Network Functions Virtualization: The Portability Challenge

Bruno Chatras and François Frédéric Ozog

Abstract

NFV is a paradigm shift in the telecom industry, where the functions traditionally hosted in purpose-built equipment become virtualized and run on pools of standard IT servers. NFV creates a twofold challenge for the telecom and IT industries: virtual appliances must deliver high performance while being portable across commodity servers. This article proposes an approach to address both requirements at the same time, relying on the notion of synthetic devices abstracting hardware-dependent acceleration functions.

Building on a well-established trend in the IT industry, virtualization is rapidly entering the telecom market, network functions virtualization (NFV) being its latest incarnation [1]. Leveraging cloud computing technologies, NFV is the subject of much interest and may become a reality in telecom operators' networks quicker than expected two years ago. Currently, most telecom equipment is sold in the form of vertically integrated systems with applications running on, and tightly coupled with, purpose-built middleware and hardware. NFV is a paradigm shift from the current situation to a cloud model where telecom applications (a.k.a. network functions) are virtualized and run on pools of commodity servers. The NFV infrastructure is assumed to consist of pools of industry standards servers, often referred to as common off-the-shelf (COTS) servers. Although there is no formal definition of what a COTS server is, it is generally accepted that these are servers commonly deployed in the IT industry, equipped with general-purpose processors such as X.86, Power8, and ARM, as opposed to application-Specific integrated circuits (ASICs). In data centers, this approach is already widely applied to many types of applications other than those hosted in telecom operators' networks. What makes NFV unique are the telecom industry's demanding requirements for five-nine availability and for predictable end-to-end real-time performance.

Obtaining high performance from COTS servers has been an area of creativity and innovation in the IT industry for several years. Many acceleration solutions have been defined and successfully implemented on bare metal environments, tightly coupled with application software. NFV brings additional challenges. On one hand, as application software is expected to be independent from hardware platforms, it is not acceptable that this software has to incorporate hardware-specific libraries to leverage acceleration solutions. On the other hand, the presence of an abstraction layer (i.e., the hypervisor) between the applications and hardware resources prevents the applications from accessing all the capabilities of the acceleration solutions available on bare metal. This article proposes an approach to respond to both challenges at the same time, leveraging paravirtualization technologies, extended with the concept of synthetic devices abstracting hardware-dependent acceleration functions and limiting whenever possible virtualization barrier crossing.

The remainder of the article is organized as follows.The next section provides a short summary of the European Telecommunications Standards Institute (ETSI) NFV architectural framework. Then we introduce the performance requirements associated with the virtualization of network functions, and we review the main solutions supported by the industry. Following that we discuss the portability challenge created by such solutions. Finally, we analyze several options toward the specification and standardization of an abstraction layer to enable meeting both challenges at the same time.

NFV Architectural Framework

ETSI defines an NFV architectural framework (Fig. 1) enabling virtualized network functions (VNFs) to be deployed and executed on an NFV infrastructure (NFVI), which consists of COTS hardware resources (computing, storage, and network) wrapped with a software layer that abstracts and logically partitions them [2]. In hypervisor-based deployments, a VNF is typically mapped to one virtual machine (VM) in the NFVI but may also be split into multiple VNF components (VNFCs) loaded on separate VMs (e.g., with different scaling requirements). The deployment, execution, and operation of VNFs on an NFVI are steered by a management and orchestration (M&O) system, the behavior of which is driven by a set of metadata (a.k.a. NFV descriptors) describing the characteristics of the network services and their constituent VNFs. The M&O system [3] includes an NFV orchestrator (NFVO) in charge of the life cycle of network services, a set of VNF managers in charge of the life cycle of the VNFs (including VNF scaling out/in), and a virtualized infrastructure manager (VIM), which can be viewed as an extended cloud management system responsible for allocating and releasing NFVI resources upon request of the VNFM and NFVO.

Bruno Chatras is with Orange Labs.

Francois Frédéric Ozog is with 6WIND.

0890-8044/16/$25.00 © 2016 IEEE

Performance Challenges and Well-Known Solutions

NFV Acceleration

Obtaining high performance (high throughput, fast processing, short transit delays, etc.) from COTS servers has been an area of creativity and innovation in the IT industry for several years. Many acceleration solutions have been defined and successfully implemented in non-virtualized environments.

