New Kid on the Block: Network Functions Virtualization: From Big Boxes to Carrier Clouds

Leonhard Nobach
TU Darmstadt
lnobach@ps.tu-darmstadt.de

Oliver Hohlfeld
RWTH Aachen University
oliver@comsys.rwth-aachen.de

David Hausheer
TU Darmstadt
hausheer@ps.tu-darmstadt.de

This article is an editorial note submitted to CCR. It has NOT been peer reviewed.
The authors take full responsibility for this article's technical content. Comments can be posted through CCR Online.

ABSTRACT

Network management currently undergoes massive changes towards realizing more flexible management of complex networks. Recent efforts include slicing data plane resources by network (link) virtualization and applying operating system design principles to Software Defined Networking to rethink network management. Driven by network operators, network management principles are currently envisioned to be even further improved by virtualizing network (middlebox) functions. The resulting Network Functions Virtualization (NFV) paradigm abstracts network functions from dedicated hardware to virtual machines running on commodity hardware. This change in the design of carrier networks is inspired by the success of virtualization in the server market. By deploying NFV, network operators envision to achieve benefits similar to the server market and elastic cloud services, e.g., flexible and dynamic service provisioning, increased resource utilization, improved energy efficiency, and ultimately decreased operational costs. Despite these efforts, the ability of NFV to satisfy performance demands is often questioned. Tackling these challenges opens a set of research questions that felt short in the current discussion and are of particular relevance to the SIGCOMM community. In this position paper, we therefore provide an overview on the current state-of-the-art and open research questions.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design; C.2.3 [**Computer-Communication Networks**]: Network Operations

Keywords

Network functions, middleboxes

1. INTRODUCTION

Today's networks not only include forwarding elements but are increasingly comprised of active networking components that monitor or process the traffic. These components are referred to as middleboxes and realize *network functions*. The number of middleboxes deployed in current networks can even be in the same order as the number of forwarding elements [64]. Typical network functions improve network performance (e.g., proxies or TCP optimizers) and network security (e.g., intrusion detection systems or firewalls). These functions are commonly run on dedicated hardware that is specialized to provide a single network functionality. These hardware appliances are deployed at specific sites and are wired in use-case-dependent forwarding graphs (i.e., service chains). As a result, they are difficult to maintain, update, and scale.

Thus, the current realization of network functions in dedicated *hardware appliances* challenges the flexible provisioning of services. For example, the elastic scaling of services according to varying customer demands is challenged by the limited availability of hardware that is installed at certain locations in a carrier network. To satisfy changing demands, the quantity of the available hardware, their location, and the timely procurement and installation process have to be carefully predicted. This prediction, however, introduces a non-negligible risk: First, if the forecast deviates from actual demand, relocation of misplaced, or liquidation of unused hardware is costly. Secondly, the aforementioned relocation or additional procurement needs time and prevents the network from quickly reacting to changing demands.

This problem of flexible service provisioning has been solved in the *server market* by virtualization techniques and the subsequent realization of scalable Infrastructure-as-a-service (IaaS) cloud services. Such services allow tenants of IaaS clouds to rapidly provision computing resources without changes to the server hardware. This includes a dynamic increase or decrease (scale in / out) of the used resources and a better utilization of idle capacity. It therefore offers potential for cost reduction and improved energy efficiency. As a result, the IaaS cloud computing paradigm not only is a technical concept, but has also become a business model (examples include Amazon AWS and private clouds operated with OpenStack).

Initiated by a consortium of Internet Service Providers (ISPs) in 2012 [11], the *Network Functions Virtualization* (NFV) concept aims at applying these solutions to *carrier networks*. Concretely, NFV solves the problem of flexibly provisioning network functions by realizing them in virtualized environments running on commodity hardware platforms. As a result, ISPs envision to achieve benefits similar to advances in the server market, e.g., elastic resource provisioning, improved energy efficiency, and ultimately reduced costs. While the current efforts on NFV are *mainly* driven by ISPs and commercial interests, they still offer a set of research challenges that are slowly being picked up by the research community (see for example the *HotMiddlebox* workshop).

The realization of NFV particularly requires achieving high packet processing performance. It is often questioned if NFV can satisfy these demands, since the additional software complexity challenges obtaining performance figures that are comparable to dedicated hardware. Tackling this challenge opens a set of research questions that are of particular relevance to the SIGCOMM community. In this position paper, we therefore *i)* review current industry and research efforts on NFV and *ii)* outline research challenges arising in NFV with a particular focus on achieving NFV performance (we refer to [49] for an overview on other research challenges). By this, we aim to provide a starting point for re-

search in the emerging area of NFV. We start by summarizing the core principles and goals of NFV as well as ongoing standardization efforts in Section 2. We then complement the standardization perspective by providing an overview of NFV use cases that are currently being addressed in the academic literature in Section 3. Since achieving high packet processing performance is critical but not optimized in current virtualization techniques, we focus on discussing performance-related aspects in Section 4. Lastly, we discuss research challenges to outline a first research agenda on NFV.

2. NETWORK FUNCTIONS VIRTUALIZATION

A network function (NF) is an intermediate traffic processing entity in a network. From a system management perspective, it is a building block of a network that monitors or processes network traffic (see also ETSI's definition [13]). Besides that, there are control-plane NFs, which take over management tasks beyond traffic processing.

The term *Network Functions Virtualization* (NFV) was brought to attention by Internet Service Providers through standardization by the European Telecommunications Standards Institute (ETSI) [11]. It describes a new paradigm in which network functions are envisioned to be abstracted from dedicated hardware appliances (so-called *middleboxes*), and run as virtualized network functions (VNFs) on cheaper general-purpose hardware.

To further save costs, virtualization and IaaS cloud mechanisms can be applied to elastically provision network functions. Thus, one target of NFV is to reduce capital and operational expenses caused by the procurement, installation, and maintenance of specialized networking hardware to implement the functions. Hardware expenses are reduced to the one of a cloud infrastructure, commonly comprised of mentioned general-purpose servers and forwarding equipment. This allows infrastructure services to scale with user demand. Examples include the dynamic provisioning of additional NFV instances in peak hours or the concentration of functions in fewer datacenters in times of under-utilization.

