Empowering Cloud Computing With Network Acceleration: A Survey

Lorenzo Rosa, *Graduate Student Member, IEEE*, Luca Foschini, *Senior Member, IEEE*, and Antonio Corradi, *Life Senior Member, IEEE*

***Abstract*—Modern interactive and data-intensive applications must operate under demanding time constraints, prompting a shift toward the adoption of specialized software and hardware network acceleration technologies. This specialization, however, poses significant scalability, flexibility, security, and economic sustainability challenges for application developers. Cloud computing holds the potential to overcome these obstacles by offering the cost-effective option to access specialized acceleration technologies through standard cloud interfaces. Nevertheless, that integration is still challenging for cloud providers. In the cloud, physical resources are hidden behind a virtualization layer, whereas acceleration technologies make applications directly interact with the hardware. To bridge this gap, recent literature explores the possibility of empowering cloud platforms with accelerated networking as a commodity, thus offering the innovative option of *Network Acceleration as a Service*. This survey reviews these recent research efforts by adopting popular technologies like XDP, DPDK, and RDMA as a reference. To organize the surveyed research in a comprehensive framework, we identify four key aspects that pose critical problems to the integration of acceleration options in cloud computing - access interfaces, virtualization techniques, serviceability, and security - and systematically discuss the associated challenges. Then, we present the issues to be further addressed and outline the most promising research directions for the full integration of network acceleration within next-generation cloud computing platforms.**

***Index Terms*—Cloud computing, network acceleration, next-generation networks, RDMA, XDP, DPDK.**

I. Introduction

IN THE last two decades, cloud computing has become a cornerstone of the modern digital economy, pushing organizations across the globe to radically shift their approach to IT resources, now perceived as *utilities* accessed anytime from anywhere on a *pay-per-use* basis. On the provider side, the success of this model is rooted in the possibility of achieving significant economies of scale. By efficiently spreading the high costs of operations, such as hardware purchase, maintenance, power supply, etc., over a large customer base, cloud providers can offer IT resources *as a service* at competitive prices. From a technical perspective, these advantages descend from the use of *homogeneous general-purpose* technologies at scale, which allows providers to both maximize customers and reduce costs by purchasing large batches of off-the-shelf hardware components [1], [2].

Today, we are at the dawn of another revolution in the computing domain. The widespread adoption of the Internet of Things (IoT) concept [3] and the ubiquitous availability of Internet services is fostering the emergence of a new generation of *interactive and data-intensive* applications in areas such as transportation, industrial automation (Industry 4.0), healthcare, telecommunications, education, entertainment, and many others. These applications are designed to produce timely answers to either user prompts or events generated by devices: next-generation communication networks (5G and beyond) are already being designed to enable high-throughput and low-latency traffic processing to enable customized and real-time decision-making, innovative adaptive services, and advanced forms of automation [4], [5], [6], [7], [8], [9], [10], [11].

However, these applications are very challenging to support within the well-established cloud computing model, as they have demanding requirements: to move, query, and process a large amount of data under tight time constraints. Thus, these applications achieve their full potential only when executing either on *specialized software stacks* or on *hardware accelerators*, specialized hardware that implements basic functions such as computing (e.g., Graphics Processing Units, GPUs), storage (e.g., Non-Volatile Memory Express, NVMe), and networking (e.g., *Smart* Network Interface Cards, SmartNICs). The performance advantages offered by these acceleration options have increasingly pushed companies to prefer them to the standard, general-purpose cloud resources [12], [13], [14].

Ideally, the availability of acceleration technologies in the cloud would couple the advantages of the cloud model with their performance benefits. Cloud providers are well positioned to fill this gap, as they already own the necessary infrastructure and have the expertise to manage it. However, the adoption of special-purpose software stacks and devices comes with several technical challenges, ultimately related to a lack of support from developers and manufacturers for virtualization and sharing among multiple users [13], [14], [15], [16], [17]. For this reason, providers are offering acceleration only through forms of dedicated physical resources (*bare-metal instances*) [18], [19], [20]. This solution is neither ideal nor cost-effective, as it forces providers to manage separate infrastructures and users to pay higher prices to rent these services.

Manuscript received 30 June 2023; revised 7 December 2023, 17 January 2024, and 11 March 2024; accepted 12 March 2024. Date of publication 14 March 2024; date of current version 22 November 2024. This work was supported in part by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" under Grant PE00000001 (Program "RESTART"). *(Corresponding author: Lorenzo Rosa.)*

The authors are with the Department of Computer Science and Engineering, University of Bologna, 40126 Bologna, Italy (e-mail: lorenzo.rosa@unibo.it; luca.foschini@unibo.it; antonio.corradi@unibo.it).

Digital Object Identifier 10.1109/COMST.2024.3377531

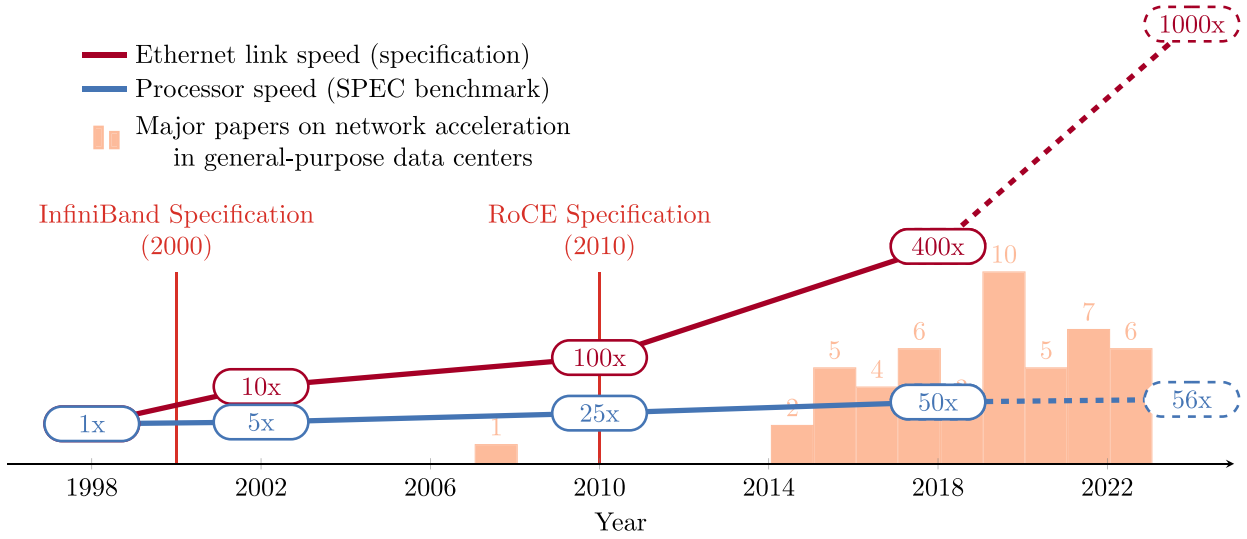


Fig. 1. A comparison between the performance improvement of Ethernet link speeds and processor speeds in the last 25 years. As the gap widens, papers concerning network accelerators in general-purpose data centers start to appear in the literature.

In this paper, we survey recent proposals in the computer system literature that introduce the idea of full integration of acceleration technologies as a commodity option in general-purpose cloud platforms. In particular, our work considers a specific aspect of cloud computing, the networking infrastructure, because it represents the most relevant example of the rapid and challenging evolution of the cloud landscape. Accordingly, we classify the relevant papers in a taxonomy that reflects the most important challenges related to the availability of ***Network Acceleration as a Service*** **(*NAaaS*)** in public clouds: *access interfaces*, *virtualization techniques*, *serviceability*, and *security*. Our discussion will cover the various technical, operational, and economic aspects of network acceleration in the cloud, including architectures, protocols, performance, security, and deployment challenges.

To motivate the need for acceleration in cloud networking, Fig. 1 shows the evolution of the cloud networking scenario over the past few years: the growth rate of communication links in data centers has largely outpaced the performance improvements of other host resources, in particular the processor cores. With the end of the Moore's law, processor performance increased of just about 3.5% per year in the last eight years, in sharp contrast with the rapid standardization of higher Ethernet network link speeds [21], [22]. This different performance evolution trend reverses the traditional assumption in computer systems that network operations are slower than host processors in processing packets: because the CPU is involved in the data processing pipeline, the standard networking stack available in common operating systems is becoming the bottleneck of data center networking [23], [24], [25].

At the same time, the increasing amount of CPU cycles spent for high-performance networking is subtracted from user applications, i.e., to the core business of cloud and data center providers. For example, in a typical long-lived TCP flow between two hosts, more than 50% of the total CPU cycles are spent for data copies at the end hosts [23]. For an average request, a server running a simple key-value store spends only 6% of the total CPU cycles within the application logic, whereas 85% is spent within the kernel networking stack [25].

As a consequence, application developers are increasingly shifting to use network acceleration technologies. By minimizing the processor intervention on data plane operations, these technologies let applications leverage the full speed of modern communication links, and providers to dedicate a bigger portion of CPUs to user applications. Fig. 1 correlates the different growth rates of network and processors speed with the number of major research papers that use network acceleration technologies to speed-up typical data center services. As the gap between the two curves widens, the number of papers increases and stresses the urgency to find efficient ways to include high-performance networking within general-purpose cloud infrastructures [14], [26], [27], [28], [29], [30].

In this context, a few acceleration technologies gained momentum as cost-efficient options: the Linux *eXpress Data Path* (XDP) [31], the *Data Plane Development Kit* (DPDK) [32], and *Remote Direct Memory Access* (RDMA) [33]. As we further comment in Section III, they all achieve significantly higher throughput and lower latency than standard networking by following common design principles, such as zero-copy transfers, minimal context switching, and asynchronous processing. The main difference among these options is how they implement these principles, as they target different needs and yield to different benefits [14]. XDP provides in-kernel acceleration. DPDK bypasses the OS kernel and handles networking from userspace. RDMA takes a step further by following a different communication model: it allows remote processes to share memory areas over the network and to transparently offload the necessary operations to a special network card called RDMA NIC

(RNIC), which can be implemented using a wide range of hardware devices [33], [34], [35].

Despite the advantages that *NAaaS* would bring to cloud users both in terms of increased performance and CPU overhead reduction, cloud providers remain skeptical about this option because it comes with unsolved challenges, especially for hardware-based acceleration. For example, is it possible to *preserve the native performance of acceleration technologies while integrating them* into a cloud infrastructure in a scalable and cost-effective way? How can providers adapt their internal services to network operations that bypass the existing *control infrastructure*? The proposals we survey in this paper address these questions in a fragmented way: each solution tends to investigate one or a few specific aspects of this topic, such as which network access interface the provider should expose to users, how efficiently the data path between applications and hardware devices can be virtualized, how such a path can be controlled to enforce different levels of Quality of Service, and how such access can be secured from internal or external malicious users. Instead of providing a complete context of usage and a consequent operation model, the outcomes of such research are sometimes not compatible or even clashing with one another, and a unifying idea is still missing.

Conversely, enabling accelerated networking in cloud platforms is a challenge associated with the collaboration and interplay of several stakeholders, including hardware manufacturers, operating system designers, cloud providers, and even application developers, and all have to agree on common principles at several abstraction levels.

From our perspective, we believe that a standardization effort is crucial for the evolution of both network acceleration technologies and the cloud paradigm. Since data-intensive applications are increasingly common, the ecosystems of cloud computing and of network acceleration must converge, closing the existing technological and architectural gaps while accelerated cloud systems are still at a prototype stage, and avoiding the proliferation of divergent proposals. This paper focuses on those gaps by providing a comprehensive picture of the state of the art on the availability of *NAaaS* in cloud data centers, with the goal of describing the main research challenges to be faced, how these challenges have been addressed in the literature, and if and how these approaches can be combined together.

TABLE I
CONTRIBUTIONS OF PREVIOUS SURVEYS AND COMPARISON

Surveys	Key Contributions
[11], [36]–[38]	Focusing on NFV, these works survey the hardware and software acceleration options for virtual network functions, including those deployed in the cloud, but do not consider the approaches to improve the cloud integration of network acceleration technologies.
[13]	This survey categorizes the hardware acceleration technologies available today, not limited to networking but including any aspect of computing and communication. The problem of cloud integration of these devices is discussed as an open research challenge.
[14]	This survey categorizes the software network acceleration technologies available today, including XDP and DPDK, their applicability in real-world scenarios, and some related use cases. The problem of cloud integration is only marginally discussed with reference to I/O virtualization solutions.
[15], [16]	These works review the virtualization, scheduling, and isolation techniques to integrate respectively GPUs and FPGAs into cloud infrastructures, mostly to accelerate *compute-intensive* tasks. Focusing on a specific device, they consider various deployment options and assess the associated challenges.
Our survey	This work reviews the state of the art of the availability of *NAaaS* in cloud platforms. We consider how cloud platforms are impacted by the integration of network acceleration technologies, survey the literature proposals to implement it, and discuss the associated open challenges and future research directions.

A. Related Surveys

Previous surveys investigated the interplay between cloud computing and network acceleration techniques in connection with the concept of Network Function Virtualization (NFV). NFV is a paradigm in network architecture that promotes the replacement of dedicated hardware middleboxes with more flexible software instances that can be dynamically scaled and migrated. However, general-purpose hardware cannot achieve the same performance as special-purpose application-specific devices. Thus, previous research reviewed not only the deployment options for virtualized network functions, including VMs and containers in public clouds, but also the possible software and hardware acceleration technologies to improve the performance of their network operations [11], [36], [37], [38].

In particular, Linguaglossa et al. [36] and Shantharama et al. [37] review several network acceleration technologies and acknowledge the performance overhead associated with their virtualization, suggesting that future research must focus on the challenge of providing better virtualization support.

In this work, we review recent contributions targeting that challenge. Our survey adopts a holistic approach and considers any application that needs to efficiently access network acceleration, regardless of its business logic: we survey the techniques to minimize the overhead introduced by the cloud virtualization layer. For a telco operator, NFV separates the virtual function logic from the hardware that executes it, whereas *NAaaS* improves the efficiency of this separation.

Hence, *NAaaS* is an enabler not only for NFV, but also for the emerging Network Applications (NetApps) that offer vertical-specific services directly from the network infrastructure (see Section II-B), as well as for the emerging disaggregation trend of the Radio Access Network (RAN) [39].

