a different accelerator is allowed (in particular to cope with emergency response situations).

A further refinement of the aforementioned goals is to make sure that VNFs can leverage those accelerators regardless of:

- the diversity of VNF software execution environments:
  - Operating Systems (Windows®, Linux®, etc.);
  - Java™ Virtual Machine;
  - RUMP Kernels;
  - DPDK, ODP;
- the diversity of software frameworks or libraries for acceleration, each having provisions to leverage software or hardware acceleration technologies; and
- the diversity of virtualisation environments such as KVM, Xen, VMWare ESX or Microsoft® Hyper-V.

NOTE 3: The present document focuses on acceleration of the VNF execution, it does not cover how a VNF influences packet forwarding in the NFV,which is covered by ETSI GS NFV-IFA 003 [i.3] or by ETSI GS NFV-IFA 004 [i.4]. Requirements for the packet dispatching control interface are however specified by the present document.

NOTE 4: "Microsoft®, Windows®, Linux® and Java™ are examples of a suitable products available commercially. This information is given for the convenience of users of the present document and does not constitute an endorsement by ETSI of these products".

The accelerators considered span several different execution aspects:

- network stack acceleration: Packet dispatching, manycasting, partial networking stack offloads (L4, IPSec, etc.);
- network payload acceleration: compression, transcoding, pattern matching;
- cryptography (encryption, hashing, random number generation, etc.);
- storage;
- digital Signal Processing; and
- algorithmic Acceleration.

4.1.2 Network related acceleration

The network related accelerations have to take into account the network overlays involved. While many NICs offer TCP accelerations (checksum calculation and verification, TCP Segmentation Offload, etc.) some cannot operate when the traffic is in overlay networks (either layer 2 such as VxLAN or layer 3). Furthermore, when the traffic itself is tunnelled, for instance GTP for a GGSN node, TCP accelerations on such traffic is to be considered as partial networking stack offload.

4.1.3 Storage related acceleration

There is local storage on the compute node and remote storage on other NFVI nodes. The acceleration considers how both types of storage are accessed from the VNF.

4.1.4 Algorithmic acceleration

Algorithmic acceleration can be used by VNFs and NFVI within an NFV environment in order to achieve a better performance and more deterministic behaviour when executing algorithmically complicated functions.

4.2 Software architecture

4.2.1 Overview

The industry already successfully implemented an abstraction layer for networking and storage in the form of virtio-net and virtio-block that works across virtualisation environments and across software libraries:

- A VNF leveraging virtio-net ports and virtio-block disk controllers can be live migrated between servers running different virtualisation environments.
- A VNF does not have to be updated as new networking or disk hardware is available on the market.
- Microsoft® Windows®, Apple® OS/X, Linux®, DPDK, ODP, Netmap can make use of virtio-net for accessing network packets.

NOTE : "Apple® is an example of a suitable product available commercially. This information is given for the convenience of users of the present document and does not constitute an endorsement by ETSI of this product."

The present document extends this model to specify an acceleration architectural model for NFV.

The NFV architectural framework can include a Virtual Accelerator in addition to Virtual Compute, Virtual Storage and Virtual Network that can abstract the underlying proprietary vendor-specific hardware offload accelerators and provide a virtual instance that the VNFs can use. The Virtual Accelerator shall provide a standardized vendor agnostic accelerator interface that VNFs can use to access the underlying accelerator.

4.2.2 Acceleration model

4.2.2.1 General

The acceleration model, as depicted in figure 4.2.2.1-1, is based on Extensible Paravirtualised Devices (EPD) implements one or more instances of Virtual Accelerators and is used by applications through an EPD driver running in the VNF's Execution Environment (EE) that exposes the abstract interface of a VA.

An EPDrelies on collaboration with an EPD backend executing in the Execution Environment of the NFVI (i.e. hypervisor domain) to benefit from the acceleration.