2.1 Standardization Activities

We start by reviewing standardization activities as the first forum for (industry-driven) efforts on developing NFV.

The NFV standardization activity at the ETSI is organized as Industry Specification Group and was initiated in 2012 upon the publication of a first white paper that coined the term NFV [11]. The initiative currently involves more than 200 member companies working in different NFV-related working groups. The overall mission of the ISP and industry-driven initiative is to provide guidelines for an open and interoperable NFV ecosystem. Meanwhile, the ETSI NFV working groups produced a variety of documents discussing NFV-related aspects. These include *i)* use cases and terminology definitions, *ii)* architectural frameworks and virtualization requirements, *iii)* security and reliability aspects, *iv)* infrastructure considerations, *v)* management and orchestration, and *vi)* performance and service quality (see [3] for a full overview). These documents resemble the current single richest body of NFV documents.

Standardization work at the IETF focused on attempts to form working groups (i.e., Virtual Network Function Pools (VNFPOOL) and Network Function Virtualisation Configuration (NFVCON)) and the dissemination of different draft documents in different established working groups starting in 2013. For instance, the VNFPOOL initiative focused on reliability aspects arising when realizing virtualized network functions and published several draft documents that outline use cases and their requirements (see e.g., [71]). Besides these initiatives, several established working groups evaluated the NFV idea and published draft documents (e.g., focusing on IPv6 considerations [22]). Other related efforts concern work on service function chaining within an established working group [10] and works on network abstraction and programmability (see e.g., the SDNRG, FORCES, and I2RS working groups).

Activities at the Internet Research Task Force (IRTF) concern the formation of a Network Function Virtualization Research Group [4] and several drafts. These drafts describe ongoing work on monitoring [37], verifying NFV services [65], unifying carrier and cloud networks [66], software defined infrastructures and scalability [48, 58, 36], policy aspects [28], and resource management [40].

2.2 SDN, Network Virtualization, NFV: What's the Difference?

One natural question concerns the relation of NFV to other recent advances in network management. The most prominent approaches are network virtualization [23] and Software Defined Networking (SDN) [35].

Network virtualization aims at virtualizing network links to allow multiple virtual networks to coexist on a single physical link. This attempt creates a separation of concerns for different roles: *i)* infrastructure providers managing the physical infrastructure and *ii)* service providers that create and manage the overlaying virtual networks [23]. By this, network virtualization introduces new means for sharing and managing network links.

Orthogonal to network virtualization, Software Defined Networking (SDN) rethinks network management by splitting the control and data plane and by applying operating system design principles to the control plane design. These principles include the introduction of control plane layers and open interfaces between these layers. SDN can be used *i)* to realize network virtualization, *ii)* to instantiate and manipulate forwarding graphs (service chains) [17], or *iii)* to realize simple, commonly stateless network functions such as packet filters. The design of SDN, however, focuses on managing the packet forwarding process. Typical network functions (e.g., proxies or TCP optimizers) instead go beyond the functionality of SDN. They often involve stateful manipulations of packets and are implemented in rather complex software systems. Thus, NFV is an orthogonal technology that addresses different needs of network management, i.e., flexible and efficient provisioning of network functions.

3. USE CASES

Network middleboxes can be found in many places: at home, in small businesses, large enterprises, and of course in carrier environments. The functions they implement are versatile, and NFV is aimed to replace them. This section introduces use cases of NFV—traditional and novel network functions—and reveals early as well as elaborate efforts to replace them. By compiling a representative set of NFV use cases that were mainly addressed and evaluated by researchers, we complement standardization documents that summarize the industry perspective (see e.g., [12]).

3.1 Carrier Clouds

Carrier clouds aim to bring the benefits of cloud computing to traditional carrier networks [67]. These benefits include efficient resource usage and resource scalability and elasticity. In contrast to (typically larger) public clouds, carrier clouds are hosted by the carriers themselves and are thus closer to the end user, or are located deeper inside the network. Thus, carriers keep control over their network and are able to control security and privacy aspects.

NFV serves as a key enabler for realizing carrier clouds. Carrier clouds can be used to realize many of the use cases that are discussed in the subsequent sections.

3.2 Fixed Access Networks

Virtual Residential Gateways. Residential gateways interconnect home networks with residential access networks. These gateways have high provisioning and update costs. For example, the deployment of new services, features, or changed configurations often requires the roll-out of firmware updates. Such a roll-out is challenged by the heterogeneous device landscape, in which firmware updates might not always be possible. To address these challenges, it has been proposed to virtualize the gateway and to subsequently replace it by a L2 switch [12, 72, 25]. Replacing the gateway at the users' home by a layer 2 switch offloads network functions (e.g., NAT, firewall, DHCP) to per-user virtual machines running in the carriers' network (e.g., running in a carrier cloud). This offloading reduces the complexity of residential gateways, improves network debugging and flexible provisioning of features and updates. The evaluation of virtual residential gateways shows increased performance, i.e., higher throughput and lower access latencies. It further promises cost reductions of up to 90% in call centers and up to 46% on product returns [72]. However, since the switched architecture extends the users' LAN to the carriers' network, this mechanism has potential security and privacy implications.

This idea is generalized by the concept of **Edge-as-a-Service** [24]. By applying cloud computing concepts to fixed and mobile edge networks, access network components can be shared in a flexible way. An example use case involves sharing radio and WiFi access to distribute network traffic more efficiently.

In light of this development are efforts to virtualize the PPPoE access concentrator (see RFC 2516) which terminates the PPP link at the ISP's side [16]. It is the primary network function of a Broadband Remote Access Server (BRAS) and provides authentication of its customers, billing, logging, and potentially address assignment. Performance evaluation of commodity hardware with 10GE interfaces showed that line-rate performance can be achieved for full-sized packets. Another use case comprises to virtualize VPN endpoints or concentrators. Like PPP/PPPoE, this also involves header pushing/popping, but additionally involves payload processing for encryption or signing.