This survey considers the cloud integration of both *hardware-based* and *software-based* network acceleration technologies. Peccerillo et al. [13] extensively review the hardware acceleration options available today, not limited to networking but including any aspect of computing and communication. The paper recognizes that these devices can be much more effective than general-purpose processors to

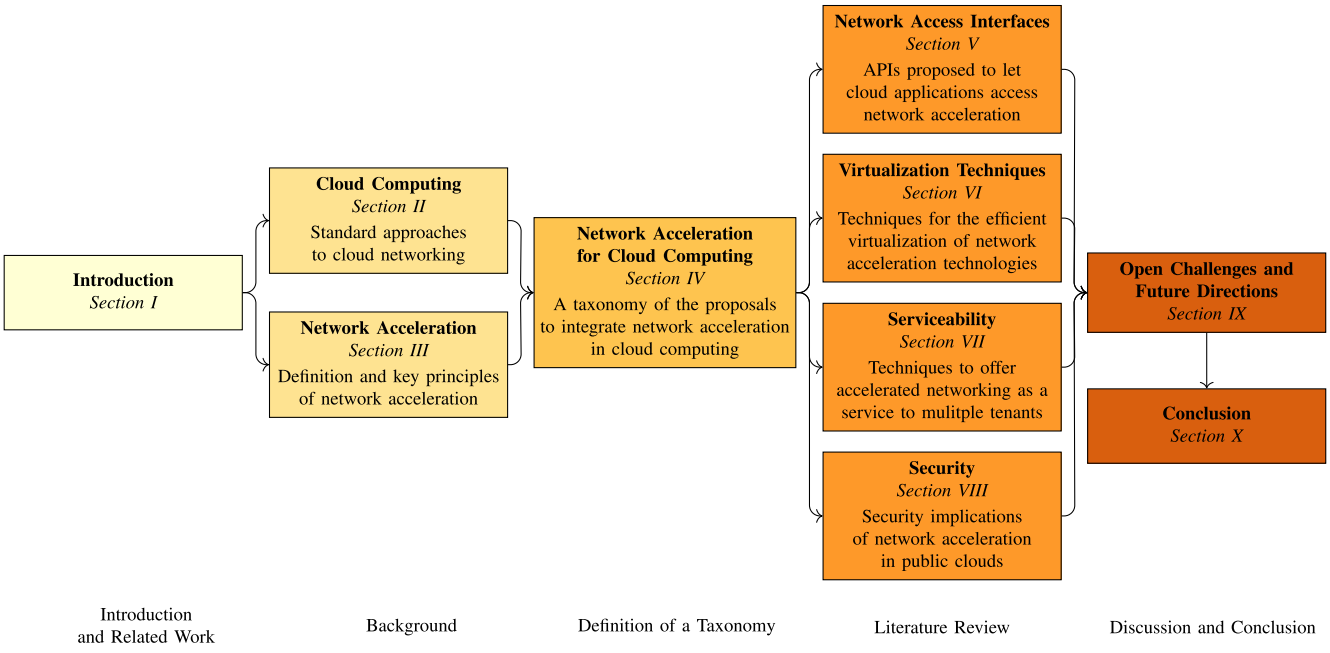


Fig. 2. Overview of the survey structure.

satisfy the globally growing demand for computing power and performance but also that this specialization comes with an intrinsic heterogeneity problem. Given the lack of proper virtualization support, the possible cloud integration of these devices is described as an open research challenge. In this paper, we are interested in how cloud platforms can offer access to those devices as a service to customers.

Similarly, Freitas et al. [14] review software-based acceleration technologies for fast network packet processing, including XDP and DPDK. After a careful analysis of the current bottlenecks of the Linux kernel, the paper reviews how different in-kernel and kernel-bypassing approaches can improve end-host packet processing performance, yielding to different advantages and trade-offs. The survey also includes a brief discussion about the virtualization of these technologies, but it only considers a few examples of I/O virtualization (see Section VI-A). On the contrary, our paper focuses specifically on the cloud integration of these technologies and carefully analyzes all the several implications of the approaches proposed in the literature on the design of cloud infrastructures.

Sharing the same goal of this survey, the works by Hong et al. [15] and Bobda et al. [16] investigate the possibility of enhancing cloud platforms with GPU and FPGA hardware respectively. Hong et al. survey the GPU virtualization techniques and their scheduling methods, focusing on the performance and fairness issues that might arise between multiple tenants. However, other aspects like security, serviceability, and observability are not extensively covered. Bobda et al. systematically review the existing contributions to provide FPGA acceleration in cloud platforms, considering various deployment options and their implications on programming interfaces, virtualization, sharing, and security.

Therefore, both these surveys focus on the specific requirements of a single kind of device (GPU, FPGA) and consider, to a different extent, the literature contributions about their integration in cloud infrastructures mainly to accelerate *compute-intensive* workloads. On the contrary, this work reviews the various aspects of a cloud infrastructure that must evolve to include *network* acceleration technologies regardless of the availability of a specific kind of hardware. We consider RDMA as our reference communication model for *hardware-based* acceleration: as Section III explains, this model can be implemented through a wide range of different devices.

Overall, to the best of our knowledge, this is the first work to comprehensively review the state of the art of *NAaaS* in cloud infrastructures. The novelty of our approach lies in three key factors. First, we do not make assumptions about applications business logic: users should be able to leverage acceleration for general-purpose networking. Second, we do not tailor our review to a specific acceleration technology or hardware device, but move from the consideration that both software-based (e.g., XDP, DPDK) and hardware-based (e.g., RDMA) options share common design principles. Finally, not only do we cover the virtualization techniques that make *NAaaS* solutions feasible, but we also discuss the aspects related to their practical deployment in cloud infrastructures (programming interfaces, serviceability, security) and the challenges associated with the interplay among several stakeholders, including in particular cloud providers and hardware manufacturers.

B. Structure of the Survey

This survey follows a step-by-step structure, as Fig. 2 illustrates. In this Section we introduced the scope of our survey and discussed its novelty in comparison with previous work. In the remainder of the paper, we first present an

overview of the main features and properties of cloud computing platforms (Section II) and of accelerated networking (Section III). In that background part, we stress the fundamental dichotomy between the virtualization paradigm and the kernel-bypassing, accelerated approaches. Secondly, we propose a taxonomy of the most important aspects of cloud networking impacted by the introduction of network acceleration techniques (Section IV). Then, we survey the scientific literature in the field of computer systems about the support to network acceleration in cloud computing, by systematically organizing the relevant papers in the proposed taxonomy (Sections V–VIII). Finally, Section IX summarizes the lessons learned and identifies the most compelling open challenges and the future research directions that need to be addressed toward the availability of *NAaaS*. Section X concludes the work.

II. Cloud Computing

This Section briefly introduces the concept of cloud computing from a networking perspective. In particular, we define four aspects of cloud networking that according to the scientific literature are more impacted by the integration of network acceleration: network access interfaces, virtualization techniques, serviceability, and security. We review the current standard approaches to handle each of these aspects, thus offering the necessary background for readers to understand the motivation and the technical feasibility of the solutions reviewed in the remainder of the paper. We also consider the growing decentralization trend of cloud infrastructures and briefly motivate why the approaches surveyed in this paper are even more compelling in these emerging scenarios.

A. Introduction to Cloud Computing

Since its definition, the cloud computing paradigm gained wide popularity in virtually any economical and social sector. The key reason of that success resides in the availability of computation, networking, and storage resources *as a service*, accessible anytime and anywhere, so that companies in any economical sector no longer need to buy and maintain their own on-premise IT infrastructure. Cloud providers transparently manage the physical infrastructure, billing users for their real resource usage, e.g., charging per time unit, number of requests to a service, or amount of transferred data (*pay-per-use* model). Thus, companies can elastically scale resources based on the actual demand, saving upfront costs and avoiding to pay for idle machines under low traffic, but still being able to respond to peaks in demand. Depending on the agreement between users and providers (Service Level Agreement, SLA), different kinds of cloud resource offerings are possible: a widely popular classification distinguishes between Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) depending on the user visibility of the underlying computing resources [40]. Although this categorization no longer captures the whole spectrum of available offerings [41], it is still considered an important reference.

The focus of this paper is not on a specific cloud model but rather on the mechanisms for users to access accelerated networking, independently of their degree of visibility on the network and whether the network is managed by users or by providers on their behalf. However, we assume that user applications execute either within VMs or containers.

A VM is a deployment technique based on a software-based replication or emulation of actual hardware architectures, enforced by a program called *hypervisor*. A container is a different technique based on the isolation of a user process on a shared OS kernel. Both VMs and containers are used by cloud providers for tenant isolation: users get access to the machine resources, their operating systems (in case of VMs) and applications, whereas the provider keeps control on the underlying physical cloud infrastructure. This distinction is not strict: approaches commonly used in one can be seamlessly applied in the other, for instance, leveraging system call filtering alongside containers or employing sandboxing and user namespaces in virtual machines [42].

Regardless of the specific model, *resource virtualization* is a key pillar of the cloud computing model, as it decouples the users perspective of working on dedicated resources from the provider physical view, and we claim that virtualization plays a crucial role in the dynamicity that characterizes any cloud offering. For example, it enables the elastic scaling of resources in response to the current load of users, saving them significant costs compared to on-premise approaches. Virtualization also enables *multi-tenancy*: the same physical resources can be allocated to different users, even belonging to different organizations. Because providers can optimize the use of their equipment, they can also offer a cheaper service: indeed, multi-tenant clouds generally represent the most economically appealing solution for most enterprises.

From a networking perspective, virtualization is critical to support the communication patterns of modern cloud applications. According to the *microservice* architecture [43], cloud users tend to disaggregate applications across separate business functions, which reciprocally communicate. To support this pattern, cloud providers must enforce two forms of virtualization: not only an efficient network access (*I/O virtualization*), but also virtual private overlay networks among machines (*network virtualization*). There are multiple technical challenges associated with those two kinds of virtualization: which *interface* customers should use to access the network; which *virtualization techniques* allow the same tenants to efficiently while preserving their isolation; how network properties can be flexibly adapted to meet different SLAs (*serviceability*); how the *security* of network operations on a shared infrastructure can be guaranteed.

Although these aspects are certainly not the only concerns for cloud providers, according to the scientific literature they are the most impacted by the integration of hardware-based network acceleration in cloud platforms, thus forcing providers to find novel solutions to retain the existing features of cloud networking while also accommodating network acceleration technologies. To help the reader understand the motivation for those novel solutions, in this Section we provide an overview of the current standard approaches to these fundamental cloud networking aspects (network access interfaces, virtualization techniques, serviceability, security).

Then, Section IV will refer to these aspects for the introduction of innovative solutions in support to *NAaaS* for faster cloud networking.

B. An Emerging Trend: The Cloud Continuum

The success of the IoT concept and its widespread adoption across various application domains are driving an evolution in the cloud computing paradigm. With the pervasive availability of connected devices, there is an increasing demand for applications to consume, analyze, and generate diverse data from a variety of sources under tight time constraints. Centralized clouds can only partially meet these demands and new decentralized computing infrastructures, able to host cloud applications outside data centers, have started to appear in recent years, improving aspects such as response time and bandwidth use for performance-critical services [7].

In the telecommunication ecosystem, the paradigm of Multi-access Edge Computing (MEC) has been developed to answer this trend. MEC envisions that operators leverage resources physically co-located with their equipment to host new services that analyze, process, and store the data in close proximity to mobile users [44], [45]. The recent push toward the disaggregation and virtualization of the Radio Access Network (RAN) answers to the need of achieve additional flexibility and scalability to host new network services and requires the capabilities to efficiently execute performance-critical virtualized applications (e.g., xApps, rApps) [39].

More broadly, the entire design of next-generation communication networks (5G and beyond) is oriented to the support of high-throughput and low-latency traffic processing, with the goal to enable a wide range of performance-sensitive applications (e.g., augmented reality, robotics, autonomous transportation, etc.) to run on the operator networks rather than on remote cloud data centers. This trend is embodied by the concept of Network Applications (NetApps), which allow external users to design vertical-specific services that use network capabilities via APIs [46], [47].

In the Information Technology ecosystem, a similar concept, Fog Computing, was proposed to move compute-intensive services close to data sources and allow them to meet key performance targets. Factory-local small data centers, modern power grids, smart homes, autonomous vehicles, etc. are all potentially capable of hosting low-latency services [48], [49], [50].

Combining the ability to run performance-sensitive, localized applications both at the edge and within the telco infrastructure with the high capacity from the cloud, the *Cloud Continuum* has emerged as a paradigm that can support the heterogeneous requirements of small and large applications through multiple layers of a computational infrastructure that combines resources from the edge of the network as well as from the cloud [51]. The resulting model is a composition of edge, fog, and cloud layer designed to support applications with heterogeneous Quality of Service (QoS) requirements, including performance-critical services.

To overcome the intrinsic resource heterogeneity of these layers, the Cloud Continuum paradigm relies on a *layer of virtualization* that, like in cloud data centers, decouples physical resources from the application code. The resulting computing model is thus a continuum of virtualized resources offered as a service that enables the hosting of application components across the different layers. Across the continuum, providers can operate according to a cloud-like model, for example by assigning slices of the resources to different tenants, by guaranteeing isolation, and by distributing the workload at all levels of the infrastructure [5], [51].

In this context, the need to integrate network acceleration technologies is perhaps even more compelling than in cloud data centers. In the telco ecosystem, the literature on NFV (see Section I-A) already explored the adoption of software and hardware acceleration technologies to reduce the virtualization overhead [10], [52]. However, as we previously discussed, this integration is not considered an option available as a service to general-purpose applications. Unfortunately, the majority of the contributions that target this integration challenge is focused on centralized data centers. In Section IX-C, we describe the extension of *NAaaS* approaches to next-generation network infrastructures, as well as to the broader cloud continuum, as a future research direction.

C. Network Access Interfaces

A desirable property for cloud platforms is that users can deploy unmodified application binaries within the virtualized environments, thus taking advantage of virtualization without any change to their existing applications. At the same time, cloud developers should ideally create new applications without the burden of learning and understanding new frameworks, programming languages, or interfaces, as they can continue to use the same tools they are familiar with.

Such flexibility usually stems from the availability of *virtual* network interfaces within VMs and containers, which cloud users may access by using standard APIs: in particular, cloud applications generally use the standard and ubiquitous *POSIX Socket API* to communicate over the network. We stress here this specific aspect because the network acceleration technologies discussed in this survey natively expose a different interface, and cloud providers that integrate acceleration devices in their platforms will trade the portability and ease of programming of the former with the efficiency of the latter (see Section V).