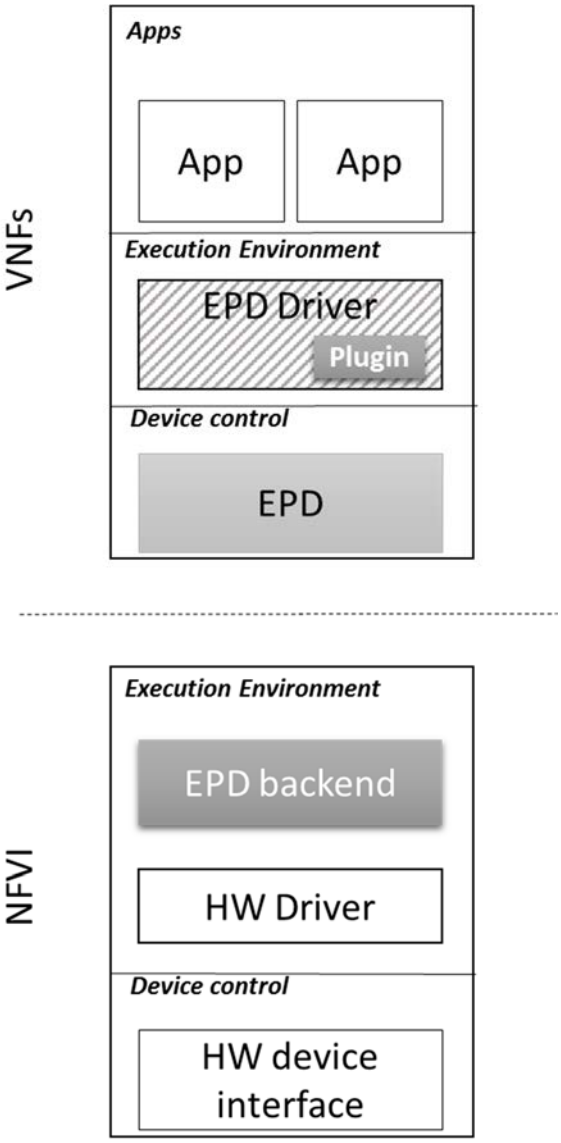


Figure 4.2.2.1-1: Acceleration model

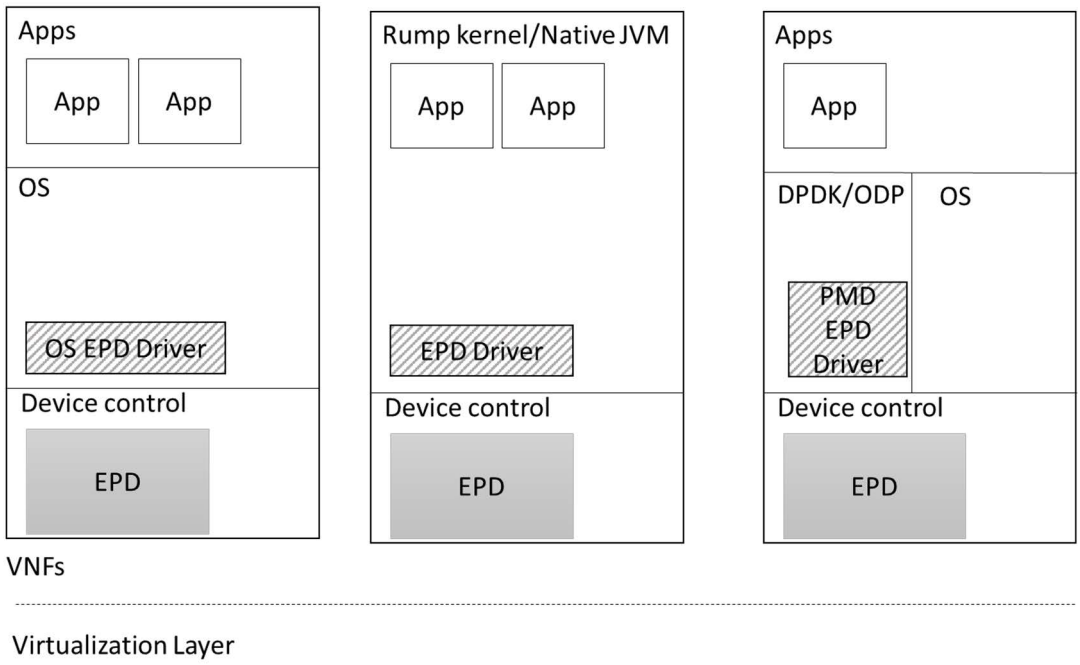
The acceleration model allows a single HW device to expose multiple VAand multiple instances of VA per VNF.

The backend may provide EPD drivers with plugins to allow intra-VNF acceleration that avoid virtualisation barrier crossing. A VNF may - but need not - use this plugin depending on security or administrative policies.

4.2.2.2 VNF aspects

For each specified Virtual Accelerator, the VNFCs to be accelerated shall include an EPD driver that provides the an abstract interface to the EPD.

The VNFC's Execution Environment implements EPD drivers for each EPD it supports and makes it available to applications through an acceleration API or framework (see figure 4.2.2.2-1).



NOTE: The "Device Control" is an interface that is used to communicate/control devices, real or virtual hardware (PCI configuration is one example of such device control interface). The "Device Control" of a VM is created and maintained the hypervisor. The "Device Control" of a server is made available to software by the processor. What happens when data is read or written to this space depends on many aspects, it can be trapped by the hypervisor itself or QEMU or other software.

Figure 4.2.2.2-1: Acceleration model - VNF part

4.2.2.3 Virtualisation Layer aspects

A Virtual Accelerator backend is associated with one or more EPSs from one or more types. It implements an adaptation layer to the hardware or to hypervisor domain resources. For instance, in the case of virtual NIC, the EPD backend does not pass the packets to the hardware but rather to a software stack (either host networking stack or other optimized data path).