Acceleration solutions are implemented in hardware, software, or both. Hardware acceleration typically relies on a co-processor or other specialized hardware entity (hardware accelerator) deployed to offload processing from the main processor or to bring additional resources. Software acceleration is about designing application software to make the most efficient use of computing and other resources.

In the NFV framework, acceleration is either embedded in the VNF in the form of software or exposed to the VNFs by the NFVI. Acceleration is of paramount importance when considering virtualization of network functions operating in the data plane that have to forward a huge number of packets with minimum delay (e.g., security gateways, gateways in mobile packet core networks, broadband access servers) and/or perform CPU-intensive processing (video transcoding, encryption, etc.).

Software Acceleration

Data plane acceleration programming environments such as the Data Plane Development Kit (DPDK), initially launched for Intel processors and now an open source project (dpdk.org), or Open Data Plane (ODP) from Linaro for ARM processors are examples of VNF-embedded accelerations. The verified strategic assumption of DPDK and ODP is that the operating system networking stack is too slow, and the network hardware must be driven directly from the application. They both enable applications to bypass this stack and retrieve raw data from the network interface card (NIC) through a poll mode driver rather than interrupt-based kernel drivers. With the poll mode, the system's central processing unit (CPU) can periodically check for the arrival of incoming network packets without being interrupted, and use the freed-up processing cycles for performing other tasks.

Hardware Acceleration

Well-known hardware acceleration solutions that can be provided by the NFVI to the VNFs include, but are not limited to, using persistent memory resources (a new high-performance storage paradigm leveraging memory slots rather than disks), access to additional computing resources (massive scale co-processing), network I/O acceleration on the NIC, generic cryptographic acceleration (e.g., in relation to encrypted storage), and compression/decompression acceleration (e.g., in support of deep packet inspection).

Network I/O acceleration on the NIC can take many different forms. An example is the use of advanced queue managers to enable multi-core load balancing so that selected processor cores directly receive a fraction of the traffic (e.g., receive side scaling [RSS] load balancing). Transmission Control Protocol (TCP) or cryptographic processing offload is another example: TCP offload capable NICs send to the host processor only reassembled data rather than each packet associated with a TCP connection. This greatly reduces the peripheral component interconnect (PCI) bus overhead influence but comes at the price of a limited number of concurrent connections. Another approach consists of using remote direct memory access (RDMA) technology to enable fast communications between applications by enabling the NIC to transfer data directly to or from application memory, without involving the CPU.

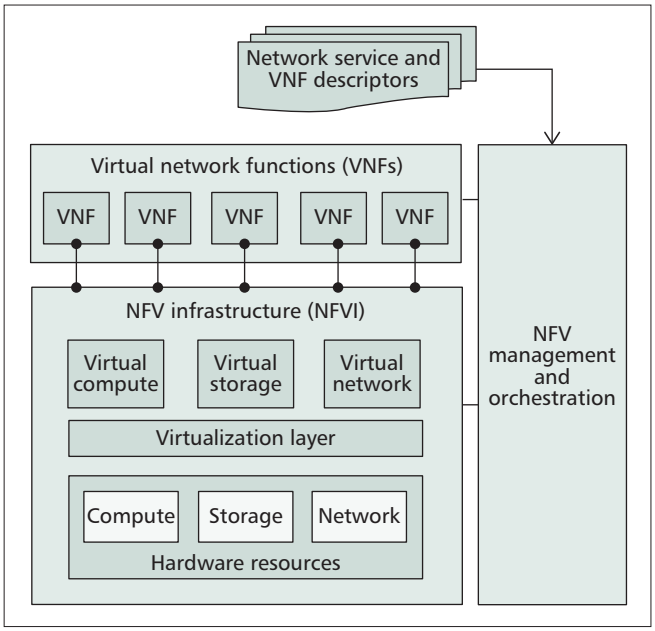

Figure 1. NFV architecture framework.

In addition, a fraction of NFV network traffic workloads is made of specific protocols such as Diameter over the Stream Control Transmission Protocol (SCTP) and/or requires tunnelling support such as general packet radio service (GPRS) Tunneling Protocol (GTP) and multiprotocol label switching (MPLS). Those workloads defeat RSS load balancing techniques that have been designed for typical enterprise workloads and thus require new versions of hardware accelerators in the form of extended packet to core load balancing mechanisms hosted on the NICs.