D. Virtualization Techniques

Resource virtualization is a fundamental principle of the cloud computing paradigm. From a networking perspective, we already distinguished between two kinds of virtualization, *I/O virtualization* and *network virtualization*, according to previous literature on this topic [53], [54]. Although those two aspects are closely intertwined, in the former case we refer to the mechanisms to enable VMs or containers running on a shared physical host to access an external network. In the latter case, we consider the techniques to create virtual private overlay networks among a set of VMs or containers belonging to the same users or tenants. Both these aspects are crucial for

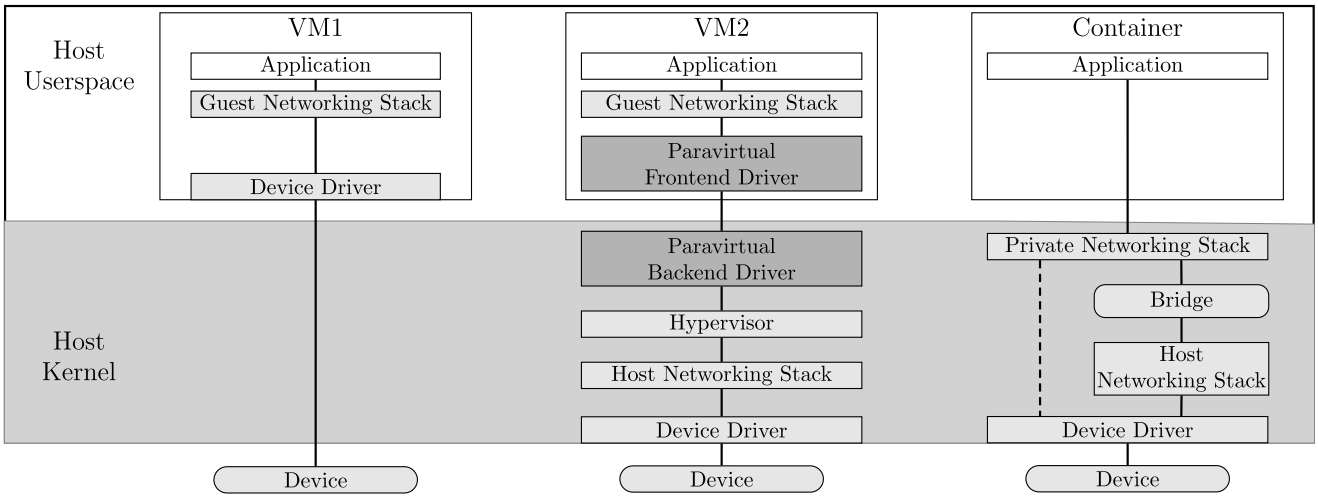


Fig. 3. The two prominent techniques for I/O virtualization for VM: direct device assignment (left) and paravirtualization (center). On the right, two modes for container networking: host (dashed line) and overlay (solid line).

cloud networking as they influence the performance, flexibility, and isolation properties of communication.

Furthermore, as we discuss in the remainder of the survey, the most important challenges concerning the introduction of network acceleration technologies in cloud platforms will emerge in the context of *I/O virtualization* and *network virtualization*. In the following, we briefly discuss the standard approaches typically used by cloud providers to enforce those two forms of virtualization [8], [54], [55], [56], [57], [58], [59].

1) I/O Virtualization: A critical challenge for applications running in virtual environments is to efficiently access I/O devices, especially when network performance is a critical concern. The techniques to obtain an efficient end-host network performance differ for VMs and containers. For VMs, the most prominent approaches are direct device assignment and paravirtualization, represented in Fig. 3 left and center.

Direct device assignment reserves a device instance exclusively to a VM or container (*passthrough*), so each virtual environment (VM or container) requires a distinct physical network adapter. To mitigate this heavy scalability limit, recent devices support a form of *hardware-assisted virtualization* called Single Root IO Virtualization (SR-IOV [60]) that makes them appear as multiple separate devices called Virtual Functions (VFs). Each VF can be assigned to different VMs as if it were a distinct device. Either way, direct device assignment allows to exclude the hypervisor from the network critical path: virtualized applications can access the network as if they were physical hosts, thus achieving the best network performance. However, this technique tightly couples network devices and virtual environments, strongly limiting the inherent flexibility of virtualization: for example, live migration becomes impossible to support, because the hypervisor cannot create a snapshot of the network state.

The paravirtualization technique, instead, splits the device driver into a *frontend driver*, located in the guest OS of a VM, and a *backend driver* on the host (Fig. 3), where those two drivers exchange commands through a dedicated communication channel. This separation lets the hypervisor in full control over the network control and data planes, thus providing a high degree of flexibility: because traffic is mediated by software, it can be easily controlled. However, this also introduces overhead on data path operations, especially when crossing the guest/host boundaries. The virtio [55] framework is the *de facto* standard tool for paravirtualization, and it allows the hypervisor to expose paravirtualized devices to the guests. To mitigate the performance overhead introduced by paravirtualization, virtio clearly separates the data plane, which handles the actual network traffic between the host and the guest, and the control plane, which allows to exchange control messages about the data plane. The data plane is implemented as a set of shared memory areas, called *virtqueues*, between the frontend driver on the guest and the backend driver on the host. Those memory regions are managed as couples of ring buffers holding the network data to be received and transmitted, similarly to the actual queues of physical network devices. Each virtual device can have zero or more queues associated, with the limitation that each queue must be associated with a distinct vCPU. Conversely, the control plane consists of a notification mechanism used between the frontend and the backend driver to discover and signal new data in the queues. For network devices, that notification mechanism is implemented as a direct inter-process communication channel between the two drivers.

The paravirtualization technique is popular for VMs, but it introduces an overhead that is often unacceptable in the case of containers. Indeed, containers are considered a lightweight form of virtualization compared to VMs, as they let applications execute directly on the host operating systems [42]. Various mechanisms are used to isolate containerized applications from the host; from the networking perspective, each container is generally assigned a *network namespace*, which includes a separate instance of the kernel networking stack (Fig. 3 right). Hence, it is possible to create interfaces directly in that namespace, without resorting to virtio, to assign arbitrary IP addresses, and to use the same device drivers installed in the host. In *host* mode, data are forwarded directly to the device driver, so the network configuration is the same

as bare-metal applications (dashed line in Fig. 3). Otherwise, data can be directed by using a software-based datapath to a software bridge (*overlay* mode) and then forwarded to the host networking stack for the actual network access (solid line).

2) Network Virtualization: In the case of I/O direct device assignment for VMs (or *host* networking mode for containers), the traffic of a virtualized application appears on the network as the traffic of a bare-metal application on a physical host. This option might make sense in a dedicated infrastructure, such as a private cloud or on-premise infrastructure, but is not suitable for multi-tenant scenarios, where traffic isolation among tenants is paramount. Instead, providers usually adopt a paravirtualization approach (for VMs) or the *overlay* networking mode for containers, with the goal of segregating user traffic within controlled boundaries (*network virtualization*).

The key technology to enable network virtualization is the *virtual switch*, a software module in the host kernel that acts as packet dispatcher among network interfaces. Once the traffic of a virtualized application reaches the backend driver (in case of VMs, see Fig. 3 center) or exits the network namespace (in case of containers, see Fig. 3 right), the switch forwards it on the physical network. In the cloud, virtual switches represent the first network hop for user applications. Hence, they are the main tool for providers to build logically isolated, virtual private overlay networks to connect user application components, or to implement customer-supplied network spaces.

Importantly, the software flexibility allows cloud providers to offer richer network semantics, by configuring traffic shaping policies and dynamically adapting such configuration to external events, e.g., mutated network conditions, new policies to enforce, or user requests. With software switches on each host, providers can scale such control actions to a high number of servers, while at the same time keeping the actual physical network simple, scalable, and thus very fast.

Important traffic shaping policies include network isolation of VMs and containers via different forms of tunneling, such as VXLAN [61], security, migration, QoS enforcement, and generally all the key *serviceability* aspects that we discuss in Section II-E. To leverage this flexibility as much as possible, major cloud providers design their own software switches [62], although open-source versions are widely available, such as Open Virtual Switch (OVS) [63].

E. Serviceability

Virtualization allows cloud providers to dynamically optimize their resources in the most cost-effective way. For instance, providers can automatically trigger various forms of load balancing and optimizations toward an efficient use of their data center, transparently to the end users; the same mechanisms can be employed to elastically scale user resources according to different SLAs. All these actions are crucial for providers to make their virtualized resources *serviceable*, in the sense of ready to be offered as a service to their customers.

Among several serviceability aspects, some of those related to networking are particularly relevant for this survey, namely *monitoring and logging*, *QoS enforcement*, and *live migration*, as the current techniques to implement them are not compatible with the fundamental principles of accelerated networking. Cloud providers currently control those aspects through software-programmable virtual switches (see Section II-D2), but current approaches to network acceleration provisioning also bypasses those switches, thus forcing providers to recover serviceability through alternative strategies.

In the following, we briefly introduce those three main serviceability aspects, leaving to Section VII the discussion on how they can co-exist with network acceleration technologies.

1) Monitoring and Logging: A key factor for successful cloud operations is a rich set of *observability* tools. By monitoring every aspect of their software and hardware infrastructure, cloud providers can motivate several technical (e.g., control actions) and business (e.g., pricing levels) decisions. Control actions can trigger automatically, thus improving the overall quality and performance of the offered services, and include, for example, tracing the connection behavior of single VM or container, diagnostic any kind of problems, locating performance bottlenecks; all at the scale of a data center.

Overall, the *pay-per-use* business model is enabled by the ability of monitoring fine-grained actions from customers. Monitoring data are also generally exposed to end users in the form of *Key Performance Indicators* (KPIs), so that customers can verify the respect of the SLA negotiated with the provider [64], [65].

In some cases, monitoring data can be logged and kept on a persistent support for the offline analysis of the operations, or the detection of malicious actions. More and more often, logging is also becoming a legal obligation [66], [67] that enables independent auditors to investigate controversies among users and providers.

2) QoS Enforcement: The definition of Quality of Service (QoS) is wide, so here we limit our considerations to networking aspects. Nonetheless, QoS is a crucial concept for cloud providers as they base their business model on the offering of different levels of services: basic resources with limited capabilities are generally cheap for users, whereas advanced features can be purchased with a fee. Beyond economical considerations, QoS management is important also for technical reasons, because on a multi-tenant cloud it ensures that no tenant can exhaust the whole available resources. Concretely, QoS control is enforced through packet quotas, bandwidth caps, and other kinds of limits on network resources, such as the number of links, or of on-demand virtual private networks a user can instantiate [68], [69], [70].

3) Live Migration: Cloud users expect to deploy applications through automated orchestrators that take care of the necessary management, automated scaling, and lifecycle operations of their application components. On their side, providers in multi-tenant clouds are continuously looking for an optimal allocation of user resources on the available infrastructure, by taking into account the user requirements, the SLAs, and also the overall data center status. Since the data

center conditions change over time, because of shifting user workloads, load balancing needs, infrastructure maintenance, etc., both providers and their customers need a dynamic way to scale, re-allocate, and balance their resources (*continuous scaling*).

In this context, *live migration* is a powerful tool for the administration of data centers that allows providers to move running VMs or containers from one physical host to another. With live migration, even critical applications can be moved without service disruption and with controlled performance overhead, thus allowing providers to dynamically balance resource usage and to isolate portions of the infrastructure for maintenance or updates.

Unfortunately, the live migration of network resources is one of the biggest challenges in the whole migration process even when standard networking is adopted. The migration procedure requires to checkpoint and restore the state of all the active communication channels of an application. In turn, the checkpointing procedure requires the availability of specific support mechanisms from major operating systems: for instance, the Linux kernel modified its implementation of TCP to introduce a new state (`TCP_REPAIR`) that helps the migration operations. The same visibility issue affects *accelerated* network channels, as we discuss in Section VII-C.

F. Security

An increasing number of companies completely rely on the cloud to design, deploy, and operate their own services, as well as to manage system and customers data. Customers need to trust cloud providers for high-quality services, in particular for the security and the availability of applications and data. Nowadays, the trust relationship between users and providers is based on a set of security *best practices*, consolidated in one decade of operations in the cloud industry, including security mechanisms at many levels: firewall and hypervisor, cryptographic mechanisms, authentication and access control, traffic isolation, etc. [71], [72]. Even for security-related policies, *dynamic actions* are paramount for the effectiveness of those mechanisms, which are expected to be easy to reconfigure and capable of immediately enforcing new security policies.

At the network level, in particular, providers must guarantee the confidentiality, integrity, and authentication of the traffic between users and external networks as well as on virtual private overlay networks among user machines. Resource access is another critical aspects, especially in public shared infrastructure: hence, provider define mechanisms such as access control lists (ACL), security groups, Firewall as a Service (FWaaS) to ensure only authorized users can access to network resources. When different tenants share physical resources, providers must guarantee the *isolation* of their computations and communication activities from those of other users. For example, network traffic flows from different tenants must not interfere with each other. Even worse, a malicious tenant must not be able to infer which kind of workloads or traffic patterns other tenants are running by just inspecting the performance of its own resources [73], [74], [75]. Finally, another paramount security requirement is attack detection: providers should be able to recognize an attempt to break their system, notifying users if necessary. Recent legal regulation, for example, mandate many forms of auditability and require that providers warn users about possible data breaches [66], [67].

III. Network Acceleration

In this Section we provide the necessary technical background to understand the potential benefits, the challenges, and the trade-offs associated with the integration of network acceleration technologies in cloud infrastructures. In the first part of the Section we define the concept of *network acceleration* and distinguish between *software-based* and *hardware-based* technologies. Rather than delving into complex technical details, we aim at presenting the *common design principles* behind different technologies (XDP, DPDK, RDMA, etc.), which are all based on the fundamental concept of clear separation between the control and the data planes of communication [76], [77]. Then, because a significant majority of the surveyed literature focuses on hardware-based technologies and specifically on the RDMA communication model, we present an extensive background on them.