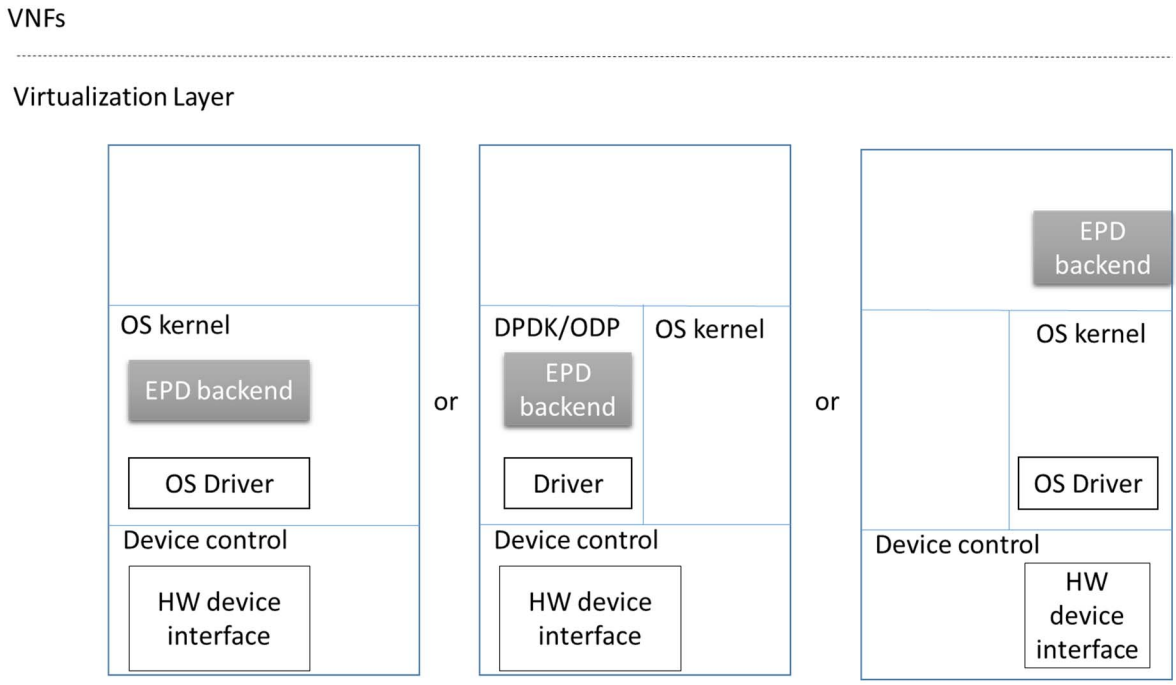


Figure 4.2.2.3-1: Acceleration model - hypervisor part

As illustrated in figure 4.2.2.3-1, the EPD backend may:

- implement acceleration directly in software;
- pass acceleration software plugin to the virtual accelerator driver; or
- drive acceleration hardware regardless of its housing/location.

The acceleration hardware can be collocated on the same compute node or located on a remote node as a network attached accelerator (for instance CloudRAN signal processor pool).

NOTE: The nature of the remote node, a "Physical Network Function" or any other entity, is for further study.

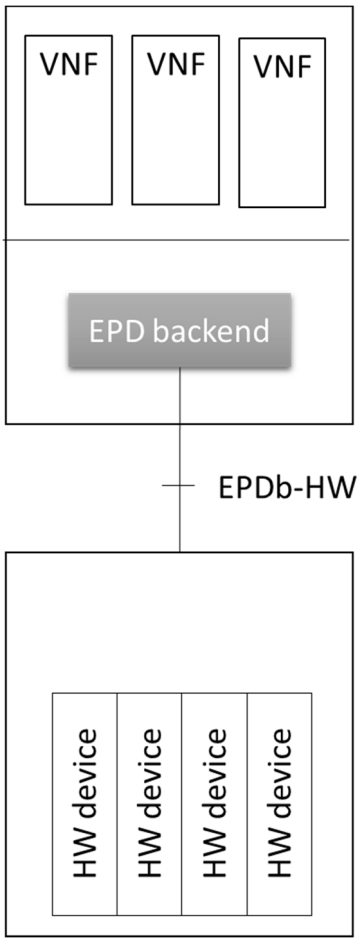


Figure 4.2.2.3-2: Acceleration housing

The EPDb-HW interface between the EPD backend and the actual hardware, shown in figure 4.2.2.3-2, is outside the scope of ETSI NFV standardization.

4.2.2.4 Intra-VNF acceleration

There are two acceleration types that can be performed within a VNF.

The first type is intra-VNFC acceleration and accelerates a VNFCI. When the acceleration is provided by resources directly accessible by the processor (collocated GPU cores, DSP, FPGA), the acceleration model shall allow direct acceleration leverage without crossing the virtualisation boundary.

In compliance to the problem statement, the EPD driver shall not include any processor specific acceleration, but rather the Virtualisation Layer shall provide the EPD at runtime with relevant software plugin so that it can leverage the acceleration.

The VNF can define security or administrative policy regarding the usage of such software plugin.

NOTE 1: The model is quite similar to Linux® VDSO model.

The second type is inter-VNFC acceleration. It can modify how the traffic is routed or forwarded between VNFCIs and may include performing additional acceleration on packet forwarding on the infrastructure network resources. This is described in ETSI GS NFV-INF 005 [i.6] as Dynamic Optimization of Packet Flow Routing (DOPFR).

The acceleration is performed by the infrastructure network resources under the control of an infrastructure network control function. For example, the infrastructure Network Controller provides logical switch constructs, one per VNFCI, for its exclusive use to offload packet processing. The logical switch may share underlying infrastructure network resources with other logical switches. The infrastructure network controller is responsible for partitioning the infrastructure network resources into logical switches. The operations performed on the logical switch by the VNFCI are translated into requests on the underlying infrastructure network resources by the infrastructure network control function.

NOTE 2: The logical switch Lifecycle Management is described in ETSI GS NFV-IFA 004 [i.4].

NOTE 3: The infrastructure network control function may but need not be in the Network Controller.

Figure 4.2.2.4-1 shows an example VNF with inter-VNFC acceleration where the red path shows the non-accelerated packet flow and the blue path shows the accelerated packet flow.