3.3 Radio Access Networks

Similar to fixed access networks, several works suggest to virtualize components of the radio access and the network core.

Mobile Access. In the radio access, the virtualization of core components can increase flexibility and allow efficient transitions between different network architectures [53]. As an example, several works propose to virtualize the baseband unit as a common network function of 3G and 4G mobile networks [33, 43]. Besides simplified transitions between network technologies, NFV can ease prototyping [52] and resource sharing [51, 73]. In case of the latter, the physical network infrastructure is envisioned to be sliced into several virtual networks, similar to network virtualization. This further simplifies prototyping by operating several network instances in parallel and allows the network to be used by multiple virtual network operators. The latter allows physical radio access resources to be shared more efficiently and reduces the barriers for new (virtual) carriers to enter the market. An example includes the virtualization of LTE eNB air interface [73].

Mobile Core. Beyond the virtualizing radio access components, recent work has also proposed to virtualize components of the network core. One example concerns the virtualization of the Evolved Packet Core in LTE networks [68]. The proposed architecture orchestrates lightweight virtualized mobile core network instances that run in a (carrier) cloud. This concept allows operators to scale their mobile core network with varying traffic demand.

3.4 Network Core

Translation Technologies. To ease IPv4-to-IPv6 transition, Dual-Stack Lite (DS-Lite) reduces the need to operate dual-stack networks. It establishes a light-weight IPv4-over-IPv6 tunnel from the residential gateway to an address family transition router. Thus, only a single IPv6 network needs to be maintained, while IPv4 is tunneled to legacy peering points. DS-Lite functionality is suitable for virtualization, in particular since a relevant Address Family Transition Router implementation is available in software [1]. Similarly, Carrier Grade NAT gateways are suitable for virtualization.

Multimedia Services. The IP Multimedia Subsystem is a key component of all-IP Next Generation Networks. Its architecture consists of a multitude of components that that can be offered as virtualized cloud services to achieve resource scalability [20]. This effort includes virtualizing the Session Border Controller (see RFC 5853) as an Application Layer Gateway (ALG). Among other use cases, it provides accessibility for clients behind NAT gateways and enforces security by detecting and dropping Denial-of-Service (DoS) or other malicious SIP signaling attempts. A high gateway performance is required to handle a large number of simultaneous SIP/RTP sessions. The evaluation of a virtualized gateway implementation [50] shows its general feasibility, but at lower performance than gateways implemented in dedicated hardware.

Load Balancing. Many popular services (e.g. websites) must cope with large amounts of requests. NFV can improve network operation by dynamically instantiating virtual load balancing instances depending on the current network load. Because of the expected load, the realization in NFV is performance-critical.

Dynamic Provisioning of CDN Caches. Content delivery concerns the problem of moving content closer to end-users. For this, CDN caches are established in proximity to the access network. NFV offers the potential to dynamically provision CDN caches in which the physical infrastructure can be shared by multiple CDN operators. When using NFV to implement such a cache, not only network performance is crucial, but also storage requirements play a dominant role. The prototypical Xen-based implementation of a CDN cache shows that a virtualized cache can serve content at 10GE line-rate [39].

3.5 Security

Other example use cases concern achieving network security.

Firewalls. While stateless firewalls up to the transport layer can be implemented with SDN [41], more advanced threat mitigation techniques, that require at least TCP session tracking, need to be realized in software. This includes application layer analysis, e.g., signature-based malware detection. Firewalls resemble a suitable virtualization target since they can be scaled with network traffic demands. However, as for other network functions, realizing high packet forwarding performance is crucial.

DDoS Mitigation. Various strategies for thwarting distributed denial-of-service (DDoS) attacks in cloud datacenters exist [38]. While describing a method to detect long-lived flows and several attacks that may be mitigated with this technique, it was found that using NFV allows for faster time-to-market for novel attack mitigation strategies, better scalability and lower energy consumption.

4. NFV PERFORMANCE TODAY

The realization of NFV requires the achievement of high packet processing performance. However, if performance is not taken carefully into account during implementation, the interaction between the general-purpose nature of the CPU and its periphery, and the complexity of the software, can severely deteriorate the former. Depending on the respective application, performance figures include classical metrics, e.g., *delay*, *jitter*, or *throughput*. The latter is often expressed as *packets per second* (pps) to respect the header processing load of small packets. Besides these dataplane-related performance metrics, also the *startup* or *migration time* can become relevant, especially for highly-dynamic per-user VNFs.

Compared to the implementation in optimized hardware, the realization of NFV is challenged by a multitude of overheads in the entire software and virtualization stack. For example, it has been shown that existing protocol stacks are not suitable for achieving line-rate performance when processing small sized packets on multiple 10GE interfaces [45]. These effects can be more pronounced in virtualized environments such as NFV, when additional software layers or even kernel stacks need to be traversed. Furthermore, virtualization has been shown to introduce additional packet jitter [70]. As a result, attempts to bypass the kernel and to further optimize virtualization systems have been made [32, 63, 47, 46].

We therefore focus next on reviewing the current state-of-the-art solutions to address performance issues arising with the realization of NFV. Despite these advances, realizing performance-critical network functions in NFV (e.g., virtualized switches) are still challenged by the aforementioned imposed overheads. Thus, finding approaches to further improve performance is the first pressing research challenge required to convincingly demonstrate the feasibility of NFV (Section 5).

4.1 Virtual Packet Forwarding

The task of *virtual packet forwarding* between virtual machines and physical interfaces on the same hardware node is critical for the performance properties of an NFV infrastructure. The architectural component that implements virtual packet forwarding on a physical computing node is called vSwitch (Virtual Switch) because of its functional similarity to physical switches (see Figure 1a).