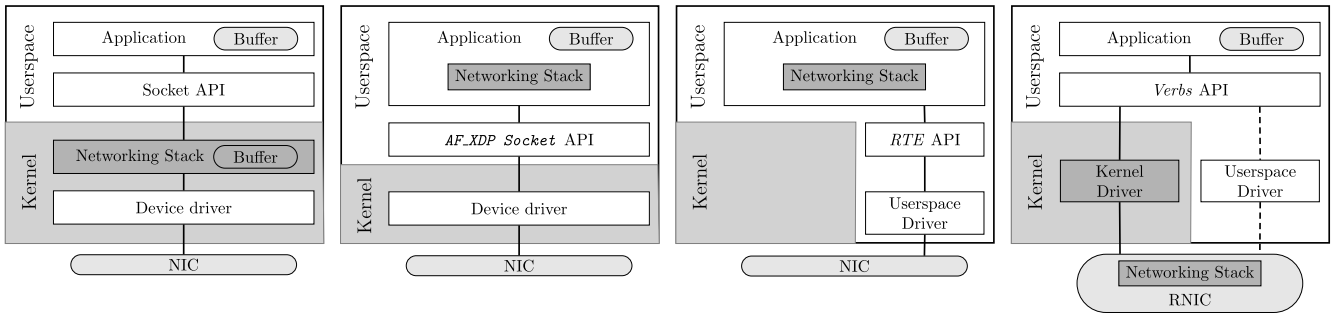
A. Introduction to Network Acceleration

In the context of this survey, the term *network acceleration* refers to the minimization of the performance overhead incurred from the initiation of an application's data transmission request to the execution of the send operation by the hardware NIC. Similarly, it refers to the minimization of the performance overhead from the NIC's reception of a packet to its delivery to the receiver application.

Based on the considerations introduced in Section I about the end of Moore's law, network acceleration technologies tend to improve end-host networking by minimizing the processor intervention on these operations. This idea reflects a shifting balance highlighted by recent research: modern networking links are so fast that to utilize its full potential, developers must clearly separate the *control plane*, which expresses the application logic and thus entails CPU intervention, from the *data plane*, where data must be free to move at the link speed [76], [77]. Thus, applications can leverage the full speed of modern communication links, and data center providers dedicate a bigger portion of their CPUs to user applications.

In particular, it is possible to identify three key *design principles* that modern network acceleration technologies follows to guarantee clear separation of control and data planes:

- *Zero-copy Data Transfers*. Memory copy operations are by far the most significant bottleneck for end-host networking [23], [25]. Ideally, a complete separation between control and data planes would require a model in which any host involved in the communication has a single copy of any given data item at a certain memory location: the network equipment places incoming data at in a designed area (or reads them from it), from which data is never moved. The control plane refers to the items by sharing references to that location. Although maintaining just a single copy of each data item might



(a) Standard kernel-based networking

(b) XDP in-kernel software acceleration

(c) DPDK kernel-bypassing software acceleration

(d) RDMA kernel-bypassing hardware acceleration

Fig. 4. A comparison between (a) the standard kernel-based networking stack vs network acceleration technologies: (b) XDP, (c) DPDK, (d) RDMA. RDMA clearly separates the control path (solid line) from the data path (dashed line).

not be always practically viable for many reasons (isolation, security, etc.), high-performance network protocols should strive to minimize their number.

- *Minimal Context Switching*. Context switches between processes, especially at the boundary between userspace and the kernel, use precious CPU cycles. Hence, it should be avoided performing protocol processing and other data plane actions in the kernel, as that involves passing controls multiple times between user and kernel processes. Ideally, data plane operations should be performed by a single or a small set of processes or threads in userspace, thus also improving data and instruction cache locality, or, even better, completely offloaded to hardware.
- *Asynchronous Processing*. Because control and data plane potentially execute at very different speeds, programming network interfaces and protocols should be ideally designed to be completely asynchronous and lock-free, thus avoiding that control plane stalls data plane, or vice versa. That is practically not always possible, e.g., when parallel processing is performed on multi-core processors, but the more this separation is enforced, the better is network performance.

Although all the network acceleration technologies share these design principles, each implements them in different ways, partially or totally bypassing the standard OS kernel, as Fig. 4 visually summarizes. In this survey we distinguish between two categories of network acceleration technologies: *software-based* and *hardware-based*. In the following, we summarize the main differences of these approaches.

1) Software-Based Technologies: Software-based acceleration technologies improve end-host network performance without requiring any special hardware to be installed. On the one hand, that is an economical advantage as this option does not require additional hardware than the existing general-purpose network equipment. On the other hand, the absence of dedicated hardware means that compute-intensive operations such as protocol processing must be performed by the CPU, thus burning precious processor cores [14].

Currently, the most prominent software-based acceleration technologies are the Linux eXpress Data Path (XDP) and the Data Plane Development Kit (DPDK). XDP [31] is the most conservative option with respect to the standard network stack: packet processing executes within the kernel, thus retaining the existing observability tools and isolation mechanisms but also the user/kernel context switches. XDP enables the execution of user-provided code (*eBPF programs*) for each packet at the lowest layer of the kernel networking stack, located within the driver of network devices. This allows users to forward packets directly to/from user applications, thus bypassing the higher layers (e.g., kernel-based TCP/IP) that are responsible for significant packet processing overhead.

Instead, DPDK [32] is designed to accelerate packet processing through a *kernel-bypassing* approach that removes also the user/kernel context switches. Originally developed to accelerate virtual network functions, DPDK has been increasingly adopted as a form of end-host network acceleration for general-purpose applications. A set of C libraries let users directly interact with the hardware NIC through a userspace version of the network device drivers designed to allow zero-copy data transfers and totally asynchronous packet processing. To minimize latency, usually applications employ one or more threads that continuously check for packets from the applications to be sent on the network and from the network to be sent to the application. This *busy polling* approach is very effective in terms of performance, but it also represents one of the major drawbacks of DPDK because it induces a high CPU and energy consumption. Furthermore, bypassing the kernel also prevents DPDK applications to use standard kernel-based observability tools and isolation mechanisms.

2) Hardware-Based Technologies: Hardware-based acceleration technologies offload one or more computationally intensive tasks to a *network accelerator*, a special-purpose hardware device. In the networking domain, these devices can effectively execute compute-intensive operations, such as protocol processing, much faster than general-purpose processors, by removing the bottleneck of software-based protocol implementations and by allowing application developers to leverage the full potential of modern network links. Also in this case, different approaches can either retain the flexibility and isolation of in-kernel networking [52] or adopt a *kernel-bypassing* solution for the sake of performance (Fig. 4d). The most prominent example in this category is RDMA, which we extensively introduce in the remainder of this Section.

In this survey, we consider the challenges of cloud integration for both software-based and hardware-based acceleration

technologies. As these technologies all follow the same design principles, the approaches surveyed in our literature review (Sections V–VIII) are generally valid for all of them.

Nevertheless, there is a key performance difference among these two categories: by offloading compute-intensive tasks to a specialized device, hardware-based technologies maximize the performance benefits for customers and do not burn processor cores, thus making them the most appealing option for cloud customers. For this reason, the significant majority of the surveyed works focuses on the cloud integration of hardware-based technologies and, in particular, of RDMA. As a consequence, in the remainder of this Section we provide an extensive background on RDMA, which will help the reader understand the technical solutions surveyed by this work.

B. Introduction to RDMA

Network acceleration devices can be significantly heterogeneous in terms of hardware implementation, supported tasks and protocols, interface exposed to developers, and programmability. From a hardware perspective, most devices are built using three main technologies, namely Field Gate Programmable Arrays (FPGAs), Application-Specific Integrated Circuits (ASICs), or Systems-on-a-Chip (SoCs), each with different characteristics and properties. Depending on their goals in terms of product cost and complexity, manufacturers and researchers can decide to embed specific network functions in a network card (e.g., [78], [79]), or to provide them through an additional device (*bump-in-the-wire* approach [80], [81]).

From a software perspective, developers interact with these devices through interfaces that can have different degrees of standardization, expressiveness, ease of programmability, and associated performance overhead, depending on the underlying hardware. An extensive review of these devices and of their interfaces can be found in [13]. We limit our discussion to observing that the heterogeneity of these devices creates a practical concern both for providers and developers: because applications interact directly with the device drivers, different devices might assume different communication abstractions, models, and vendor-specific access interfaces.

Toward a unification, Remote Direct Memory Access (RDMA) has emerged in the last decade as a convenient, cost-effective approach to network acceleration. RDMA is a standard communication model that allows a process to directly access the memory address space of another process on a remote machine [33]. The RDMA specification defines an asynchronous, general-purpose semantics to leverage the features of network accelerators, such as zero-copy communication, but it does not require a specific implementation: any hardware device, including in principle general-purpose CPUs [82], may implement its model. Thus, RDMA introduces a uniform access layer to hardware acceleration.

The popularity of RDMA as a network acceleration model makes it an ideal candidate as a reference technology for hardware-accelerated approaches in our survey. Since its origin in the High Performance Computing (HPC) community [34], RDMA received significant industrial support from hardware manufacturers, which made it available also for general-purpose networking, and became a popular networking technology both in industry and academia [35], [78], [79]. As a consequence, network acceleration devices supporting RDMA are increasingly available as off-the-shelf components, by allowing application processes to communicate directly, bypassing the kernel-level networking stack and achieving zero-copy transfers, significantly higher throughput and lower latency, and lower CPU utilization.

In the following, we provide an introduction to the main aspects of RDMA that serves as the necessary background for the reader to understand the technical solutions proposed by the reviewed literature (Sections V–VIII). We start with an overview of the protocols that implement its communication model through the *de facto* standard RDMA interface (*Verbs*). We then consider the different kinds of RDMA operations, the network primitives, and their semantic, as these aspects are relevant to discuss the available approaches to *NAaaS*. Overall, the RDMA protocols and access interfaces are composite and not compact, and the complexity stems from the need to balance both the requirement of high asynchronicity and the necessity of precise tools for monitoring and quality control.

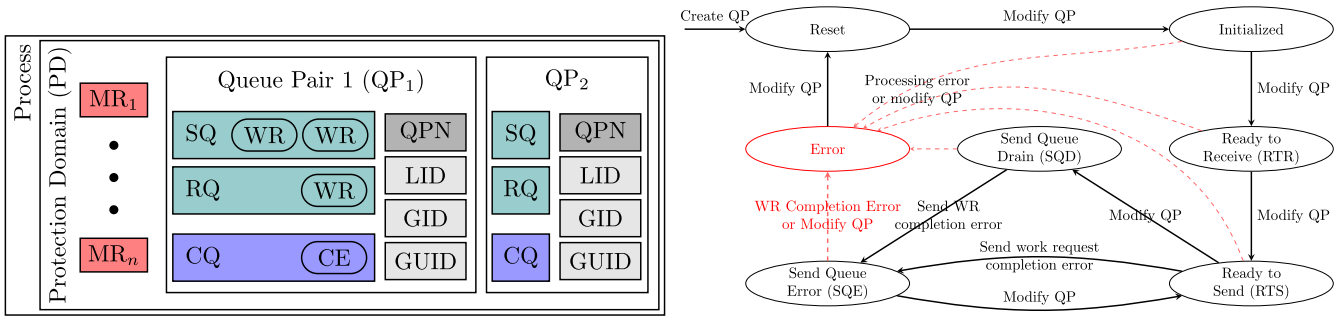
C. Protocols That Implement RDMA

Although the diffusion of RDMA in the cloud community is a relatively recent phenomenon, it has been a well-established communication technology in the HPC community for decades. One reason for this confinement was related to the available technology: for a long time, the only existing implementation of RDMA was InfiniBand [33] that defines a specialized stack of protocols designed to run at high-speed on special-purpose hardware, which includes not only cables, but also special NICs (Host Channel Adapter, HCA) and switches to create a lossless fabric with a credit-based mechanism.

Even though the Infiniband stack offers a lighter transport option than TCP in terms of protocol overhead and CPU consumption, the high cost of the entire solution traditionally limited the adoption of RDMA to HPC deployments. The increased popularity of RDMA outside HPC environments became possible with the definition of other two protocols that implement the RDMA specification, namely RDMA over Converged Ethernet (RoCE), and Internet Wide Area RDMA Protocol (iWARP) [83]. In the following, we briefly introduce their main characteristics.

RoCE [35] enables RDMA over ordinary Ethernet networks. The latest version, RoCEv2, also enables layer 4 (UDP) encapsulation, whereas higher-layer headers are InfiniBand headers. This design enables RDMA over standard Ethernet infrastructure, because only the NICs should be upgraded to support RDMA on existing deployments. Because that choice drastically reduces the upfront investments required for new equipment that supports RDMA, even cloud providers have started to consider this option for their infrastructure, thus effectively paving the way for the adoption of this technology at a much larger scale [84], [85].

Differently, iWARP is a suite of protocols that layers RDMA over TCP/IP, based on a complex mix of protocols to enable



(a) Objects defined by the OFA Verbs API.

(b) Queue Pair State Machine

Fig. 5. Main concepts of RDMA networking.

zero-copy data transmission using the traditional networking stack [86], [87]. The actual acceleration consists in offloading the TCP/IP protocol processing to the hardware NIC, thus effectively bypassing the OS kernel but also preserving the TCP semantic to deal with packet losses without requiring additional support. Nevertheless, the choice to rely on the unmodified TCP stack has led to performance and capability limitations compared to Infiniband and RoCE. On the one hand, the complexity of TCP protocol, even when performed by specialized hardware, is quite expensive compared to either UDP or to the Infiniband transport protocol. On the other hand, as TCP is connection-based, iWARP only supports point-to-point connected transport service, limiting the supported communication patterns. Therefore, the vast majority of data center operators today prefers the use of RoCEv2 over iWARP.

D. Communication Interface: RDMA Verbs

In traditional networking, applications require network resources and operations from the operating system through the standard POSIX socket API. In contrast, RDMA programs interact with the operating system only to establish (or to close) a communication channel (see Fig. 4). Then, applications can directly interact with each other through a standard asynchronous API called *RDMA Verbs*, defined by the InfiniBand specification [33].

Properly, *Verbs* are just semantic descriptions of the behavior any RDMA protocol must provide: there are no additional details, so that implementations are free to define their own API syntax for functions, structures, types, etc. To prevent the proliferation of incompatible definitions, the OpenFabrics Alliance (OFA) has created a vendor-independent C API, called *OFA Verbs*, to access the three protocol stacks we have introduced (InfiniBand, RoCE, and iWARP) through the same interface [88]. Although *Verbs* are nowadays the *de facto* standard interface for RDMA, they are also a very low-level interface and developers must write a large amount of code with tens of parameters to perform simple data transfers. Such difficulty represents a significant obstacle to a widespread adoption of RDMA, especially in cloud environments where developers are used to simple, user-friendly interfaces.