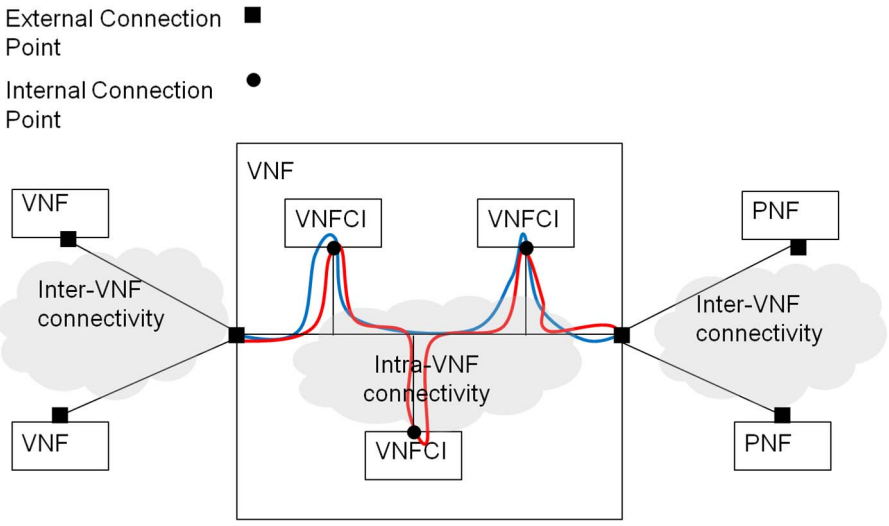


Figure 4.2.2.4-1: Inter-VNFC acceleration

NOTE 4: The acceleration boundaries consist of the VNF external connection points.

Figure 4.2.2.4-2 shows an example logical switch representation for figure 4.2.2.4-1.

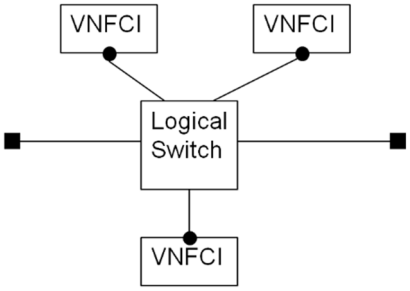


Figure 4.2.2.4-2: Logical switch representation for figure 4.2.2.4-1

Figure 4.2.2.4-3 shows another example VNF with inter-VNFC acceleration where the red path shows the non-accelerated packet flow and the blue path shows the accelerated packet flow. Note that the need to modify the intra-VNF connectivity in order to provide acceleration in the infrastructure network requires the external endpoints to have virtual links to the internal connection points of a VNFCI.

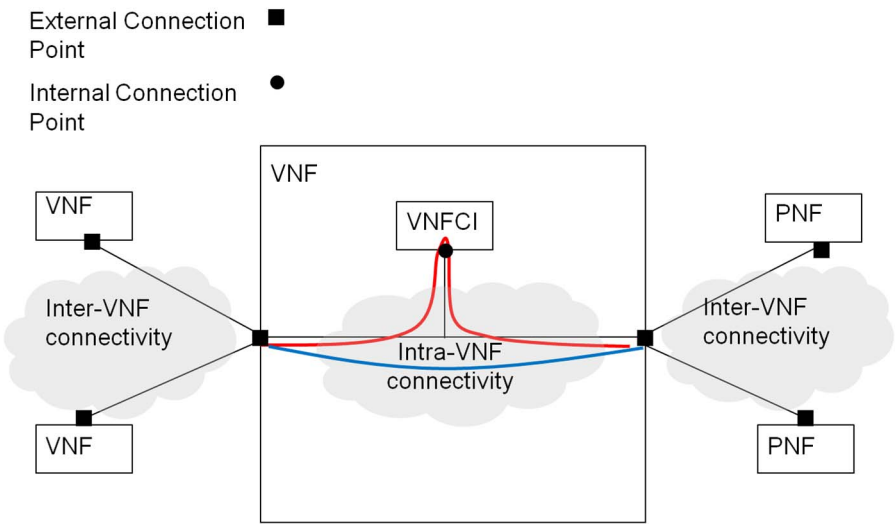


Figure 4.2.2.4-3: Inter-VNFC acceleration with single VNFCI

Figure 4.2.2.4-4 shows an example logical switch representation for figure 4.2.2.4-3.

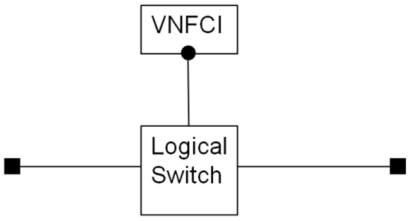


Figure 4.2.2.4-4: Logical switch representation for figure 4.2.2.4-3

5 Abstract Interface functional requirements

5.1 Overview

Virtual Accelerator is made visible and used by the EPD Driver over the Vn-Nf reference point. In order to achieve this visibility, control and usage technologies such as virtio can be used.

ETSI GS NFV-IFA 001 [i.7] lists a number of use cases that define many different accelerators covering very diverse functional domains such as cryptography and video transcoding.

An EPD Driver is integrated in VNF specific Execution Environments, so their structure and north bound interfaces or APIs are out of scope of the present document. The EPD Driver shall still implement a defined behaviour and state machine when leveraging any accelerator.

Clause 5.2 covers generic requirements for all accelerators and clause 5.1 covers requirements applicable to the EPD drivers. Subsequent clauses define functional requirements for each selected accelerator from ETSI GS NFV-IFA 001 [i.7] use cases. All interface requirements specified from clause 5.4 onwards apply to both the northbound interfaces of an EPD driver (i.e. between the EPD driver and applications) and the southbound interface of an EPD (i.e. between the EPD and the EPD backend).