The **Linux bridge** [7] is part of the Linux kernel and allows the kernel to act as a vSwitch by instantiating *bridges*. Bridges are simple Layer 2 forwarding instances between virtual or physical interfaces. Despite its ease of use, its wide adoption and well-known stability, the Linux bridge lacks flexibility on the data plane. Especially for NFV, custom forwarding rules beyond L2 bridging are required, like offered by OpenFlow. **MacVTap** [5] is a collapsed network forwarding and TAP driver for Linux (see Figure 1b). It also does not support OpenFlow.

This lack of features is addressed by the *Open vSwitch* [56, 55]. It adds features of enterprise-class switches (e.g., LACP link aggregation, 802.1q VLAN tagging and QoS policing) and supports OpenFlow. The forwarding performance at single-core systems has been shown to reach 1.8 million packets per second (Mpps) as compared to the Linux bridge that only reaches 1.1 Mpps [27]. With increasing packet size, throughput in packets per time keeps the same, but is bounded by physical limits of the interface. The results show performance differences between different versions.

VALE [62] is another software switch, primarily developed for high-performance throughput between VMs. It is implemented as an extension of the **netmap** driver, which was developed by the same authors. Compared to Open vSwitch, it is very lightweight and does not contain advanced configuration backends. Furthermore, it is restricted to simple MAC-layer forwarding and does not support OpenFlow, thus it does not easily integrate into SDNs. The authors of VALE and Netmap claim a far better performance than Open vSwitch, however, this claim is based on a comparison with an OVS version modified to fit into the NetMap framework [61]. VALE is used by ClickOS (see Section 4.5).

When copying large amounts of data in memory, the negative impact on CPU consumption is tremendous [69]. Therefore, **zero-copy** forwarding is one of the most important goals of VM network forwarding. Optimally, the packet remains on the same spot in memory during the complete process on a single hardware node, and only a pointer to it is handed over to, from, or between VMs. To reach this goal, support for it must be realized in the vSwitch, the hypervisor, and the guest network drivers. Originally implemented in QEmu [9], IVSHMEM is a prominent enabler implementation to achieve this goal. IVSHMEM is also supported by Open vSwitch and DPDK (see Section 4.3). However, zero-copy as proposed by IVSHMEM leaves a lot of open questions, like if this support can be extended to different types of VNFs, for example. Furthermore, zero-copy arises security concerns for the packet buffer memory pages, as all components involved require complete access to that memory region.

4.2 Hardware-Assisted Packet Forwarding

Software switches (e.g., the Linux bridge or Open vSwitch) are complemented by hardware-assisted approaches for faster processing. These techniques offload the general-purpose processor and speed up network performance [42].

The most prominent standard is *single-root I/O virtualization* (SR-IOV) by the PCI Special Interest Group (PCI-SIG). A SR-IOV-capable network card (see Figure 1c) appears as a PCI device to the host, providing a *physical function* (PF). With the PF driver, the host is able to configure internal, hardware-based virtual bridges and spawn multiple virtual network interfaces, so-called *virtual functions* (VFs). Virtual functions appear as additional stand-alone PCI devices to the host. These interfaces then may be passed to different VMs via *PCI passthrough*. However, beyond the fact that SR-IOV-capable network interfaces may be costly, the hardware-based approach has several limitations. On the one hand, flexibility is reduced as the forwarding is restricted to the vendor's hardware capabilities (which may be restricted to common layer 2 bridging). On the other hand, the benefit of hardware abstraction is lost.

A principle similar to SR-IOV is introduced with *Virtual Machine Device Queues (VMDq)*, however with a very different architectural approach: By default, a network card writes into a single receive queue. The vSwitch then must decide about to which instance queue to forward the packet, which consumes a major amount of CPU cycles. VMDq relieves the CPU from the latter task by creating multiple receive queues, and lets the network card classify the packets in hardware to decide where to enqueue them. This classification is done by application- or user-defined rules. The main difference of SR-IOV compared to VMDq is that multiple virtual interfaces are created, while VMDq just multiplies receive queues. VMDq may not only benefit the performance of vSwitches, but also the one of a single network function (Section 4.3). Therefore, VMDq is a starting point for solving the problem of efficient input queue parallelization. However, being subject to very individual requirements from future VNFs to prioritize and distribute the packets to different queues and cores, the limits of VMDq for the parallelization of a single VNF could be quickly reached.

4.3 Network Stack Offloading

It has been shown that the performance of current network stacks does not scale with increasing line-rate, especially for small packets [45]. As a result, offloading techniques have been proposed that either bypass the kernel or offload functionality to hardware.

The first class of approaches improve the packet processing performance by bypassing the network stack. The two prominent approaches include netmap [62] and the Data Plane Development Kit (DPDK) [21]. These frameworks allow the implementation of accelerated network functions by obtaining raw and exclusive access to the NIC and thus bypassing the network stack entirely. This advantage comes at the price of requiring specialized user-land stacks performing the packet processing, instead of relying on well-maintained kernel stacks. To ease the implementation of packet processing functionality, the DPDK provides a large library of common packet processing code, including but not limited to queue management, packet classification, poll-mode drivers (PMD), packet header structs, and checksum computing [2]. DPDK supports IVSHMEM (Section 4.1). A prominent example of an application is Open vSwitch, which can optionally make use of the DPDK for its physical interfaces to increase performance. It has been shown that the already mentioned network throughput can be increased by a factor of 6 (physical-to-physical NIC) when additionally using the DPDK for acceleration [27].

The **OpenDataPlane** [8] project uses a very similar approach to DPDK. The most important difference is that the project puts a larger emphasis on a broad standardization of the network interface API. A further example in the area of kernel bypassing is the aforementioned *netmap* driver [62], which has been shown to significantly improve packet processing performance [45, 62].

The mentioned frameworks and kits for network stack offloading are a basis for developing high-performance network functions, also through allowing the developer to exploit NIC acceleration features independently of kernel support. Thus, they do not provide a full solution, but leave a lot of open questions and alternatives to implement network functions. This way, they provide the research platform of choice compared to kernel-based drivers.