To facilitate the reader understand the proposals for easier-to-use and higher-level interfaces (see Section V), in the following we summarize the most important concepts of RDMA networking. A detailed description can be found in the Infiniband specification [33].

E. Verbs Objects

The Verbs API require the manipulation of several objects, as illustrated in Fig. 5(a). In the following, we describe three of them that are crucial for the *NAaaS* approaches we review in the next Sections: Queue Pairs (QPs), Memory Regions (MRs) and Protection Domains (PDs).

A Queue Pair represents a communication endpoint and is composed by a Send Queue (SQ) and a Receive Queue (RQ). The asynchronous RDMA operations are expressed as *Work Requests* (WR) that users push to one of these queues. For instance, a user sends data by posting a Send WR to the SQ. To enable the zero-copy semantic, data to be sent and received is placed in one or more memory areas called Memory Regions (MRs). These are shared areas between the applications and the hardware NIC: this way, instead of copying data from the application memory to and from the OS, applications and the NIC only exchange WR descriptors, which internally contain a reference to data in the shared area. Finally, as a form of access control, QPs and MRs are grouped into Protection Domains (PDs): they have visibility of each other only if they belong to the same PD. In the following, we describe the most important features of these three objects.

1) Memory Regions: To allow *zero-copy* transfers, applications must first perform the *memory registration* of one or more MRs. This operation grants to the NIC the permission to directly access (DMA) those MRs during communication, avoiding data copies. The registration operation returns a couple of *memory keys* that are required during the communication phase to access local (*lkey*) and remote (*rkey*) MRs.

2) Queue Pairs: Fig. 5(a) includes a schematic representation of two Queue Pairs. There are three considerations about QPs that are relevant for this survey. First, according to the Verbs design, users must be able to asynchronously detect the outcome of the WR they submit, such as the availability of new incoming data that match a Receive WR. A third queue, called *Completion Queue* (CQ), is associated to each QP to let user discover these events. All these queues are physically located on the on-board memory of the RNIC.

Second, QPs are host local resources, uniquely represented by an integer called *Queue Pair Number* (QPN) and by three other identifiers: the GUID, which uniquely identifies the RNIC and whose function is equivalent to the MAC address on Ethernet networks; the GID, a routable address used to identify a port on a network adapter (equivalent of an IP address); the LID, a non-routable address (equivalent to TCP/UDP ports). These number will play a significant role for live application migration (see Section VII-C).

Finally, a QP is implemented as a state machine, where each state corresponds to a different phase of the RDMA communication. Fig. 5(b) shows the corresponding state diagram. Immediately after the creation, a QP is in *Reset* state, and it should traverse a number of intermediate steps to reach the *Ready To Send* state, which finally enables it to send and receive packets. In case of an error, the QP moves to an *Error* state. It is relevant to note that, as QP information is stored on the RNIC, kernel-level applications have no visibility on it. As we discuss in Section VII, that makes it hard to implement key cloud features such as monitoring or live migration.

3) Protection Domains: The Infiniband specification defines a mechanism to regulate the access to Queue Pairs and the other RDMA resources, the *Protection Domain* (PD). To create a Queue Pair or a Memory Region, it is necessary to associate it to a PD. Then, objects within the same PD can interact with each other (e.g., QPs can be associated with MRs), but not with objects belonging to other PDs. We will extensively discuss the current limitation of RDMA access control mechanisms in Section VIII-B.

F. Communication Channels: Modes and Establishment

To establish an RDMA communication channel, users can choose different *service types*, also referred as *transport modes*. The InfiniBand specification defines three service types: *Reliable Connection* (RC), *Unreliable Connection* (UC), and *Unreliable Datagram* (UD). The Reliable Connection mode offers a zero-copy, unicast, connection-oriented communication with in-order delivery. In contrast, the Unreliable Datagram mode allows each QP to exchange single-packet messages with any other UD QP, but without ordering or delivery guarantees: e.g., undelivered packets could be dropped by the receiver. This semantic makes it possible for a sender to transmit multicast messages (one-to-many). Finally, the Unreliable Connection mode builds a point-to-point channel, but without the ordering, delivery, and error detection guarantees provided by RC.

If RC mode is used, the two peer endpoints must exchange the following information before being able to communicate: the memory keys of each registered MR, the QPN, and the three addresses (GUID, GID, LID) previously described. This exchange can happen through an out-of-band channel (e.g., a regular TCP connection) or through the *Communication Manager*, a transport-neutral interface [89].

G. Semantic of Communication Operations

Once a communication channel is created, the endpoints can communicate at line-rate speed using the data-plane operations. RDMA Verbs define three possible semantic for communication operations, *two-sided*, *one-sided*, or *atomic*, which differ for different degrees of involvement of the remote CPU in the transfer, as we detail below. Alongside the main communication channel, Verbs also provide an additional channel of limited bandwidth (generally 32 bits) in the form of an *immediate* value, which is guaranteed to be delivered to the peer after the associated RDMA operation completes.

1) Two-Sided Operations: The two-sided operations are `send` and `receive`. Fig. 6 shows an example of their use: when a process *A* wishes to send data to another process *B*, *B* must previously have posted a *Receive Work Request* into the receive queue, associating a free memory buffer to it (①). Thus, *A* can post a *Send Work Request* (②), requiring the NIC to perform a zero-copy transfer into the memory area designated by *B* (③). As each transfer finishes, a completion record becomes available in the completion queue on both ends (④ and ⑤), that the application can poll to check for completion. The application also needs to ensure that there are sufficient *Receive Requests* in the *Receive Queue* to match all incoming *Send Requests*, as the latter will be dropped otherwise. Overall, this operation mode requires the receiver to synchronize with the sender to receive data and thus represents a compromise between performance and ease of programming.

2) One-Sided Operations: The one-sided `write` and `read` operations allow a process on one machine to asynchronously access a region of application memory on a remote node, without that node being aware of it and thus without involvement of the remote CPU. With reference to Fig. 6, only some steps are performed. In the case of a `read` request, the issuing application specifies the remote address to read from and a local buffer to store the data into (②). Then, the network adapter asynchronously performs the remote read (③), populates the local buffer, and posts a completion event to the local completion queue (⑤). The `write` operation works in the same way, except that data is transferred from a local buffer to the remote memory.

In summary, the remote CPU is never involved in one-sided operations and for this reason they have been described as a *double-edge sword* [90]. On the one hand, they represent the most powerful RDMA communication mode as they limit the processor intervention, and also, considering that the network cables are full-duplex, they can use the entire available bandwidth to complete the transfer. On the other hand, as we extensively discuss in Section VIII, they also have relevant drawbacks, ranging from a more difficult programmability to a notable series of associated security threads.

3) Atomic Operations: The *atomic* operations are a particular form of one-sided communication that includes remote memory synchronization such as `compare-and-swap` and `fetch-and-add`. These operations act on 64-bits values, one residing locally to the initiator host and one remotely, executed by the remote NIC which guarantees their atomicity.

Overall, the complexity of RDMA embodies the inherent difficulties of working directly with hardware accelerators. Realizing the potential performance benefits requires a significant investment of design and implementation efforts to customize systems according to the specific characteristics of

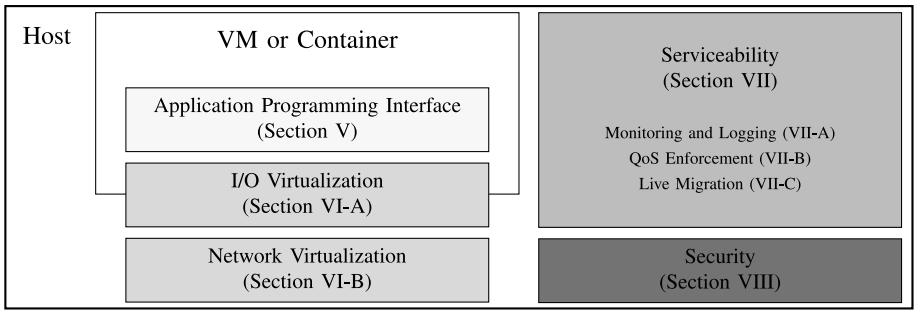


Fig. 7. A logical view of the four key dimensions of *accelerated* cloud networking where trade-offs between performance and flexibility clearly emerge: network API, virtualization techniques, serviceability, and security.

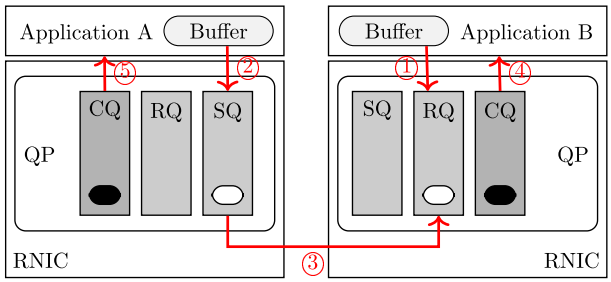


Fig. 6. Scheme of *two-sided* data transmission using RDMA. Asynchronous communication is a key feature of network acceleration technologies.

these technologies. The integration of network acceleration technologies in cloud infrastructures as *NAaaS* would make them easily available, but it also presents several challenges that we discuss in detail in the next Sections.

IV. Network Acceleration for Cloud Computing

The availability of network acceleration technologies in cloud platforms represents an appealing perspective for cloud users to meet the increasingly demanding requirements of the emerging *interactive, data-intensive* applications introduced in Section I. However, cloud platforms are founded on the principle of *resource virtualization*: a software interposition layer mediates all network operations, introducing a performance overhead that used to be acceptable until recent years. As network speeds increasingly outperform processors speeds, the cost of such interposition layer becomes a significant performance bottleneck, unacceptable for *accelerated* networking.

To face the soaring demand for network acceleration technologies, major providers have adopted the short-term strategy of removing such virtualization layer and rent *bare-metal instances* equipped with acceleration options [18], [19], [20], [91], [92], [93], [94], [95], [96]. However, enforcing tenant isolation through dedicated resources has several disadvantages. First, it is neither cost-effective for providers nor for the end customers. Second, the flexibility of bare-metal instances is minimal, thus preventing typical cloud features, such as live migration or the definition of virtual private overlay networks, from being available.

In this survey, we consider a recent trend in the scientific literature that proposes longer-term strategies for the full integration of network acceleration technologies into cloud platforms, toward the ultimate goal of enabling *Network Acceleration as a Service*. *NAaaS* is indeed an increasingly compelling option for applications to access modern high-performance network links without sacrificing the advantages of running in the cloud. To this end, we have considered the papers published at major conferences and journals in the field of computer systems and have systematically organized them into a general framework to provide a comprehensive and complete snapshot of the state of the art on the availability of *NAaaS* in cloud data centers. Our contribution is motivated by the observation that there is still no complete solution to the overall problem; we instead observe several proposals that focus on one or some specific aspects of this general problem.

In contrast with this fragmentation, we believe that a standardization effort would be fundamental for the evolution of both the network acceleration and the cloud computing ecosystems: as the demand for integrated solutions surges, and experimental forms of accelerated cloud networking are prototyped to keep up with them, the lack of a cohesive perspective poses the risks of hindering any integration initiative. After a careful literature review, we explore here the potential for combining together the existing approaches and identify the key research challenges to be addressed.

We begin our discussion by noting that the literature on this topic focuses on the trade-off between the performance acceleration granted by network acceleration technologies and the flexibility typically associated with cloud platforms. In the analysis of the solutions proposed in the literature, we observe that such trade-off emerges in connection with four key dimensions of cloud networking, which we have introduced in Section II and represented in Fig. 7. Here, we summarize the main challenges associated with each of those dimensions.

1) *Network Access Interface*. The interface to access an acceleration technology can have a great impact on performance, application portability, and ease of programming. For instance, a zero-copy, asynchronous interface like RDMA *Verbs* might optimize performance, but breaks compatibility with existing applications. Conversely, the standard POSIX Socket API is familiar to developers, but it introduces data copies that significantly harm performance at such high link speeds.

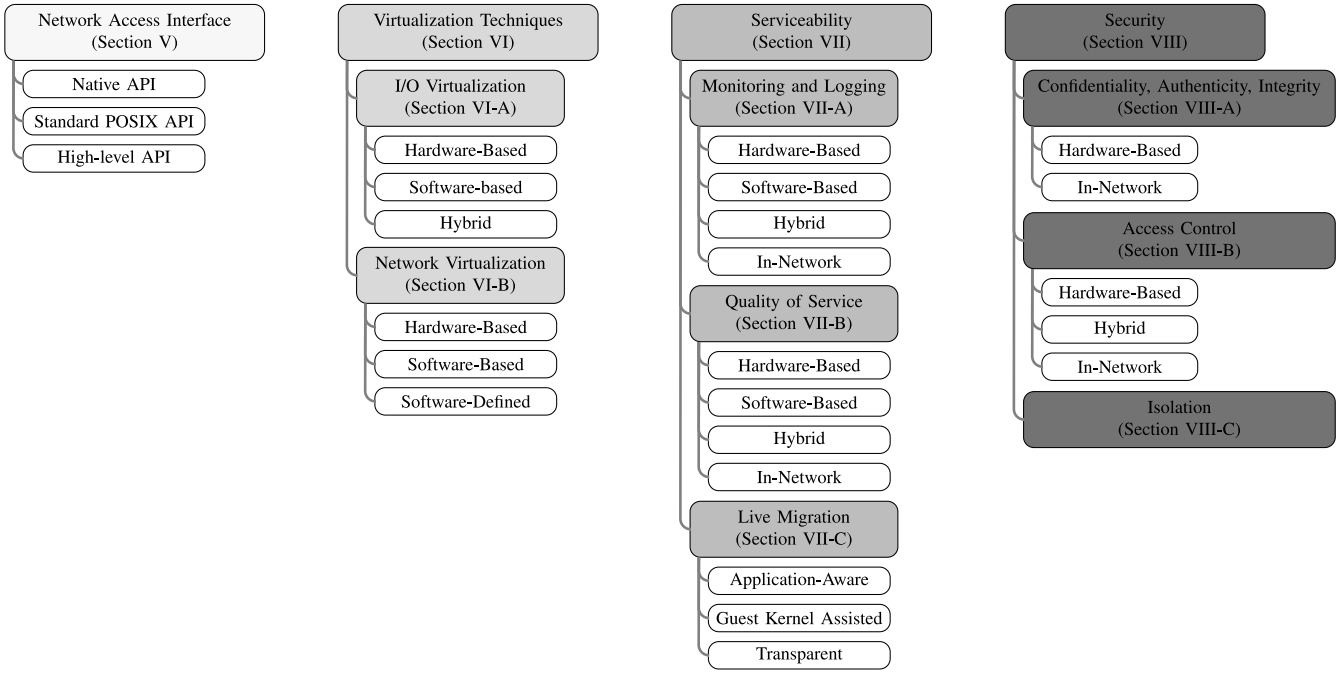


Fig. 8. Our proposed taxonomy of the literature aiming at enabling *Network Acceleration as a Service* in cloud infrastructures.