5.2 Common Acceleration Virtualisation interface requirements

Table 5.2-1

Numbering	Functional requirements description
VnNf.Accel.001	Acceleration Virtualisation interface shall support exchanging data (packets, frames, etc.) in both directions.
VnNf.Accel.002	Acceleration Virtualisation interface shall support multiple channels of communication. Channels are independent communication instances across the Acceleration Virtualisation interface. Examples of channels are virtual functions, DMA queues, ports and sockets.
VnNf.Accel.003	Acceleration Virtualisation interface shall support control usage of channels (e.g. priority, number of packets per second).
VnNf.Accel.004	Acceleration Virtualisation interface shall support scaling of VNF/VNFC.
VnNf.Accel.005	Acceleration Virtualisation interface shall support notification of multiple events for the same EPD.
VnNf.Accel.006	Acceleration Virtualisation interface shall support reading capabilities and capacities of accelerator.
VnNf.Accel.007	Acceleration Virtualisation interface shall support sending and reading configuration of accelerator.
VnNf.Accel.008	Acceleration Virtualisation interface shall support obtaining EPD plugin.
VnNf.Accel.009	Acceleration Virtualisation interface shall support state transfer (e.g. for migration) in both direction.
VnNf.Accel.010	Acceleration Virtualisation interface shall support reading statistical information of accelerator.
VnNf.Accel.012	Acceleration Virtualisation interface shall support notifying drivers with new version of EPD plugin.
VnNf.Accel.013	Acceleration Virtualisation interface shall support notifying drivers with live migration events.
VnNf.Accel.014	Acceleration Virtualisation interface shall support device lifecycle management.
VnNf.Accel.015	Acceleration Virtualisation interface shall support unidirectional and bidirectional channels.
VnNf.Accel.016	Acceleration Virtualisation interface shall support synchronous and asynchronous channels.
VnNf.Accel.017	Acceleration Virtualisation interface shall support QoS.
VnNf.Accel.018	Acceleration Virtualisation interface shall support one or more context.

5.3 EPD Driver requirements

Table 5.3-1

Numbering	Functional requirements description
VnNf.Drv.001	EPD Driver shall check version compatibility with EPD backend.
VnNf.Drv.002	EPD Driver shall check existence through the Acceleration Virtualisation Interface of plugins to be executed in the VNF context.
VnNf.Drv.003	EPD Driver shall consume (that is remove from notification queue) and optionally respond or react to Acceleration Virtualisation Interface notifications.
VnNf.Drv.004	EPD Driver shall adapt to scaling requests.
NOTE: The originator of the scaling requests to EPD Driver is not identified at the time of writing and is for further study.	

5.4 Cryptography functional group

5.4.1 Overall requirements

Table 5.4.1-1

Numbering	Functional requirements description
Crypto.001	Crypto interface shall support acceleration of defined minimum set of IPSec related cryptography operations.
Crypto.002	Crypto interface shall support acceleration of defined minimum set of SSL related cryptography operations.
Crypto.003	Crypto interface shall support acceleration of defined minimum set of TLS 1.0 and above related cryptography operations.
Crypto.004	Crypto interface shall support acceleration of defined minimum set of HTTP2.0 related cryptography operations.
Crypto.005	Crypto interface shall support acceleration of defined minimum set of SRTP related cryptography operations.
Crypto.006	Crypto interface shall support acceleration of defined minimum set of SRTCP related cryptography operations.
Crypto.007	Crypto interface shall support acceleration of cryptographic primitive operations, e.g. key generation of x bits length, obtain random number in range x…y with distribution z (Gaussian, flat, etc.), obtain prime number with bit length x.

5.4.2 Operations requirements

Table 5.4.2-1

Numbering	Functional requirements description
CryptoOps.001	Crypto interface shall support symmetric cryptography operations for a defined, minimal set of symmetric cryptographic functions, e.g. Feistel based functions.
CryptoOps.002	Crypto interface shall support asymmetric cryptography operations for a defined, minimal set of asymmetric cryptographic functions.
CryptoOps.003	Crypto interface shall support random number generation operations.
CryptoOps.003.1	Data/Metadata about the random number generation should be available, e.g. its verification/trust, provenance and properties of seed values, performance characteristics (i.e. rate of random number generation).
CryptoOps.004	Crypto interface shall support obtaining the list of available crypto algorithms.
CryptoOps.005	Crypto interface shall support asynchronous operations.
CryptoOps.007	Crypto interface shall support synchronous operations.
CryptoOps.008	Crypto interface shall support cryptography operations whose context is remotely stored in the accelerator (stateful acceleration, or more precisely functional offload).
CryptoOps.009	Crypto interface shall support cryptography operations whose context is locally stored in the VNF (stateless acceleration).
CryptoOps.010	Crypto interface shall support obtaining the cryptographic key.

5.4.3 Crypto interface requirements

Table 5.4.3-1

Numbering	Functional requirements description
VnNf.crypto.001	Crypto interface shall support cryptography session lifecycle management (initialization, configuration, reconfiguration, termination, etc.).
VnNf.crypto.002	Crypto interface shall support notification of cryptographic operations completion
VnNf.crypto.003	Crypto interface shall support obtaining the list of available hashing encryption and key generation algorithms and random number generators.
VnNf.crypto.003.1	Crypto interface shall support obtaining list of available operations provided by the accelerator: these shall include at least the hashing, encryption and key generation and random number generators.
VnNf.crypto.004	Crypto interface shall support querying the progress of ciphering, hashing, random number generation, key generation etc. operations.
VnNf.crypto.005	Crypto interface shall support handling multi-part buffers (for instance for IP fragments or SSL blocks).
VnNf.crypto.006	Crypto interface shall support notification of cryptographic events such as IPSec re-keying requests.
VnNf.crypto.007	Crypto interface shall support notification of cryptographic alarms such as IPSec stop because of SA (security association) hard volume overflow without re-keying.
VnNF.crypto.008	Crypto interface shall support obtaining the list of available cipher algorithms and modes.

5.4.4 Crypto driver requirements

Table 5.4.4-1

Numbering	Functional requirements description
CryptDrv.001	Crypto driver shall support execution of virtualisation supplied plugin for intra-VNF acceleration (based on instruction or CPU collocated resources such as GPU and FPGA).