Another class of approaches involves *function offloading*. One line of research proposes to (partially) offload packet processing functionality to dedicated hardware (see e.g., [31, 57]). Another line of research proposes to short-cut packet processing by offloading functionality directly to stack. For example, Santa [63] provides an application-agnostic kernel-level cache of frequent requests and offers potential for drastic performance improvements. The latter class of approaches denotes *specialized* solutions to implement high-performance network functions. Given their specialized nature, they either require specialized hardware—which might not be always available—or are not generally applicable to all kinds of NFV workloads. Performance benchmarks are required to first identify whether the approaches in this chapter fulfill the requirements / workload of carrier-grade network functions, or whether different architectural designs are required. Thus the question of the proper network architecture design for NFV is left open.

4.4 VM Network I/O Optimization

Different virtualization architectures exist that have different impacts on performance. Full virtualization allows to run unmodified guest operating systems since legacy devices must be emulated in software. In contrast, paravirtualization requires a special driver on the guest operating system, while the host uses an appropriate backend driver. From the performance perspective, it is desired to use paravirtualized guests. The reason is that the emulation incurs additional VM overhead and redundant context switching, while

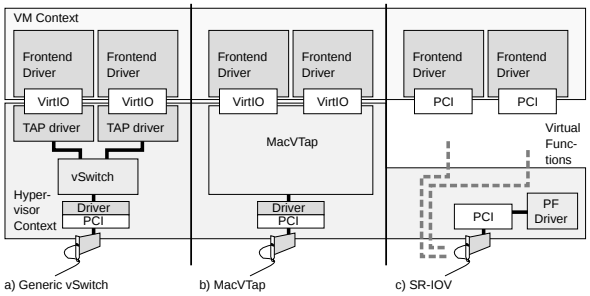

Figure 1: Architectures for VM network forwarding and I/O

paravirtualized drivers can be optimized for the special needs of the communication between hypervisor and VM. **VirtIO** [6] is a standard I/O interface for paravirtualization. It allocates shared memory, which is accessible from the hypervisor as well as the VM. The *vhost-net* backend driver provides improved efficiency on the Linux platform by moving packet forwarding to the kernel.

Context switching causes a large amount of overhead in virtualization environments [15, 74]. Thus, besides of shortening the I/O path with the usage of the hardware-abstract VirtIO or the hardware-based SR-IOV, further performance can be achieved by the reduction of context switching. Every packet which arrives may generate an interrupt, leading to a VM exit, so that the kernel can hand over the new packet to the VM driver. In contrast, poll-mode drivers (PMD) let the network interface buffer the packet, and hand it over upon a driver's request (polling). These kind of drivers do not require interrupts for operation, however, polling consumes CPU resources also during potential idle times, deteriorating resource availability or energy efficiency in this case. An alternative to polling is *interrupt coalescing*. Here, interrupts are delayed until a certain threshold is reached [26].

To address delayed interrupts, *Exitless Interrupts (ELI)* were proposed. The idea is to keep the interrupts, but reduce the need of context switching per interrupt. Instead of the VM receiving virtual interrupts from the host, the VM uses a shadow interrupt descriptor table (IDT), where interrupts are delivered directly by hardware. The ELI concept was incorporated into the *Exitless Virtual I/O System (ELVIS)* [30].

The performance of different virtual state-of-the-art I/O drivers has been evaluated by Intel [34]. The system under test was a KVM hypervisor running one virtual machine, on which a netperf benchmark was conducted. The drivers under test were QEMU's e1000, VirtIO and vhost-net. Regarding *throughput*, the e1000 driver only achieves a fraction of the performance of VirtIO and vhost-net, where the latter achieve similar performance. DPDK and SR-IOV support can further improve network throughput. Latency comparisons were conducted by Wagner et al. [60]. Here, vhost-net clearly outperforms userland-based VirtIO with a backend driver moved to the kernel and thus quicker reaction to events.

Besides these very recent and promising first steps to optimize virtualization for network I/O, a variety of research questions are still left open. The design and evaluation of the proper NFV virtualization architecture requires an in-depth study of performance requirements and bottlenecks in virtualized network functions. Such performance centric aspects concern the optimization of context switching and interrupts, depending on the workload characteristics. Another relevant research area may reside in the correct trade-off between the advantages and disadvantages of PMDs and interrupts. While the former commonly perform better when high, con-

stant throughput is required, the latter produce fewer costs regarding CPU utilization and efficiency when throughput requirements are low.

4.5 Operating System Complexity

Current research attempts question the use of fully-fledged operating systems and instead propose tailored kernels to realize NFVs. Fully-fledged OSes are desired for server applications due to flexibility, easier software development and configuration. However, in the case of NFV, network performance is critical and challenged by such a fully-fledged software stack.

A first step to address this challenge is taken by two recent research prototypes based on the Xen hypervisor. ClickOS [46, 47] realization of network functions is based on a light-weight unikernel only executing the Click modular router. This architecture allows to execute each NFV in a dedicated VM that can be quickly instantiated. The realization of ClickOS showed that several optimizations of the software stack are required to achieve high performance figures. These modifications concern the VM network forwarding architecture as well as the Xen frontend and backend network drivers. In a similar fashion, Jitsu [44] focuses on achieving resource isolation by quickly booting unikernels on demand to handle network requests.

These attempts open the question on the appropriate software architecture to realize NFV. In the case of OS kernels, the question concerns trade-offs in the use of tailored unikernels that quickly boot vs. versatile fully-fledged commodity operating systems. Future work concerns the identification and evaluation of appropriate architectures for the multitude of available network functions.

5. RESEARCH CHALLENGES

Bringing NFV from a draft to an implementation state poses a set of research questions that need to be addressed by the community. We next discuss relevant questions to outline a research agenda.