2) *Virtualization techniques*. A cloud platform that lets cloud users efficiently access an acceleration technology requires the definition of efficient virtual network interfaces, of mechanisms to exchange data between VMs/containers and physical hosts (*I/O virtualization*), and of *accelerated* virtual private overlay networks (*network virtualization*). Building such a solution in software would preserve flexibility, but also make network operations slow due to processor interposition; conversely, hardware-based approaches potentially preserve performance, but they also might subtract the full control on the network operations from cloud providers.
3) *Serviceability*. To expose network acceleration as a commodity, providers must flexibly control the way users access shared acceleration technologies, as they currently do for the rest of their infrastructure. However, the standard resource management mechanisms are typically implemented in-kernel, and thus not applicable to kernel-bypassing technologies. Finding a solution to this problem is particularly complex for three aspects: monitoring and logging, QoS policy enforcement, and the live migration of VMs and containers.
4) *Security*. The trust relationship between cloud customers and providers is based on a set of security best practices at many levels, as we commented in Section II-F. Some of these practices are mandated by law, and today are essential for the operation of providers. At the same time, many of these practices are time and resource consuming, to the point that some of them are already partially offloaded to hardware (e.g., cryptographic operations). As the existing mechanisms to provide security in the cloud become inadequate to preserve the increasingly faster network link speed, new ones must be defined for a successful integration of network acceleration technologies in existing cloud platforms.

As each dimension of cloud networking introduces specific challenges related to the trade-off between performance and flexibility, we organize the various solutions proposed by the literature on this topic according to the taxonomy in Fig. 8. For each dimension, we distinguish four coarse-grained categories, based on the general consideration that, when applicable, the reviewed solutions tend to explore four main points in the space between performance and flexibility: *hardware-based*, *software-based*, *hybrid*, and *in-network* approaches. These four approaches are orthogonal to the two kinds of network acceleration (software and hardware) we discussed in Section III-A and, generally, apply to both categories, as we discuss in each literature review Section.

In particular, *software-based* approaches focus on the realization of the network control infrastructure mainly in software, favoring flexibility over performance. On the contrary, *hardware-based* approaches aim at partially offloading the control infrastructure to hardware devices, thus giving priority to performance over flexibility. *Hybrid solutions* try to combine the best features of the two previous approaches, leveraging the fundamental principle of separation between control and data plane that underlies accelerated networking. Finally, *in-network* approaches partially or totally move control functions to the network infrastructure, such as to Top-of-Rack (ToR) switches. That allows them to recover the control features that are bypassed by acceleration technologies, preserving a moderate degree of flexibility with minimal performance overhead.

These four approaches (*hardware-based*, *software-based*, *hybrid*, and *in-network*) capture the majority of the contributions that we survey, but there are also specific aspects of cloud computing where a different classification ensures a more precise analysis of the surveyed strategies. This is the case of live migration (Section VII-C), for which we introduce the specific categories of application-aware, guest kernel assisted, and transparent techniques.

Although the dimensions of cloud networking that we consider are orthogonal to each other, they are also reciprocally influenced. For example, the network access interface that providers expose to customers is strictly related to the choice of a given I/O virtualization technology. In our discussion, we pinpoint these aspects when they are relevant and summarize the key insights in Section IX-A. At the same time, it is common that the same paper addresses more than one dimension, so in that case we mention it in every relevant Section.

The remainder of the paper is modeled after the proposed taxonomy. Section V is dedicated to *dimension 1* and discusses the problems associated to different kinds of interfaces to access the network acceleration options: the native APIs of these technologies, the standard POSIX API, or higher-level APIs. Section VI focuses on *dimension 2* (virtualization techniques), distinguishing between I/O and network virtualization. In Section VII, we survey the implications of *NAaaS* on the serviceability aspects of monitoring and logging, Quality of Service, and live migration (*dimension 3*). In Section VIII, we analyze the security challenges of the integration of network acceleration technologies in cloud platforms (*dimension 4*), by introducing strategies to enhance confidentiality, authenticity, integrity, access control, and isolation. Once completed the literature review, in Section IX we summarize the lessons learned, identify the challenges that still remain to be addressed, and discuss the future research directions. Finally, in Section X we draw the conclusions of this work.

V. Network Access Interfaces

In this Section, we consider the interfaces proposed in the literature to access accelerated networking from user applications. We organize these solutions in three categories: the *native* interfaces of the acceleration technologies, the *standard* POSIX API, or higher-level *custom* APIs. This classification stems from the different characteristics of these solutions in terms of ease of development for the end users, implementation difficulty for the cloud providers, and performance efficiency. Overall, our discussion centers on the fundamental trade-off between the efficiency of asynchronous, zero-copy interfaces and the portability and ease of use of standard APIs.

Our goal is to explore the main research directions by defining the principles and the solutions that can contribute to the emergence of an integrated evolution. Indeed, the choice of the interface has relevant consequences at the system level, in particular on the virtualization techniques that we discuss in Section VI. For instance, it determines whether providers must expose a *virtual* acceleration device to VMs and containers, or, instead, a standard virtual network adapter.

TABLE II
A Comparison Between Standard Networking (First Line) and Native APIs of Network Acceleration Technologies

Technology	Kernel integration	API	Zero-copy
Kernel TCP/IP	In-kernel	`AF_INET` Socket	No
XDP	In-kernel	`AF_XDP` Socket	Yes
DPDK	Kernel-bypassing	RTE	Yes
RDMA	Kernel-bypassing	Verbs	Yes

In this survey, our focus is on *general-purpose* network interfaces through which customers access acceleration capabilities as if they were working bare-metal. Nowadays, cloud providers also offer alternative, managed communication solutions that expose *service-specific* interfaces. In the last decade, several accelerated versions of those services have been proposed, ranging from storage solutions [26], [97], [98], [99], [100], [101] to RPC systems [102], [103], [104], from message-oriented middleware [105], [106], [107] to serverless computing [108], [109], [110]. However, all these systems offer a specific high-level service interface, preventing users to perform general-purpose network operations. Hence, we consider those systems out of the scope of this survey, leaving a detailed classification and discussion of these contributions to future work.

A. Native API

The native APIs of network acceleration technologies, such as the RDMA Verbs, are designed to maximize communication performance according to the principles of asynchronous processing and zero-copy data transfers introduced in Section III. Through these interfaces, users can control all the communication details, including the advanced features of each technology (e.g., RDMA one-sided operations). However, native APIs achieve high performance by sacrificing ease of use and portability, representing a disruptive departure from standard network interfaces. These interfaces are very low-level and complex for non-experienced systems programmers to use: not only do they assume the knowledge of a myriad of concepts and parameters (such as those introduced in Section III-E), but they are also very heterogeneous and different from each other, as summarized by Table II [111], [112], [113].

For a cloud provider, the choice to support the use of these native APIs carries relevant consequences. On the one hand, users from VMs and containers would obtain fine-grained control over several aspects of communication with minimal performance overhead, just as if they were working on a dedicated machine. Such a solution would enable existing accelerated applications to work unmodified in virtualized environments, making them portable across various deployment sites. On the other hand, that would force providers to expose a *virtual accelerated NIC* (e.g., a virtual RNIC) or its equivalent for software-based kernel-bypassing techniques (e.g., a special PMD for DPDK), which currently represents an open research challenge, as discussed in Section VI.

TABLE III
COMPARISON BETWEEN DIFFERENT APPROACHES TO NETWORK ACCELERATION TECHNOLOGIES BEHIND STANDARD POSIX API

Paper	API	Virtualization	Zero-copy	Completeness	Code available	Acceleration Technology
rsocket [114] SDP [115] UNH EXS [116]	Socket	no	no	no	yes	RDMA Two-sided
VMA [117]	Socket	no	no	no	yes	Raw UD
RemoteRegions [118]	File System	no	no	no	no	RDMA One-sided
VSocket [57]	Socket	VMs	no	yes	no	RDMA One-sided
SocksDirect [58]	Socket	Containers	if payload >16 KB	yes	no	RDMA One-sided

B. Standard POSIX API

Standard communication interfaces, such as the POSIX API, are designed to be general-purpose, portable across different hardware architectures, and easy to use. Their synchronous design and their relative simplicity made them pervasively used in any application domain, with socket-based applications that account for the vast majority of the existing cloud applications and several utilities based on them. As a consequence, many researchers have proposed to use the standard POSIX API as a transparent layer to leverage the performance benefits of the network acceleration technologies without modifying the source code of applications or requiring developers to learn new programming abstractions.

Although this choice guarantees ease of programming and preserves backward compatibility, it also potentially jeopardizes the performance advantages of network acceleration technologies. For historical reasons (see Section I), the standard POSIX API was designed on opposite principles than these emerging technologies, as discussed in Section III. For example, the POSIX API has a *write-after-send* semantic that forces data copies: after a write operation, applications can immediately reuse the data buffers because they have been copied to a library buffer. Nevertheless, the advantages of this choice have made it a quite popular option to transparently accelerate existing applications, as we discuss in the following.

From a cloud provider perspective, the contributions that follow this approach can be further classified into two groups, depending on their *virtualization awareness*, namely whether the library that exposes the standard POSIX API is aware of running within a virtual environment. In older proposals, this library is a wrapper layer that simply translates standard operations to the native API of the specific acceleration technology. In this case, providers still need to expose a virtual accelerated NIC (or its equivalent) just like discussed in Section V-A. Instead, modern proposals are virtualization-aware: the library intercepts calls to the standard operations and forward them to an optimized datapath integrated into the I/O virtualization scheme. This latter case allows providers to explore different forms of virtualization support, as we discuss in Section VI.

Table III classifies the surveyed proposals according to relevant criteria. In addition to the *virtualization awareness*, we consider also whether the *zero-copy* semantic is preserved, the *completeness* of the supported layer, and the public availability of the source code. We also report which *acceleration technology* is used when translating standard operations: as a preliminary consideration, we observe that nearly all of them target RDMA, although in different modes. In the following, we briefly comment those solutions distinguishing between *non virtualization aware* and *virtualization aware* approaches.

1) Non Virtualization-Aware Solutions: Within this first category, Rsocket [114], Sockets Direct Protocol (SDP) [115], and UNH EXS [116] share the same approach, but Rsocket appears the most successful. These solutions privilege portability over performance: once a communication operation gets intercepted by these libraries, application data are copied to a library-level buffer and then transmitted on the accelerated network, thus losing the zero-copy semantic. Rsocket provides also a custom socket interface that allows zero-copy operations, but at the price of losing transparency. An important limitation of these solutions is that they that only intercept the basic socket primitives, whereas asynchronous operations (e.g., `epoll`) are not allowed and there is limited support to multi-threading (e.g., limited support to `fork`).

VMA [117] adopts a different approach, specific to NVIDIA network hardware: once the library intercepts a network operation, data are copied, routed through a userspace TCP/IP stack (lwIP [119]), and sent directly to the NIC.

RemoteRegions [118] proposes the use of the standard *file system* API to access an RDMA network. In this model, processes expose their memory on the network as files, accessible using standard file system operations (`read`, `write`, etc.). Although that enables faster data transfers through the use of *one-sided* operations, RemoteRegions does not achieve full transparency toward applications, as memory allocation and file access is obtained by custom primitives. Moreover, these performance gains are partially lost due to data copies between application buffers and memory region buffers.

2) Virtualization-Aware Solutions: More recently, VSocket [57] and SocksDirect [58] proposed a more integrated approach specifically designed for virtualized environments, with the goal of integrating in the same library the POSIX interface and the I/O virtualization mechanisms. These works forward socket calls from the virtualized application directly to the provider-managed physical host, although in a different way. On the one hand, VSocket [57] targets socket-based applications running in VMs. For each intercepted network packet, VSocket performs one payload copy from the application to a library buffer on the sender side, and vice versa on the receive side. While this obviously hurts performance,

especially for larger payloads, a series of optimizations avoid additional copies and data are exchanged through one-sided RDMA operations through the host NIC.

On the other hand, SocksDirect [58] focuses on applications running in containers and takes a step further by introducing a mechanism for a true zero-copy semantic, based on a complex mechanism of memory page swapping. Although effective, this mechanism is fairly complex and page remapping introduces its own overhead. Hence, SocksDirect enables it only when really necessary, i.e., for large messages (>16 KB), falling back to payload copy for smaller ones. In principle, the same mechanism could be adopted by VSocket for applications running in VMs. Finally, contrary to the first category, both VSocket and SocksDirect provide complete support to the socket interface, including to asynchronous operations such as the `epoll` primitive. We further discuss the details of these two solutions in Section VI-A.

C. High-Level API

A third category of APIs explores different balances between the need for portability and ease of programming and the performance advantage of zero-copy and asynchronous interfaces. With this goal, these proposals do not replace the native APIs of acceleration technologies but rather build a different set of general-purpose programming abstractions and operations over them. Thus, although from a provider perspective these solutions do not avoid the need for a virtual accelerated NIC to applications, these higher-level interfaces simplify the concepts, parameters, and communication primitives that developers must deal with, usually defining a clear memory ownership semantic.

The main difference among these works is that they propose to build this new interface layer at different locations: in kernel space (LITE [120]) or in userspace (X-RDMA [121], nethuns [112], INSANE [113], and Demikernel [111]).

1) Kernel-Level Interfaces: LITE [120] defines a kernel-based interface that is specific for RDMA. This approach appears counterintuitive, as RDMA is widely known as a kernel-bypassing technique. Yet, by *on-loading* RDMA capabilities into the kernel, important kernel-based functionalities can be retained: in particular resource isolation, crucial in public cloud environments. The kernel keeps control on network resources (e.g., QPs), potentially enabling them to be safely shared across applications and thus improving the overall scalability. The LITE interface is based on an higher-level representation of an RDMA *memory region* with additional support for management, synchronization, and isolation. Based on that, LITE exposes a rich set of higher-level primitives, including an RPC and messaging interface, that closely resemble standard system calls, but are then mapped onto native zero-copy Verbs operations. However, a kernel-based RDMA interface forces applications to continuously cross the boundary between user and kernel spaces, incurring significant performance penalties.