5.4.5 Management and monitoring requirements

Table 5.4.5-1

Numbering	Functional requirements description
CryptMgmt.001	Crypto interface shall support retrieving and resetting cryptography statistics.
CryptMgmt.002	Crypto interface shall support notification of threshold crossing.
CryptMgmt.003	Crypto interface shall support initializing defined minimum set of security context and parameters (e.g. SA, public/private keys, certificates).
CryptMgmt.004	Crypto interface shall support retrieving possible thresholds.
CryptMgmt.005	Crypto interface shall support retrieving a set of supported parameters.
CryptMgmt.006	Crypto interface shall support the irrecoverable wiping of all stored and/or provisioned information in the cryptographic accelerator (i.e. a complete, proper, unrecoverable wipe) - either in a given context or the unit as a whole.
CryptMgmt.007	Crypto accelerator interface should support trusted computing capabilities, e.g. to verify firmware of the hardware accelerator; and other trust/signing measurement capabilities, e.g. Trusted Platform Module.

5.5 IPSec offloading functional group

5.5.1 Overview

IPSec uses cryptographic security services to protect communications over IP network. IPSec offloading aims to offload the packets security processing to the accelerator who knows how to do that from the Security Association (SA) between the sender and receiver. The SA is established by the negotiation of cryptography keys which can be accelerated via related cryptography acceleration.

5.5.2 IPSec offloading interface requirements

Table 5.5.2-1

Numbering	Functional requirements description
VnNF.IPSec.001	IPSec offloading interface shall support reception of SA information.
VnNF.IPSec.002	IPSec offloading interface shall support obtaining traffic forwarding policy, such as flow-based forwarding, subnet-based forwarding.
VnNF.IPSec.003	IPSec offloading interface shall support notification of unmatched traffic against the forwarding policy.

5.5.3 Operations requirements

Table 5.5.3-1

Numbering	Functional requirements description
IPSec.Ops.001	IPSec offloading interface shall support flow classification based on traffic forwarding policy.
IPSec.Ops.002	IPSec offloading interface shall support checking ingress traffic against the SPD.
IPSec.Ops.003	IPSec offloading interface shall support checking ingress traffic whether maps to an existing SA.
IPSec.Ops.004	IPSec offloading interface shall support notification of creating a new SA if no SA is mapped for ingress traffic.
IPSec.Ops.005	IPSec offloading interface shall support Encrypt/Decrypt operations with mapped SA.
IPSec.Ops.006	IPSec offloading interface shall support authentication operations with mapped SA.
IPSec.Ops.007	IPSec offloading interface shall support Encap/Decap operations with given SA.

5.5.4 Management and monitoring requirements

Table 5.5.4-1

Numbering	Functional requirements description
IPSec.Mgmt.001	IPSec offloading interface shall support reporting active flows statistics.
IPSec.Mgmt.002	IPSec offloading interface shall support reporting inactive flows statistics.

5.6 TCP offloading functional group

5.6.1 TCP offloading interface requirements

Table 5.6.1-1

Numbering	Functional requirements description
TCP.001	TCP offloading interface shall support enabling and disabling Partial TCP offload.
TCP.002	TCP offloading interface shall support passing TCP connection states to ToE after TCP connection is setup in case Partial TCP offload is enabled.
TCP.003	TCP offloading interface shall support releasing TCP connection states to host CPU after TCP connection is closed in case Partial TCP offload is enabled.
TCP.004	TCP offloading interface shall support releasing TCP connection states to host CPU in case Partial TCP offload is disabled.
TCP.005	TCP offloading interface shall support notifying ToE which TCP connection will be offloaded.
TCP.006	TCP offloading interface shall support notifying ToE which TCP connection will be retrieved.

5.6.2 TCP offloading type requirements

Table 5.6.2-1

Numbering	Functional requirements description
VnNF.TCP.001	TCP offloading interface shall support TCP checksum offload.
VnNF.TCP.002	TCP offloading interface shall support large segment offload.
VnNF.TCP.003	TCP offloading interface shall support large receive offload.
VnNF.TCP.004	TCP offloading interface shall support TCP acknowledgement offload.

5.7 Storage functional group

5.7.1 NVMe Over Fabric

5.7.1.1 Overview

NVM Express 1.0/1.1/1.2 [i.5] is defined assuming that PCI Express is the transport. NVMe over Fabrics enables NVMe to be used with transports where the communication between the host and controller is through messages rather than a shared memory access model as in PCI Express. The definition is flexible to support a wide range of transports.

5.7.1.2 Interface requirements

Table 5.7.1.2-1

Numbering	Functional requirements description
NOF.Ops.001	NVMe over Fabric interface shall support a minimum set of NVMe over Fabric command.

5.8 Re-programmable computing functional group

5.8.1 Re-programmabe interface requirements

Table 5.8.1-1

Numbering	Functional requirements description
VnNf.ReProg.001	Re-programmable interface shall support obtaining the capabilities of the platform (Memory, DMA, etc.).
VnNf.ReProg.002	Re-programmable interface shall support passing a compilation image.
VnNf.ReProg.003	Re-programmable interface shall support multiple computing functions.
VnNf.ReProg.004	Re-programmable interface shall support obtaining the capabilities of a particular function.
VnNf.ReProg.005	Re-programmable interface shall support memory creation in target, memory freeing, and memory copying.
VnNf.ReProg.006	Re-programmable interface should support coherent memory.
VnNf.ReProg.007	Re-programmable interface shall support transferring data to particular memory location.
VnNf.ReProg.008	Re-programmable interface shall support transferring data in a streaming mode fashion to a particular function.