Portability Challenge

Software Dependencies

Use of hardware acceleration techniques by applications can make them hardware-dependent. This is not a major concern in non-virtualized environments because solutions are validated against a specific hardware platform, but becomes a real issue in an NFV framework. Indeed, the NFV framework is expected to provide the capability to load, execute, and move VNFs across different servers in multivendor environments, a capability known as *portability*.

In virtualized environments, the hypervisor is expected to abstract hardware resources including hardware accelerators. However, VMs on hypervisors present to the guest OS a set of virtualized devices that today do not offer all acceleration mechanisms available on real hardware (e.g., TCP offload on network hardware) and lack many virtualized versions of acceleration devices such as cryptographic accelerators. As a result, solutions aimed at exposing the hardware directly into VMs have been defined.

A well-known solution is the single root I/O virtualization (SR-IOV) technology [4]. SR-IOV allows VMs, once created by the hypervisor, to share a piece of hardware without involving the hypervisor directly for all activities. For example, a physical NIC can expose a number of virtual functions that can be "attached" to VMs. The VMs will see each attached virtual function as if it was a physical card (same brand, same model as the real physical card). In other words, SR-IOV allows the creation of multiple "shadow" cards of a physical network card. Each shadow card has its own medium access control (MAC) address. Other solutions such as

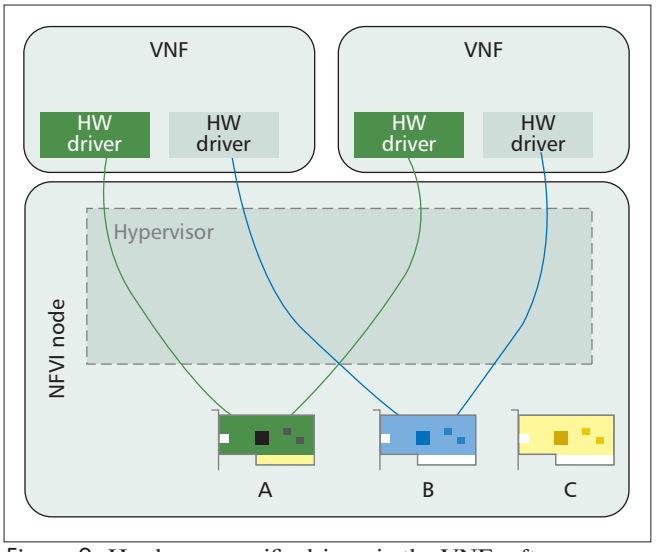

Figure 2. Hardware-specific drivers in the VNF software.

multi-root I/O virtualization (MR-IOV) or PCI pass-through are used to bypass the hypervisor and give direct access to the network hardware. Despite limitations in the number of virtual functions that can be created, and hence the number of VNFs that can use the real hardware, the aforementioned solutions allow VNFs to exhibit the same performance as their bare metal counterparts.

By leveraging direct hardware access techniques, performance objectives are met, but VNF instantiation and (live) migration become more complex or even impossible. Furthermore, as illustrated in Fig. 2, this approach requires hardware-specific drivers to be embedded in the VNF software, which implies that a VNF cannot leverage a new hardware accelerator available in an NFVI node (e.g., accelerator C in Fig. 2) until the corresponding driver has been integrated in the VNF software.

NFV Management and Orchestration Aspects

In an NFV framework, decisions on where to instantiate a VNF are made by the M&O system, based on the contents of VNF descriptors (VNFDs) and information found in the NFVI resource catalog. A VNFD is a template that describes the attributes and requirements necessary to realize a VNF. It is filled by the VNF provider and packaged with the actual VNF code shipped to the operator. Ideally, a VNFD should contain hardware/technology-independent requirements to guarantee the flexible deployment and portability of VNF instances on multi-vendor and diverse NFVI environments, for example, with diverse computing resource generations and diverse virtualization technologies However, the version of the VNFD template published by ETSI at the end of 2014 was designed to enable VNF providers to specify acceleration requrements expressed in an implementation-dependent manner to accommodate the direct hardware access approach. Examples of such requirements are an indication that the VNF software has been developed and optimized for a specified CPU model and instruction extension set; an indication that the VNF has been developed and optimized with a polled mode driver infrastructure applicable to a specified NIC type; or an indication that the VNF requires that an SR-IOV virtual function from a specified PCI express (PCIe) device be allocated to the VM.