5.1 NFV Performance Improvement

Local performance. The very first challenge arising when realizing NFV is to achieve high packet processing performances in virtualized environments. Optimizing this *local performance* concerns the whole spectrum of involved software, from kernels and network stacks to virtualization architectures. Despite the advances described in Section 4, the attempts to realizing performance-critical network functions in NFV are still challenged by the overheads imposed by software and virtualization stacks.

These overheads open research opportunities for optimizing the entire NFV stack for speed. Such an optimization is challenged by different demands imposed by different network functions. For example, some functions will require a high packet processing *throughput* (e.g., TCP optimizers), while others will require stable sessions with a low *latency* (e.g., Session Border Controller).

Besides the overheads, performance challenges arise from the present hardware architectures making use of *sequentially* working processors. Especially for *compute-intensive* network functions, it might be investigated if and how it is possible to parallelize operations on the same packet, for example, multiple cores could manipulate a single packet *in parallel* in a zero-copy memory region. A similar idea is to add reprogrammable acceleration hardware like FPGAs to an NFV infrastructure [18, 19] allowing for short, guaranteed delay and a high throughput. Reconfigurable FPGAs or NPUs preserve the flexibility target of NFV and open up opportunities for elastic redistribution of workloads with extraordinarily high performance requirements to acceleration hardware [54]. These aspects open the question for new architectural designs. While first approaches for some aspects have been proposed, the question of the proper architectural design of NFV software stacks is far from being resolved.

Global performance. The challenge of optimizing the NFV software stack (*local* performance) is complemented by the challenge to *globally* optimize the performance of network functions in the entire network. This challenge is stimulated by prior research which showed that splitting state and functionality eases the global scaling of network functions [59]. This observation further fosters research aiming to get a firm understanding of means to implement distributed network functions in a performance-efficient manner.

5.2 NFV Performance Benchmarking

To legitimately glorify or condemn NFV, and even more to compare existing solutions, it is required to have correctly quantified NFV performance characteristics. First, performance evaluations are required to study which functions—under which workloads—are suitable for commodity hardware implementation and virtualization. It has to be shown for which network functions not only the performance, but also the promised benefits of increased resource utilization, improved energy efficiency, and decreased costs can be achieved. Secondly, the performance of different NFV technology, architecture and implementation alternatives need to be compared preferably with a benchmark, which provides standardized results for even later, independent comparison. Existing standards for the benchmarking of hardware middleboxes, for example RFC 2544, do not fully respect the increased complexity caused by the entanglement between VNFs and their underlying infrastructure.

In a typical NFV architecture, the problem of NFV benchmarking can be separated into NFV *infrastructure* benchmarking and *VNF* benchmarking. The former one evaluates the generic platform which forwards traffic to VNFs and constitutes the execution environment in which they operate, for example network and server hardware, hypervisor software, vSwitches, but also network and NFVI controllers. In contrast, the latter one evaluates the implementation efficiency of VNF-specific algorithms, design choices, and their realization in code.

One can revert to a variety of metrics to calculate a benchmark. Besides the aforementioned well-known performance metrics like delay, jitter, and throughput, for either an individual VNF node or a pre-defined NFVI, a complex set of cost metrics could be gathered from the set of components. Finally, the methods to obtain them might strongly differ. The software nature of VNFs allows the direct application of a broad range of software benchmarking methodologies. In turn, *black-box* stress tests run the code in a testing infrastructure to evaluate those parameters.

To assist network planning, insights gained in empirical performance evaluations should be reflected in NFV performance models. The so created models capture the workload dependent packet processing performance of network functions and can be used for function placement and network dimensioning. Workload models can be further used for traffic generation and benchmarking.

5.3 Further NFV Research Challenges

Orchestration and Reliability. Works on SDN have progressed to realize different protocols and controller architectures, to address reliability challenges and applications (see [35] for an extensive overview). In comparison to SDN related efforts, NFV is still in draft state in which a general execution and orchestration platform is still missing. A recently formed initiative focuses on realizing such a platform as open source (see `www.opnfv.org`). Further efforts include state transfers between network functions [29], functional blocks, controllers, and descriptive languages [14]. De-

spite these initial efforts, many questions concerning the algorithmic design for the dynamic orchestration of network functions and methods for achieving reliability are still unanswered. In particular designing algorithms for operating virtualized networks in a reliable manner will be critical for the success of NFV.

Energy Efficiency. The dynamic provisioning of network functions has the potential to improve the energy efficiency of carrier networks. Answers to this question depend, however, on a multitude of system factors involving network function placement, workloads, system and data center locations, and orchestration algorithms. The design of energy efficient NFV networks thus requires the development of new algorithm controlling the function placement and evaluations of the different system designs and influence parameters.

Security and Privacy. The relocation of network functions to cloud services has the potential for creating new challenges on data privacy and network security. An example concern ISP efforts to extend the users' home networks to carrier clouds by virtualizing residential gateways (see Section 3.2). These effects are expected to be further pronounced when carrier clouds are scaled out to public clouds run by third parties. Clouds not only pose new challenges in terms of security and privacy but also offer a new set of solutions. For example, one could scale a service as one of the measures to mitigate a DDoS attack. This motivates the study of these new challenges and an open discussion disputing on these advances in network operation.

6. SUMMARY

Emerged in late 2012, Network Functions Virtualization (NFV) describes a new network management paradigm in which network functions are abstracted from dedicated hardware to virtualized machines running on commodity hardware. It addresses the desire to manage networks in a more scalable and flexible manner. This paper provides the first overview of an emerging area that currently resides in a draft state and is slowly being addressed by the research community. First signs of adoption are the growing body of academic literature and the first formation of NFV related venues (e.g., the SDNFlex workshop and the IEEE Conference on NFV and SDN to be held in 2015 for the first time). We focus on reviewing these research attempts and use cases to complement industry driven efforts by the standardization bodies ETSI and IETF. Our discussion of research challenges aims at outlining a first research agenda on NFV.

7. ACKNOWLEDGEMENTS

This work has been funded by the German Research Foundation (DFG) within the Collaborative Research Center (CRC) 1053 – MAKI as well as by Deutsche Telekom through the project "Dynamic Networks".