5.8.2 Operations requirements

Table 5.8.2-1

Numbering	Functional requirements description
ReProgOps.001	Re-programmable interface shall support obtaining the state of a particular function.
ReProgOps.002	Re-programmable interface shall support starting/stopping a particular function.
ReProgOps.003	Re-programmable interface shall support retrieving completion events from a particular function.
ReProgOps.004	Re-programmable interface shall support retrieving various notification types from a particular function.
ReProgOps.005	Re-programmable interface shall support changing the state of available functions.

5.8.3 Management and monitoring requirements

Table 5.8.3-1

Numbering	Functional requirements description
ReProgMgmt.001	Re-Programmable interface shall support initializing and configuring platforms.
ReProgMgmt.002	Re-Programmable interface shall support retrieving and resetting platform statistics.
ReProgMgmt.003	Re-Programmable interface shall support initializing and configuring particular function.
ReProgMgmt.004	Re-Programmable interface shall support retrieving and resetting particular function statistics.
ReProgMgmt.005	Re-Programmable interface shall support debugging a particular function. (step by step debug).

5.9 Dynamic Optimization of Packet Flow Routing Functional Group

5.9.1 DOPFR interface requirements

Table 5.9.1-1

Numbering	Functional requirements description
VnNF.DOPFR.001	VNF DOPFR interface shall support discovering if the infrastructure network has the optimization capability.
VnNF.DOPFR.002	VNF DOPFR interface shall support instructing the infrastructure network which packet flow will be optimized.
VnNF. DOPFR.002	VNF DOPFR interface shall support instructing the infrastructure network how the packet flow is forwarded, e.g. via forwarding graph.
VnNF. DOPFR.003	VNF DOPFR interface shall support instructing the infrastructure network which packet flow will be routed back to VNF.

5.9.2 Management and monitoring requirements

Table 5.9.2-1

Numbering	Functional requirements description
DOPFR.Mgmt.001	VNF DOPFR interface shall support report of the optimized flow statistics to VNF.

5.10 NAT offloading functional group

5.10.1 Overview

The following requirements apply to both the use of NAT within the NFVI as part of the vSwitch, and the customer specific traffic separated from the provider traffic.

5.10.2 Overall requirements

Table 5.10.2-1

Numbering	Functional requirements description
NAT.001	NAT offloading function shall support acceleration of network address translation of source IP address, destination IP address, L4 source port number, L4 destination port number.

5.10.3 NAT offloading interface requirements

Table 5.10.3-1

Numbering	Functional requirements description
NATif.001	NAT offloading interface shall support obtaining of NAT policy, such as static mapping and dynamic mapping.
NATif.002	NAT offloading interface shall support reception of NAT method, such as full-cone NAT, restricted-cone NAT, symmetric NAT.
NATif.003	NAT offloading interface shall support reception of translation information for forward and reverse translation flows.
NATif.004	NAT offloading interface shall support obtaining the state of NAT session.

5.10.4 NAT offloading Operations requirements

Table 5.10.4-1

Numbering	Functional requirements description
NATops.001	NAT offloading interface shall support notification of the change of the state of NAT session.

5.10.5 Management and monitoring requirements

Table 5.10.5-1

Numbering	Functional requirements description
NATmgmt.001	NAT offloading interface shall support retrieving and resetting NAT statistics.
NATmgmt.002	NAT offloading interface shall support initializing defined minimum set of NAT session.

5.11 VXLAN offloading functional group

5.11.1 Overview

The following requirements apply to both the use of VXLAN within the NFVI as part of the vSwitch, and the customer specific traffic separated from the provider traffic.

5.11.2 Overall requirements

Table 5.11.2-1

Numbering	Functional requirements description
VXLAN.001	VXLAN offloading interface shall support acceleration of packet encapsulation and decapsulation with VXLAN header by MAC-in-UDP.

5.11.3 VXLAN offloading interface requirements

Table 5.11.3-1

Numbering	Functional requirements description
VXLANif.001	VXLAN offloading interface shall support reception of Encap/Decap information between source VTEP and destination VTEP, such as VLAN,VNI,IP address of VTEP.
VXLANif.002	VXLAN offloading interface shall support reception of mapping information between VLAN ID and VNI.
VXLANif.003	VXLAN offloading interface shall support obtaining traffic forwarding policy, such as IP-based forwarding, MAC-based forwarding.

5.11.4 VXLAN offloading operations requirements

Table 5.11.4-1

Numbering	Functional requirements description
VxLANops.001	VXLAN offloading interface shall support decapsulation operations of VLAN and encapsulation operations with VXLAN header by MAC-in-UDP.
VXLANops.002	VXLAN offloading interface shall support decapsulation operations with VXLAN header by MAC-in-UDP and encapsulation operations with VLAN.
VXLANops.003	VXLAN offloading interface shall maintain the mappings of VLAN IDs to VNIs.

5.11.5 Management and monitoring requirements

Table 5.11.5-1

Numbering	Functional requirements description
VXLANmgmt.001	VXLAN offloading interface shall support retrieving and resetting statistics of VXLAN offloading information.

5.12 Media functional group

5.12.1 Overview

Multimedia communication services are widely used, which mainly are video services and audio services. Users use a variety of terminals such as TV sets, computers, smart phones to access the communication services. For the interoperability between terminals and requirements of services, VNFs should provide various media-processing capabilities, such as media transcoding, mixing etc. This is because the amount of media is much greater, the requirements for real-time processing and latency are more stringent, and media accelerators are needed.