Obviously, this approach defeats the portability goal and key benefits of NFV. VNF providers need to develop multiple versions of the same VNF, each targeting a different type of server. Network operators need to resort to stopgap measures such as using a homogeneous server pool or onboarding multiple images for the same VNFC, each targeting a different type of server. Furthermore, once instantiated, a VNF leveraging SR-IOV cannot be migrated to another server.

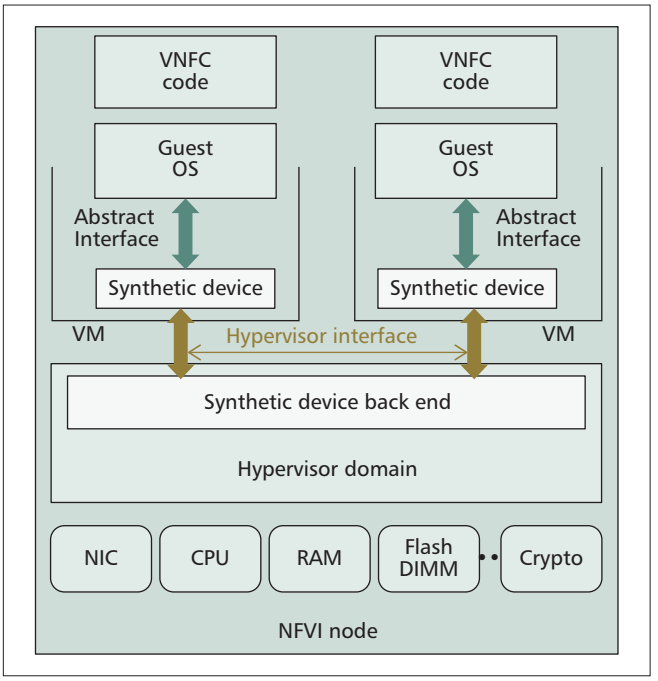

Figure 3. Abstraction layer in an NFVI node.

The challenge we are facing is thus: is it possible, and if so, how, to achieve high performance by leveraging hardware accelerators and at the same time have hardware-independent VNFs?

Toward an Abstraction Layer

The Synthetic Device Concept

The ETSI specification of the NFV infrastructure compute domain advocates the introduction in future versions of NFV specifications of a hardware acceleration abstraction layer and of a common abstract application programming interface (API) enabling the VNFC code to access acceleration services provided by the infrastructure [5].

In the IT industry, DirectX and OpenGL for graphics cards, and OpenCL for massively parallel computing are successful examples where it has been possible to define a de facto standard abstract interface for accessing very complex, ever evolving hardware objects.

The industry has also been successful at defining synthetic network interfaces such as virtio net [6, 7] that do not emulate any hardware but define a standard abstract interface to packet I/O. Those devices are used by the applications running in the VMs and rely on the hypervisor to actually implement the interface to hardware accelerators.

It is thus tempting to build on those successes to achieve the initial NFV portability goal without loss of performance. This would require providing VMs with synthetic devices for any type of hardware accelerator and integrating those synthetic devices with frameworks such as DPDK, OpenCL, and others (Fig. 3).

From Concept to Implementation

To achieve this, several steps would have to be realized for each type of network accelerator.

The first step is to define an abstract interface which specifies the possible actions that a VNF can perform and the associated data model. For an abstract interface dealing with network packets, this translates, for example,

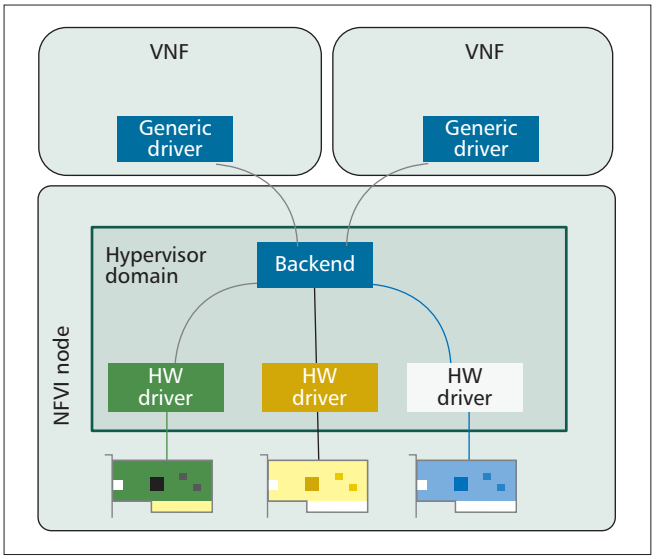

Figure 4. Hardware-specific drivers in NFVI.