8. REFERENCES

[1] Address family transition router reference implementation by the internet systems consortium. http://www.isc.org/downloads/aftr/.

[2] DPDK: API documentation. http://dpdk.org/doc/api/.

[3] ETSI NFV open documents. http://docbox.etsi.org/ISG/NFV/Open/Published/.

[4] IETF: Network function virtualization research group. https://trac.tools.ietf.org/group/irtf/trac/wiki/nfvrg.

[5] Kernel Newbies: MacVTap. http://virt.kernelnewbies.org/MacVTap.

[6] KVM: VirtIO. http://www.linux-kvm.org/page/Virtio.

[7] Linux Foundation: bridge. http://www.linuxfoundation.org/collaborate/workgroups/networking/bridge.

[8] OpenDataPlane - Project Website. http://www.opendataplane.org/.

[9] QEmu: IVSHMEM implementation. https://github.com/qemu/qemu/blob/master/hw/misc/ivshmem.c.

[10] Service function chaining IETF working group. https://datatracker.ietf.org/wg/sfc/documents/.

[11] Network functions virtualisation - introductory white paper. In *SDN and OpenFlow World Congress*, 2012.

[12] ETSI: Network functions virtualisation (NFV); use cases, 2013.

[13] ETSI GS NFV 003: Network functions virtualisation (NFV); terminology for main concepts in NFV, 2014.

[14] ETSI GS NFV-MAN 001: Network functions virtualisation (NFV); management and orchestration, 2014.

[15] M. Ben-Yehuda, M. D. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B.-A. Yassour. The turtles project: Design and implementation of nested virtualization. In *OSDI*, 2010.

[16] R. Bifulco, T. Dietz, F. Huici, M. Ahmed, J. Martins, S. Niccolini, and H.-J. Kolbe. Rethinking access networks with high performance virtual software BRASes. In *EWSDN*, 2013.

[17] J. Blendin, J. Rückert, N. Leymann, G. Schyguda, and D. Hausheer. Position paper: Software-defined network service chaining. In *EWSDN*, 2014.

[18] Z. Bronstein, E. Roch, J. Xia, and A. Molkho. Uniform handling and abstraction of nfv hardware accelerators. *IEEE Network*, 29(3):22–29, May 2015.

[19] S. Byma, J. G. Steffan, H. Bannazadeh, A. L. Garcia, and P. Chow. Fpgas in the cloud: Booting virtualized hardware accelerators with openstack. In *IEEE FCCM*, pages 109–116, May 2014.

[20] G. Carella, M. Corici, P. Crosta, P. Comi, T. Bohnert, A. Corici, D. Vingarzan, and T. Magedanz. Cloudified IP multimedia subsystem (IMS) for network function virtualization (NFV)-based architectures. In *IEEE Symposium on Computers and Communication*, 2014.

[21] I. Cerrato, M. Annarumma, and F. Risso. Supporting fine-grained network functions through Intel DPDK. In *EWSDN*, 2014.

[22] G. Chen and H. Deng. IPv6 considerations for network function virtualization (NFV). IETF Draft, 2014.

[23] N. M. K. Chowdhury and R. Boutaba. A survey of network virtualization. *Computer Networks*, 54(5):862–876, 2010.

[24] S. Davy, J. Famaey, J. Serrat-Fernandez, J. Gorricho, A. Miron, M. Dramitinos, P. Neves, S. Latre, and E. Goshen. Challenges to support edge-as-a-service. *IEEE Communications Magazine*, 52(1):132–139, Jan. 2014.

[25] M. Dillon and T. Winters. Virtualization of home network gateways. *IEEE Computer*, 47(11):62–65, Nov. 2014.

[26] Y. Dong, D. Xu, Y. Zhang, and G. Liao. Optimizing network I/O virtualization with efficient interrupt coalescing and virtual receive side scaling. In *IEEE CLUSTER*, 2011.

[27] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle. Performance characteristics of virtual switching. In *IEEE CloudNet*, 2014.

[28] N. Figueira and R. Krishnan. Policy architecture and framework for nfv and cloud services. IETF Draft, 2015.

[29] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella. OpenNF: Enabling innovation in network function control. In *ACM SIGCOMM*, 2014.

[30] A. Gordon, N. Har'El, A. Landau, M. Ben-Yehuda, and A. Traeger. Towards exitless and efficient paravirtual I/O. In *ACM Systems and Storage Conference*, 2012.

[31] S. Han, K. Jang, K. Park, and S. Moon. PacketShader: A GPU-accelerated software router. In *ACM SIGCOMM*, 2010.

[32] J. Hwang, K. K. Ramakrishnan, and T. Wood. NetVM: High performance and flexible networking using virtualization on commodity platforms. In *USENIX NSDI*, 2014.

[33] C.-L. I, J. Huang, R. Duan, C. Cui, J. Jiang, and L. Li. Recent progress on C-RAN centralization and cloudification. *IEEE Access*, 2:1030–1039, 2014.

[34] Intel. Network function virtualization: Packet processing peformance of virtualized platforms with Linux* and Intel architecture. Technical Report.

[35] D. Kreutz, F. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *arXiv preprint arXiv:1406.0440*, 2014.

[36] R. Krishnan, N. Figueira, D. Krishnaswamy, D. R. Lopez, S. Wright, and T. Hinrichs. NFVIaaS architectural framework for policy based resource placement and scheduling. IETF Draft, 2014.

[37] R. Krishnan, D. Krishnaswamy, D. R. Lopez, A. Qamar, S. Wright, and N. Figueira. Nfv real-time analytics and orchestration: Use cases and architectural framework. IETF Draft, 2014.

[38] R. Krishnan, D. Krishnaswamy, and D. Mcdysan. Behavioral security threat detection strategies for data center switches and routers. In *IEEE ICDCSW*, 2014.