2) User-Level Interfaces: X-RDMA [121], nethuns [112], INSANE [113], and Demikernel [111] propose a userspace interface to different network acceleration techniques, both hardware and software based, with the ultimate goal to enable the transparent portability of applications across heterogeneous technologies. The key design principle of these new interfaces is the minimization of network concepts and primitives, to simplify system programming in large-scale production environments. Accordingly, they define the abstraction of a *communication channel* to exchange messages among remote processes and a set of zero-copy and asynchronous primitives.

Similarly to LITE, X-RDMA focuses on RDMA and has a built-in support for RPC and for a messaging interface. Nethuns [112] and INSANE [113] share the same design principles but with the goal of easing the access to a wider range of acceleration technologies, including also DPDK and XDP, through a unified minimal interface, still expressive enough to enable the efficient implementation of higher-level domain-specific abstractions. Although these works do not directly provide a solution for an efficient virtualization of accelerated I/O, they explicitly address this issue by easing the requirements on the virtual device they require (see Section VI-A). Finally, Demikernel [111] adopts a different abstraction (a message queue) and clearly distinguishes between control plane and data plane operations. Developers can use the standard kernel-based POSIX system calls for control path operations, such as connection establishment, but then they can use a novel set of kernel-bypassing *datapath primitives* for the actual data transmission. These operations can then be mapped to RDMA or DPDK operations by the library.

D. Key Takeaways

In this Section, we considered the interfaces proposed in the literature to access accelerated networking from virtualized user applications. We distinguished three categories: the *native* interfaces of the acceleration technologies, the *standard* POSIX API, and higher-level APIs.

Our analysis considered the perspective of both the cloud users and providers. From a user perspective, standard interfaces guarantee code portability and ease of programming at the price of reduced effectiveness of the acceleration technologies, as data copies can heavily impact performance. On the opposite, the native interfaces of these technologies potentially maximize the performance benefits through their asynchronous design and zero-copy semantic. However, they are also associated with complex and heterogeneous interfaces that require specific programming expertise. Higher-level interfaces mitigate this issue and offer easier access to accelerated network operations, but sacrifice the compliance to standard APIs and thus potentially reduce application portability.

From a provider perspective, nearly all the considered contributions require the availability of a *virtual accelerated NIC*, or its equivalent for software-based techniques, which we discuss in Section VI as an open research challenge. Notable exceptions include nethuns [112] and INSANE [113], which ease these requirements and can work with regular hardware-based virtualization (see Section II-D1). VSocket [57] and

TABLE IV
KEY ASPECTS OF THE I/O VIRTUALIZATION PROPOSALS FOR NETWORK ACCELERATION TECHNOLOGIES. THESE WORKS INTRODUCE OPTIMIZATIONS TO EXISTING VIRTUALIZATION TECHNIQUES TO BALANCE FLEXIBILITY AND EFFICIENCY

	Proposed Technique	Target Environment	Interface	Base Virtualization Technique	Main Optimization
Firestone et al. (AccelNet) [81]	Hardware offload	VMs	Socket	Direct assignment	Temporary fallback to paravirt.
virtio PMD [122]	Efficient paravirtualization	VMs	RTE	Paravirtualization	Userspace frontend driver
Kim et al. (FreeFlow) [56]	Efficient paravirtualization	Containers	Verbs	Paravirtualization	Shared memory for zero-copy
Garbugli et al. (KuberneTSN) [59]	Efficient paravirtualization	Containers	Socket	Paravirtualization	Shared memory for zero-copy
Wang et al. (VSocket) [57]	Efficient paravirtualization	VMs	Socket	Paravirtualization	Shared memory for zero-copy
Fan et al. (virtio-rdma) [123]	Hybrid virtualization	VMs	Socket, Verbs	Paravirtualization	Guest/NIC resource sharing
Pfefferle et al. (HyV) [53]	Hybrid virtualization	VMs	Verbs	Paravirtualization	Guest/NIC resource sharing
Mouzakitis et al. [124]	Hybrid virtualization	VMs	Verbs	Paravirtualization	Guest/NIC resource sharing
He et al. (MasQ) [54]	Hybrid virtualization	VMs	Verbs	Paravirtualization	Guest/NIC resource sharing
Li et al. (SocksDirect) [58]	Hybrid virtualization	Containers	Socket	Paravirtualization	Container/NIC resource sharing

SocksDirect [58] take a step further and propose an integrated solution for I/O virtualization, as discussed in the next Section.

VI. VIRTUALIZATION TECHNIQUES

In this Section, we consider the challenge of the efficient virtualization of network acceleration technologies as a commodity into cloud platforms. As introduced in Section II, cloud users expect their machines and containers to efficiently exchange data to and from the physical network (*I/O virtualization*) and in a distributed context (*network virtualization*). The existing cloud model to provide these features is not suitable to accommodate network acceleration technologies: the key principle of minimal processor intervention clashes with the indirection layer that providers adopt for this purpose.

For software-based acceleration technologies, such as XDP and DPDK, providers might consider still acceptable the performance overhead introduced by standard I/O and network virtualization techniques, in light of the high degree of flexibility these mechanisms offer. However, such tolerance does not extend to hardware-based technologies: at their data transfer speed, that overhead becomes unacceptable. Therefore, recent research has proposed alternative strategies to achieve both forms of virtualization, focusing in particular on hardware network acceleration technologies such as RDMA.

In the following, according to our taxonomy, we first classify these contributions into approaches for *I/O virtualization* and for *network virtualization*. Within each category, we further distinguish between *hardware-based*, *software-based*, and *hybrid* techniques. Our goal is to show the different degrees of *efficiency* and *flexibility* they provide to applications.

A. I/O Virtualization

A good I/O virtualization solution consists in the definition of a *virtual* I/O device that preserves the performance of the corresponding physical device as much as possible. In Section II-D1, we have introduced the most common techniques for I/O virtualization, but we noted that neither of them actually meets both the *flexibility* requirements of cloud networking and the *performance constraints* of accelerated networks. Hence, researchers have explored ways to adapt these approaches to accommodate the performance properties of modern acceleration devices.

This Section describes three main techniques that have been proposed to fulfill these requirements, which we visually represent in Fig. 9: new forms of *hardware offload* (AccelNet [81]), a more efficient *paravirtualization* technique (virtio-PMD [122], FreeFlow [56], KuberneTSN [59], VSocket [57]), and *hybrid solutions* that combine the advantages of both (virtio-rdma [123], HyV [53], Mouzakitis et al. [124], MasQ [54], SocksDirect [58]). We summarize the key insights of these proposals in Table IV: we underline the main optimizations each work introduces over standard techniques for an efficient integration of hardware-accelerated networking into the existing infrastructures. We also report about the *target environment* and the *network access interface* of these solutions, crucial factors for their effectiveness.

1) Hardware Offload: The kind of *all-or-nothing* virtualization provided by SR-IOV [60] (see Section II-D) lacks the typical flexibility of software-based control and data paths, making features like monitoring and live migration almost impossible to achieve. To overcome this issue, AccelNet [81] has explored the possibility to combine the efficiency of the hardware-based approach with the flexibility and programmability of paravirtualization: instead of exposing a SR-IOV VF directly to applications, AccelNet creates a standard *virtual interface* through which applications connect with negligible overhead. During regular network operations, this interface is attached to a SR-IOV VF, benefiting from its close-to-line-rate network performance. Only when dynamic actions are required (e.g., live migration), the virtual interface is temporarily and transparently attached to a traditional paravirtualized datapath to ensure the sufficient degree of flexibility. This approach is already adopted in production by some providers to support DPDK networking in regular VMs [94], [95], [96].

Fig. 9 shows this technique on the left, where dotted lines represent the control and data paths during transition periods. Assuming that most applications do not often need to be migrated or to change their network configuration, this technique enables to leverage the full hardware speed and introduces a high performance overhead only during the temporary transition periods. However, as Section VII will discuss, when the datapath is offloaded to the hardware, cloud providers have no visibility on the network operations, making certain actions such as monitoring still impossible to achieve.

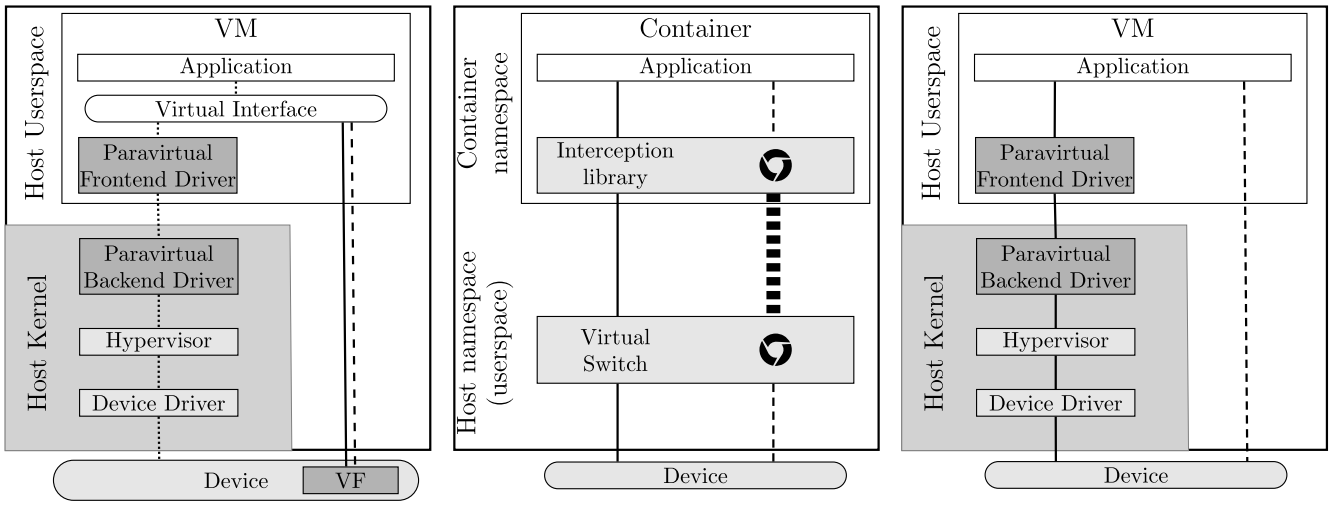


Fig. 9. I/O virtualization approaches for network accelerators: *hardware offload* (left), *efficient paravirtualization* (center), and *hybrid virtualization* (right). The solid lines represent the control path, the dashed lines the data path. The dotted line on the left figure represent the *temporary* data and control paths during dynamic network operations. The thick dots on the center figure represent shared memory.

2) Software-Based Efficient Paravirtualization: To support XDP-accelerated applications in virtual environments, currently providers rely on the standard paravirtualization technique (Section II-D1). That is not possible for DPDK and RDMA, as they both bypass the kernel and thus the standard implementation of the paravirtualization mechanisms.

Within the DPDK framework, I/O virtualization is implemented through *virtio PMD* [122], a software-based device driver that plays the role of *frontend driver* and exchange data with a backend driver in the host, thus substantially replicating standard paravirtualization techniques (see Section II-D). This is currently the solution adopted by cloud providers that do not support the hardware-based approach described above [125].

FreeFlow [56] and VSocket [57] propose a more efficient version of the paravirtualization technique, designed for high-performance networking such as RDMA and, in principle, also DPDK. Their proposals clearly separate the control and data planes, with the goal to remove the processor involvement from the data path and to enable forms of asynchronous, zero-copy communication to move packet payloads between the frontend and the backend drivers (see Fig. 9 center).

To build the data plane, these works propose the definition of shared memory areas between the two drivers, generally implemented as *ring buffers*, where message payloads will be placed. That way, the two drivers only have to exchange notifications about the presence of new messages in the designated area. In turn, the backend driver might register the shared memory area with the NIC, thus enabling true zero-copy operations: every time a VM and container posts a write operation, the payload of the request is accessible directly from the NIC, and vice versa for reception, thus removing time-consuming payload copies from the data path.

However, device drivers must still involve the processor to exchange request descriptors and completion notifications. Although these descriptors are generally exchanged through asynchronous communication channels, such as UNIX sockets or *virtio* queues, these mechanisms might still introduce too much overhead and ultimately become a performance bottleneck. FreeFlow mitigates also this problem through the definition of a *fast path* for latency-critical applications, such that even the notification exchange happens over a shared memory area. Even though this mechanism is effective, it also requires a spinning thread that continuously polls for updates the memory area, a solution not ideal for cloud environments where processor cores are precious assets [81].

Further optimizations are possible by considering the specific virtualization and acceleration technologies. For example, FreeFlow, which only targets containers, does not expose a virtual network interface to application, but gives applications the visibility of the physical NIC available on the host. When applications try to access it through the native primitives (e.g., RDMA Verbs), FreeFlow intercepts these calls through a client library that forwards them to a custom virtual switch on the host, which has the same function of a *paravirtual backend driver*. In the specific case of RDMA, any request to create a new connection will return a *virtual QP* created by the library, which plays the role of a *paravirtual frontend driver* in this case. KuberneTSN [59] follows the same approach of FreeFlow, but focusing on containerized DPDK applications. Instead, VSocket exposes a socket interface to applications running in VMs. As a consequence, once it intercepts calls to network operations, it can directly forward them to the host and thus maximize communication efficiency compared to FreeFlow. Specifically, VSocket targets RDMA and uses using a single RDMA connection to transmit all the traffic toward each remote peer.

3) Hybrid Virtualization: A number of works have introduced the concept of *hybrid virtualization* to balance the flexibility of the paravirtualization mechanism with the performance efficiency of the hardware-based option [53], [54], [58], [123], [124]. The goal of these approaches is a complete separation between the data and control planes, in order to preserve the flexibility of a software control path, typical of paravirtualization techniques, while also leveraging

the efficiency of a bare-metal access to the NIC, typical of hardware-assisted virtualization techniques. As Fig. 9 shows on the right, *hybrid virtualization* uses a standard paravirtualization mechanism for control-plane actions, such as connection establishment. Then, it lets user applications directly interact with the hardware device for the actual data plane operations, such as send and receive. To cross the virtualization layer, many of these approaches define a shared memory mechanisms to enable zero-copy data transfers, as we discuss in the following.