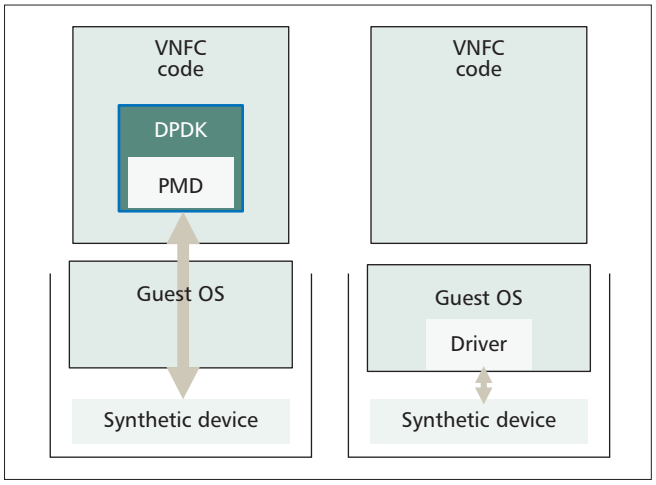

Figure 5. Drivers for synthetic devices.

into simplistic "AreTherePackets," "GiveMePackets," and "SendPackets" primitives; for cryptography this could be primitives such as "LoadSecurityAssociation," "EncryptDataAtOffset," and so on. Of course, many other primitives are required, and a rich data model has to be specified. Defining such an interface involves selecting a subset of known features and methods currently supported by commercially available hardware acceleration solutions, which in itself is a technical challenge.

The second step is to create a synthetic device that implements the abstract interface via diverse techniques such as PCI configuration space object, I/O registers emulation, physical address space ranges, message passing, and interrupt triggering.

The third step is to create a synthetic device "backend" that will support many instances of a synthetic device and execute in the context of the hypervisor, its supporting operating system (if any), or even another VM (e.g., if SR-IOV or any hypervisor bypass mechanism is used simultaneously). Its role is to map each synthetic guest device to the underlying hardware object and control its usage (e.g., ensure that wrong behavior of a VNF has no impact on other VNFs). The backend may in turn instantiate a virtual device in the context of the hypervisor or the host operating system. For instance, this is required when a VNF NIC synthetic device needs a companion NIC virtual device in the hypervisor domain to be integrated with a virtual networking component. Libvirt [8] is an open source framework that can be instrumental in implementing the synthetic device, its backend, and the required communications. Moreover, this framework also allows creating hypervisor-independent synthetic devices [9]. For network operators, a major benefit of this approach is that hardware-specific drivers reside in the NFV infrastructure rather than in the VNFs. As shown in Fig. 4, this enables them to install new or upgraded hardware accelerators without any impact on these VNFs.

The fourth step is to integrate synthetic devices in existing software frameworks such as DPDK and ODP by providing appropriate drivers (e.g., poll mode drivers for DPDK; Fig. 5). These frameworks may have to integrate new acceleration APIs or have to adapt existing APIs to cover new functionalities. For instance, as there is currently no compress/decompress API in DPDK, it may be necessary to include one to leverage related hardware acceleration features exposed as a synthetic device.

A further refinement of this abstraction layer is to allow intra-VNF acceleration when the acceleration leverages special processor instruction sets. The instruction set can be dependent on the processor generation, or result from programming on-die or closely connected field-programmable gate array (FPGA) capabilities. Intra-VNF acceleration will rely on the synthetic device driver executing some form of software plugin that implements the abstract interface leveraging the processor-specific instruction set. From a technical standpoint, positioning the catalog of all available plugins in the hypervisor domain or in the guest is very similar. However, in the context of NFV, one has to remember that the VNF software is usually supplied to a network operator by a third party VNF provider, and the operator is not expected to modify it, as opposed to the hypervisor domain that remains under its control. This is why the proposed abstraction layer positions the catalog of plugins in the hypervisor domain, relying on the synthetic device backend to supply the plugin to the synthetic device driver. Those plugins are conceptually similar to Linux Kernel virtual dynamically linked shared objects (VDSOs), which are kernel routines supplied to user applications as a replacement for some system calls (e.g., gettimeofday) to avoid crossing the user/kernel boundary.