In addition to the transcoding operation which is described in ETSI GS NFV-IFA 001 [i.7], clause 5.1.5, there are also several basic media acceleration operations:

- Video and audio mixing in the conference service.
- Video encoding and decoding in multimedia service.

5.12.2 Media overall requirements

Table 5.12.2-1

Numbering	Functional requirements description
Media.001	Media interface shall support acceleration of defined minimum set of video related operations.
Media.002	Media interface shall support acceleration of defined minimum set of audio related operations.

5.12.3 Media operations requirements

Table 5.12.3-1

Numbering	Functional requirements description
MediaOps.001	Media interface shall support video transcoding (e.g. codec, video resolutions, resizing, horizontal stretching) operations.
MediaOps.002	Media interface shall support audio transcoding (e.g. codec) operations.
MediaOps.003	Media interface shall support video decoding operations.
MediaOps.004	Media interface shall support video encoding operations.
MediaOps.005	Media interface shall support video mixing operations (e.g. composite multiple video sources with effects into a single canvas).
MediaOps.006	Media interface shall support audio mixing operations.

5.12.4 Media interface requirements

Table 5.12.4-1

Numbering	Functional requirements description
VnNf.Media.001	Media interface shall support synchronous operations.
VnNf.Media.002	Media interface shall support asynchronous operations.
VnNf.Media.003	Media interface shall support notification of media (audio and video) operations completion.
VnNf.Media.004	Media interface shall support querying the progress of media (audio and video) operations.
VnNf.Media.005	Media interface shall support acceleration of concurrent media (audio and video) operations.
VnNf.Media.006	Media interface shall support obtaining the list of available media formats.

5.12.5 Management and monitoring requirements

Table 5.12.5-1

Numbering	Functional requirements description
MediaMgmt.001	Media interface shall support obtaining the utilization efficiency of media acceleration resources, e.g. usage of the media acceleration resources for audio transcoding, etc.
MediaMgmt.002	Media interface shall support obtaining the statistics of media operations (e.g. the coexisting number of video transcoding).
MediaMgmt.003	Media interface shall support obtaining the capacity provided by media acceleration resources.

Annex A (informative): Authors & contributors

The following people have contributed to the present document:

Rapporteur:

- François-Frédéric Ozog, 6WIND

Other contributors:

- Bruno Chatras, Orange
- Rob Diamond, ARM
- Bob Monkman, ARM
- Brian Skerry, Intel
- Jinwei Xia, Huawei
- Andrew Thurber, Cisco
- Rabi Abdel, Altera
- AiJuan Feng, Huawei
- JianXing Hou, Huawei
- Qian Wang, China Telecom
- Yunpeng Xie, China Telecom
- Dan Daly, Intel
- Jan Ignatius, Nokia Networks
- Michaek Rooke, Nokia
- Allen Chen, Altera

Annex B (informative): Bibliography

- ETSI GS NFV-INF 004: "Network Functions Virtualisation (NFV); Infrastructure; Hypervisor Domain".
- Microsoft®: Network Driver Interface Specification (NDIS) version 6.0; IPSec Offload Version 2.

NOTE: Available at https://msdn.microsoft.com/library/windows/hardware/ff556996.aspx.

- ETSI GS NFV-INF 001: "Network Functions Virtualisation (NFV); Infrastructure Overview".

Annex C (informative): Change History

Date	Version	Information about changes
November 2014	0.0.1	Scope and Table of Contents
February 2015	0.0.2	Added Clause 3 Contributions: 00038r3
February 2015	0.0.3	Revised Scope, Added clause 4.1 Problem Statement Contributions: 00039r3
February 2015	0.1.0	Added clause 4.2.1 Overview Contributions: 000312
May 2015	0.2.0	Added clauses: 4.2.2 Acceleration model, Clause 5 overview and more sub clauses, Clause 6 & 7 for crypto acceleration. ETSI drafting rules and styles alignment on all clauses Contributions: 000314r4, 000484r2, 000485r2, 000415r3
May 2015	0.2.1	Corrected reference, added definitions & abbreviations, IPSec offload, scope clarification, communication channel clarification, acceleration interface requirements Contributions: NFVIFA(15)000597r2, NFVIFA(15)000630r2, NFVIFA(15)000709r1, NFVIFA(15)000715r2, NFVIFA(15)000719r3
July 2015	0.2.2	Added generic crypto and SRTP, SRTCP related requirements; DOPFR Contributions: NFVIFA(15)000817r1, NFVIFA(15)000818, NFVIFA(15)000823r2
August 2015	0.2.3	Added NAT (NFVIFA(15)000908r3) and VxLAN offloads (NFVIFA(15)000909r2); crypto modifications (NFVIFA(15)000955r1); updates related to "synthetic device" rename to EPD (NFVIFA(15)000968)
October 2015	0.2.4	Cryptography enhancements (NFVIFA(15)0001137r5), added Media acceleration (NFVIFA(15)0001210r3, TCP Offload (NFVIFA(15)0001275); added DOPFR (NFVIFA(15)0001297r2), GPU Computing renamed Re-Programmable Computing and added content (NFVIFA(15)0001368r2)
December 2015	0.3.0	Scope change (NFVIFA(15)0001440) and editorial corrections (NFVIFA(15)0001439r3). Storage NFVIFA(15)0001310r2. Removed all editor's notes
January 2016	0.3.1	Editorial corrections NFVIFA(15)0001680, NFVIFA(15)0001681, NFVIFA(15)0001682.

History

Document history		
V2.1.1	March 2016	Publication