[39] S. Kuenzer, J. Martins, M. Ahmed, and F. Huici. Towards minimalistic, virtualized content caches with minicache. In *ACM HotMiddlebox*, 2013.

[40] S. Lee, S. Pack, M.-K. Shin, and E. Paik. Resource management for dynamic service chain adaptation. IETF Draft, 2014.

[41] Y.-D. Lin. Research roadmap driven by network benchmarking lab (NBL): Deep packet inspection, traffic forensics, embedded benchmarking, software defined networking and beyond. *International Journal of Networking and Computing*, 4(2):223–235, 2014.

[42] J. Liu. Evaluating standard-based self-virtualizing devices: A performance study on 10 GbE NICs with SR-IOV support. In *IEEE IPDPS*, 2010.

[43] A. Lometti, C. Addeo, I. Busi, and V. Sestito. Backhauling solutions for LTE networks. In *International Conference on Transparent Optical Networks*, 2014.

[44] A. Madhavapeddy, T. Leonard, M. Skjegstad, T. Gazagnaire, D. Sheets, D. Scott, R. Mortier, A. Chaudhry, B. Singh, J. Ludlam, J. Crowcroft, and I. Leslie. Jitsu: Just-in-time summoning of unikernels. In *USENIX NSDI*, 2015.

[45] I. Marinos, R. N. Watson, and M. Handley. Network stack specialization for performance. In *ACM SIGCOMM*, 2014.

[46] J. Martins, M. Ahmed, C. Raiciu, and F. Huici. Enabling fast, dynamic network processing with clickos. In *ACM HotSDN*, 2013.

[47] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici. ClickOS and the art of network function virtualization. In *USENIX NSDI*, 2014.

[48] C. Meirosu, A. Manzalini, J. Kim, R. Steinert, S. Sharma, and G. Marchetto. Devops for software-defined telecom infrastructures. IETF Draft, 2014.

[49] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys Tutorials*, (99), Sept. 2015.

[50] G. Monteleone and P. Paglierani. Session border controller virtualization towards "service-defined" networks based on NFV and SDN. In *IEEE SDN4FNS*, 2013.

[51] V. G. Nguyen and Y. H. Kim. Slicing the next mobile packet core network. In *IEEE ISWCS*, 2014.

[52] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet. Openairinterface: A flexible platform for 5g research. *ACM CCR*, 44(5):33–38, Oct. 2014.

[53] H. Niu, C. Li, A. Papathanassiou, and G. Wu. Ran architecture options and performance for 5G network evolution. In *IEEE WCNCW*, 2014.

[54] L. Nobach and D. Hausheer. Open, elastic provisioning of hardware acceleration in NFV environments. In *IEEE NetSys*, pages 1–5. 2015.

[55] J. Pettit, J. Gross, B. Pfaff, M. Casado, and S. Crosby. Virtual switching in an era of advanced edges. In *Workshop on Data Center–Converged and Virtual Ethernet Switching*, 2010.

[56] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker. Extending networking into the virtualization layer. In *ACM HotNETs*, 2009.

[57] I. Pratt and K. Fraser. Arsenic: A User-Accessible Gigabit Ethernet Interface. In *IEEE INFOCOM*, 2001.

[58] Z. Qiang. Elasticity VNF. IETF Draft, 2014.

[59] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield. Split/Merge: System support for elastic execution in virtual middleboxes. In *USENIX NSDI*, 2013.

[60] S. Rao and M. Wagner. Achieving peak performance from Red Hat KVM-based virtualization. Technical report, 2010.

[61] L. Rizzo, M. Carbone, and G. Catalli. Transparent acceleration of software packet forwarding using netmap. In *IEEE INFOCOM*, 2012.

[62] L. Rizzo and G. Lettieri. Vale, a switched ethernet for virtual machines. In *ACM Conference on Emerging Networking Experiments and Technologies*, 2012.

[63] F. Schmidt, O. Hohlfeld, R. Glebke, and K. Wehrle. [poster abstract] Santa: Faster packet delivery for commonly wished replies. In *ACM SIGCOMM Poster*, 2015.

[64] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making middleboxes someone else's problem: Network processing as a cloud service. In *ACM SIGCOMM*, 2012.

[65] M.-K. Shin, K. Nam, S. Pack, and S. Lee. Verification of nfv services: Problem statement and architecture. IETF Draft, 2014.

[66] R. Szabo, A. Csaszar, K. Pentikousis, M. Kind, and D. Daino. Unifying carrier and cloud networks: Problem statement and challenges. IETF Draft, 2014.

[67] T. Taleb. Toward carrier cloud: Potential, challenges, and solutions. *IEEE Wireless Communications*, 21(3):80–91, 2014.

[68] T. Taleb, A. Ksentini, and A. Kobbane. Lightweight mobile core networks for machine type communications. *IEEE Access*, 2:1128–1137, Sept. 2014.

[69] K. Vaidyanathan, W. Huang, L. Chai, and D. K. Panda. Designing efficient asynchronous memory operations using hardware copy engine: A case study with i/oat. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–8. IEEE, 2007.

[70] J. Whiteaker, F. Schneider, and R. Teixeira. Explaining packet delays under virtualization. *ACM CCR*, 41(1):38–44, Jan. 2011.

[71] L. Xia, Q. Wu, D. King, H. Yokota, and N. Khan. Requirements and use cases for virtual network functions. IETF Draft, 2014.

[72] H. Xie, Y. Li, J. Wang, D. Lopez, T. Tsou, and Y. Wen. vRGW: Towards network function virtualization enabled by software defined networking. In *IEEE ICNP*, 2013.

[73] Y. Zaki, L. Zhao, C. Goerg, and A. Timm-Giel. Lte wireless virtualization and spectrum management. In *IFIP WMNC*, 2010.

[74] B. Zhang, X. Wang, R. Lai, L. Yang, Z. Wang, Y. Luo, and X. Li. Evaluating and optimizing I/O virtualization in kernel-based virtual machine (KVM). In *Network and Parallel Computing*. 2010.