Integration in the NFV Architectural Framework

The synthetic device approach does not require any fundamental change to the NFV architectural framework. However, it modifies the nature of the information that VNF vendors will have to provide in the VNFD and the decision logic that the VIM will execute to make decisions about the placement of VNF components on the servers of the NFVI. Indeed, rather than including references to specific hardware accelerators in the VNFD, VNF providers will list the abstract interfaces (i.e., acceleration capabilities) they need (crypto, video transcoding, etc.) along with key performance indicators (12 Gb/s, 10,000 tunnels, etc.), and the VIM will match this information with compute nodes inventory information.

Standardization and Experiments

Further development of this concept is taking place within the framework of the second specification phase of the ETSI NFV Industry Specification Group. Indeed, processing acceleration and VNF portability are among the major topics on the ETSI NFV Phase 2 agenda. One of the items in the Phase 2 work programme aims at specifying a set of abstract interfaces enabling a VNF to leverage acceleration services from the infrastructure, regardless of their implementation. Beyond the specification work, testing and experiments will be needed to validate this concept. This is one of the subjects being considered within the framework of the OPNFV project, an open source project hosted by the Linux Foundation, focused on developing an integrated open platform for NFV.

Conclusion

This article has reviewed the twofold challenge that NFV has created for the telecom and IT industries: virtual appliances must deliver high performance while being portable across commodity servers. The concept of a synthetic device we introduce in this article provides a promising approach to reconciling the need for processing acceleration solutions with VNF portability requirements.

References

[1] "NFV: An Introduction, Benefits, Enablers, Challenges & Call for Action," NFV white paper; http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[2] ETSI GS NFV 002, "Network Functions Virtualisation (NFV); Architectural Framework," http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_nfv002v010201p.pdf

[3] ETSI GS NFV MAN 001, "Network Functions Virtualisation (NFV); Management & Orchestration," http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf

[4] Y. Dong *et al.*, "High Performance Network Virtualization with SR-IOV," *Proc. 2010 IEEE 16th Int'l. Symp. High Performance Computer Architecture*, 2010, pp. 1–10.

[5] ETSI GS NFV INF 003, "Network Functions Virtualisation (NFV); Infrastructure; Compute Domain," http://www.etsi.org/deliver/etsi_gs/NFV-INF/001_099/003/01.01.01_60/gs_nfv-inf003v010101p.pdf

[6] The virtualization API: virtio-net http://wiki.libvirt.org/page/Virtio

[7] G. Motika *et al.*, "Virtio Network Paravirtualization Driver: Implementation and Performance of a De-Facto Standard," *J. Computer Standards & Interfaces*, vol. 34, no. 1, Jan., 2012, pp. 36–47.

[8] Libvirt : The Virtualization API http://libvirt.org/index.html

[9] R. Russel, "Virtio: Towards a De-Facto Standard for Virtual I/O Devices," *ACM SIGOPS Op. Sys. Rev.*, vol. 42, no. 5, July 2008, pp. 95–103; ACM Digital Library: http://dl.acm.org/citation.cfm?id=1400108.

Biographies

BRUNO CHATRAS (bruno.chatras@orange.com) is a senior standardization manager at Orange Labs Networks and a member of the Orange experts' community on Future Networks. He is the Chairman of the ETSI Technical Committee on Network Technologies and a Vice-Chairman of the ETSI ISG NFV. He joined Orange Labs in 1985, where he started his career by developing GSM standards. Since then he has held several management positions and led the Intelligent Networks R&D Unit.

FRANÇOIS-FRÉDÉRIC drives 6WIND business development in the telecom and cloud markets. He is an entrepreneur with 30 years of experience in technical, sales, and marketing positions. He is the Rapporteur of the ETSI NFV work item on VNF Acceleration Interface and is the author of seven patents. He holds a degree in computing science from Universit de Paris VII.