In a more detailed view, user applications in VMs and containers have the visibility of a *virtual NIC* bound to a *frontend* driver. During connection establishment, this driver contacts the backend driver in the host kernel, triggering the creation of a fairly complex *memory remapping mechanism*. This mechanism allows the guest and the physical NIC to share memory for exchanging data and related information in a zero-copy fashion. Thereafter, the datapath is completely memory mapped: during the actual communication phase, guest applications can directly interact with the hardware device as if it were running on the host, without any active intervention from the hypervisor on the critical path.

From the perspective of I/O virtualization, this approach represents an original space in the trade-off between flexibility and performance, because the hypervisor maintains control on the data plane to enforce properties such as isolation and portability, but without performance penalties. In fact, existing solutions that adopt this approach perform close to the network hardware limit. However, hybrid virtualization also comes with some relevant drawbacks. First, just like in hardware-based solutions, the hypervisor loses the possibility to enforce the different data plane policies that are typically used by cloud providers to manage traffic on established data connections: it can only act with per-connection granularity. Secondly, this approach is limited to connection-based communication and is not applicable for datagram-based interactions, as these lack a preliminary setup phase. Lastly, this approach poses strong maintenance constraint on the provider to keep the frontend driver and the memory remapping mechanisms up to date with the hardware evolution.

A final consideration about hybrid approaches is that all the existing proposals only consider VMs, except for SocksDirect [58], which offers a socket interface for containerized applications: a monitor process runs in each host, acting as a backend driver with the role to set up the direct data path among local or remote containers. Communication occurs through a shared memory channel for containers located on the same host, whereas remote containers interact directly without the mediation of the monitor, according to the hybrid approach. In this case, since a container is just a process for the operating system, there is no need for a custom frontend driver and containerized applications can directly use the host driver.

B. Network Virtualization

Through *network virtualization*, cloud providers build the illusion that user machines (and containers) communicate on a private network when in reality they share the same physical infrastructure with other cloud tenants. In this Section, we survey the proposals to create *accelerated* virtual private overlay networks in public clouds (e.g., virtual private RDMA networks) that preserve the flexibility of standard network virtualization techniques.

As pointed out in Section II-D2, the core of any network virtualization solution is a *virtual switch*, a component that forwards traffic between the virtualized applications and the physical network. The *programmability* of virtual switches is a crucial property, as it enables providers to dynamically configure and re-configure network features, such as the overlay network topology. However, because typical virtual switches are software-based, they turn out to be inefficient for both software-accelerated and hardware-accelerated network speeds, especially in reconfiguration.

According to our taxonomy, in the following we consider various approaches proposed in the literature to overcome this obstacle and classify them as *hardware* (AccelNet [81], Nitro [126], BlueBird [127]), *software* (VSocket [57], FreeFlow [56]), and *hybrid* (or *software-defined*) (MasQ [54]), depending on how and where the typical functions of a virtual switch are implemented to improve the overall performance. Table V provides a summary of this classification. In the following, we comment each category in more detail by focusing on how the virtual switch is implemented by different contributions for different acceleration technologies.

As a preliminary consideration, we note that the network virtualization of software-based acceleration is easier to support for providers. Both XDP and DPDK let applications directly manipulate Ethernet frames, leaving to applications the choice of which higher-level protocols to implement. In the cloud context, usually applications adopt the standard TCP/IP protocols to enable compatibility with existing network infrastructures. That makes it easier to use standard hardware-based or software-based switches as a base to design improved switching mechanisms. Instead, RDMA adopts a different communication model which requires much deeper modifications of existing tools, including virtual switches.

1) Hardware Approaches: Hardware-based solutions offload the virtual switch to a hardware device. In particular, the network virtualization functions can be implemented at different positions between the physical host, where the virtualized applications are running, and the first hop of the underlying network: on the NIC, as an embedded feature of the device; on a ToR switch; or in the middle between the two (*bump-in-the-wire* approach). To improve both the scalability and flexibility of those devices, two trends have recently emerged: on the one hand, major cloud providers started developing *custom solutions* with enhanced programmability; on the other hand, *modern hardware*, such as programmable NICs [79] or ToR switches [128], [129], is increasingly capable of dynamic actions that programmers can easily configure through standard languages such as P4 [130], [131].

Many off-the-shelf NICs currently offer a hardware implementation of virtual switches, such as OVS [63], but although

TABLE V
AN OVERVIEW OF THE NETWORK VIRTUALIZATION APPROACHES PROPOSED TO ENABLE *Accelerated*, VIRTUAL PRIVATE OVERLAY NETWORKS IN PUBLIC CLOUD PLATFORMS

	Network Virtualization Approach	Implementation of *virtual switch* functions
Firestone et al. (AccelNet) [81]	Hardware	*bump-in-the-wire* device (special-purpose FPGA)
Amazon Nitro [126]	Hardware	SmartNIC (special-purpose ASIC)
Arumugam et al. (BlueBird) [127]	Hardware	SmartToR (P4 programmable switch)
OVS-DPDK [63]	Software-based	Data plane offloaded to DPDK
Wang et al. (VSocket) [57]	Software-based	Partially embedded in the I/O virtualization mechanism
Kim et al. (FreeFlow) [56]	Software-based	Centralized controller for memory address translation
He et al. (MasQ) [54]	Hybrid (software-defined)	Centralized controller for vGID/GID translation

these implementations can achieve optimal performance, they also have some important limitations when deployed at scale in data centers. First, this form of hardware offload suffers from scalability problems, as the on-chip memory is usually limited and it is challenging to cache the contexts of several virtual networks [132]. Secondly, the degree of *programmability* of such options is still far from the software-based equivalents: despite the differences of the specific implementation technology (ASIC, System-on-a-chip, FPGA, etc.), writing programs for such devices is not as easy as using standard software tools for the same goals [133], [134].

Among the *custom solutions*, major cloud providers have mainly focused on the design and deployment of *special-purpose* devices to augment the capabilities of commodity network cards with embedded switching features, or to place a programmable hardware device between the network switch and the card itself (*bump-in-the-wire* approach). Azure AccelNet [81], and Amazon Nitro [126] are the most notable examples of this latter approach that consists in placing a specially-designed device in cascade to the NIC to implement the switching features. AccelNet [81], in particular, uses FPGA hardware to build a device that offers hardware-like performance efficiency, but also exposes a rich interface, partially in software and partially in hardware, to achieve the result of *software-like* programmability.

Such custom approaches currently represent the closest solutions to achieve the goal of *Network Acceleration as a Service*, because they have been specifically designed to implement data path operations in hardware while also offering built-in support to network virtualization. Indeed, as we will comment in Section IX, these proposals represent early examples of an evolution of commodity network hardware acceleration devices to include features for cloud virtualization.

Overall, however, such custom approach has also some important drawbacks: first, only major cloud providers have sufficient resources to design and deploy custom hardware devices. Furthermore, because the virtualization features embedded into the devices must be programmable by software, usually these devices do not completely offload all the switching capabilities and some control path operations remain in software, requiring a careful orchestration between the hypervisor and the custom hardware device. Finally, custom solutions are generally tailored to specific provider needs and do not necessarily represents optimal solutions under all possible aspects [134].

An alternative approach to custom hardware design for network virtualization offload is today the utilization of *modern programmable network cards* (*SmartNICs*) or switches (*SmartToR*s) [128], [129]. With these devices, it is possible to programmatically define custom and dynamic packet processing pipelines through a set of scripts in the standard P4 language [130] that will be enforced by the hardware device. Compared with *static* hardware implementations of virtual switches, this approach allows a much more flexible traffic management and definition of virtual private overlay networks for cloud tenants. For example, Azure BlueBird [127] implements and deploy a high-performance network virtualization system for the Azure *bare-metal* instances, where hardware accelerators are available to cloud customers. Combined with the I/O virtualization strategies surveyed in the previous Section, this approach has the potential to meet the flexibility requirements of cloud networking while preserving the performance advantage of hardware-accelerated networking.

2) Software-Based Approaches: The standard software-based virtual switches, such as OVS [63], introduce an unacceptable overhead on high-performance communication. To preserve its significant flexibility, a faster version of OVS, called OVS-DPDK, has been designed. OVS-DPDK retains the same switching logic and programmability properties, but the data plane is accelerated using DPDK. Thus, any application (including XDP and DPDK-based applications) can transparently obtain a faster overlay network with minimal complexity for the provider. However, even in OVS-DPDK, the complex switching logic is completely software-based and thus introduces a non-negligible overhead on the data plane.

To further reduce the performance overhead, some contributions designed software-based switches that introduce further optimizations, whose nature and effectiveness depend on which network access interface is exposed to cloud users: *POSIX* or *Verbs*. These proposals, in particular the latter, mainly address RDMA networks, as the performance impact of a software-based data plane is particularly critical.

In the first case, VSocket [57] explores the possibility to *partially eliminate* the virtual switch from its communication model, leveraging the same indirection layer that enforces the I/O virtualization (see Fig. 9 center) to also set up an overlay network. To allow two remote guest applications to communicate, the two local backend drivers need to know the IP address of the peer *physical* host in order to open a shadow RDMA connection. To obtain it, the VSocket frontend drivers within the VMs first ask their local backend driver for the

(encrypted) local host IP address and exchange it with the remote counterpart over a regular TCP socket. Once received the remote physical IP, the frontend drivers communicate it to the local backends, which decrypt it and set up the RDMA communication channel.

When the *Verbs* interface is exposed to applications, software-based virtual switching becomes more challenging. Differently from XDP and DPDK applications, which can rely on standard cloud mechanisms to build overlay networks, RDMA endpoints must exchange a certain amount of information, such as GIDs and the memory addresses to read/write data (see Section III): depending on how the I/O virtualization is implemented, the creation of an RDMA overlay network requires the *translation* of one or both these tokens between their *virtual* and *physical* values. User applications running in containers, like in FreeFlow [56], have direct visibility of the host physical NIC, thus they can simply exchange the physical GIDs through a TCP connection. However, RDMA peers also need to exchange the addresses of remotely-accessible memory areas: because these addresses are only valid in the address space of the containerized application, the physical RNIC cannot use them directly. To enable memory translation, FreeFlow introduces a logically centralized *controller* that keeps the mapping between virtual and physical addresses of each tenant (a design similar to SDN techniques). Although this solution requires that nodes retrieve such information on the network, the performance impact of this solution is not high: virtual switches would retrieve the mapping from the controller during the connection establishment phase and have the possibility to cache them. Instead, when user applications run in VMs, they might have the visibility of a *virtual RNIC*, with a virtual GID that requires a similar translation strategies: we discuss this case in the next paragraph.

3) Hybrid Approaches: The hybrid approaches for I/O virtualization leave only the control plane under the supervision of the hypervisor, whereas the data plane is directly managed by the virtualized applications. As a consequence, virtual switches have only control on the communication control plane: whereas they usually enforce data plane policies *per-packet*, with this approach only *per-connection* actions are possible. MasQ [54] defines this option a form of *software-defined* network virtualization: all the control actions are performed at connection establishment time, and then data flows with neither hypervisor mediation nor forwarding step.

Hence, the set up of overlay networks must happen during this preliminary phase. Although for XDP and DPDK applications, which typically use TCP/IP protocols, that would be straightforward using standard tools, this process is much more difficult for RDMA channels. In that case, virtualized applications must first exchange RDMA-specific information with the remote peer. If the application has direct visibility of the host RNIC, the solution discussed in the previous paragraph can be adopted. Instead, when applications have visibility of a virtual RNIC to applications (e.g., when applications run in VMs), the network virtualization problem becomes slightly different. When a guest application wants to connect to a remote peer, it will target a virtual IP address (on RoCE networks; for Infiniband networks, there is an equivalent address), a virtual GID (vGID), and a virtual memory address; to establish the connection on behalf of the guest, the host on the initiator side must know a corresponding physical IP address and a physical GID, as well as the physical memory address for RDMA.

To this end, the role of the virtual switch is to decouple the network and memory views of guest applications and hosts. Also in this case, a possible approach to configure the switch with such information is to have a centralized controller that keeps the association between a vGIDs, different for each tenant, and the corresponding physical GIDs, as well as between the virtual and the physical values of any other token.

C. Key Takeaways

In summary, this Section reviewed the literature on the virtualization techniques for network acceleration technologies in cloud platforms. We first considered *I/O virtualization* approaches, whose goal is to efficiently transfer data between the physical devices and the isolated environment where user applications execute. Then, we surveyed solutions for *network virtualization*, which aim at enabling virtual private overlay networks among remote user applications through the adoption of a virtual switch. In both cases, we classified the solutions as *hardware*, *software*, and *hybrid* approaches.

We noted that only software approaches currently guarantee the necessary flexibility for providers to keep full control on user network operations, by integrating the acceleration technologies with existing cloud mechanisms (hypervisors, virtual switches). Although several technical solutions analyzed in this Section aim at reducing the performance impact of this software interposition, the optimization they introduce are mainly useful for software-based acceleration technologies such as XDP and DPDK. For technologies based on hardware accelerators, such overhead is still too high to allow applications to benefit in a meaningful way. Hybrid approaches significantly mitigate this overhead, but they also excessively reduce the granularity of possible provider interventions. We further expand these consideration in Section VII.

Conversely, hardware approaches provide excellent performance but a reduced degree of programmability and flexibility. Nevertheless, these solutions are the closest to achieve the goal of *NAaaS* by enhancing hardware devices with both additional programmability and custom virtualization support. Although these solutions are currently difficult to program and often proprietary, we believe that a tighter cooperation of both cloud providers and hardware manufacturers could further advance the state of the art, as Section IX extensively discusses.

VII. Serviceability

In this Section, we survey recent contributions that propose to support *serviceability* in combination with network acceleration techniques. As introduced in Section II-E and IV, some typical serviceability aspects are challenging to combine with accelerated networking: *monitoring and logging*, *QoS*

TABLE VI
AN OVERVIEW OF THE DIFFERENT ASPECTS OF THE *Serviceability* CONSIDERED BY THE LITERATURE ON *NAaaS*

	Monitoring and Logging	QoS Enforcement	Live migration
Kim et al. (FreeFlow) [56]	Software-based	—	—
Ma et al. (X-RDMA) [121]	Hybrid	—	Application-aware
He et al. (MasQ) [54]	Hybrid	Hybrid	Application-aware
Mellanox ConnectX-6 [78]	Hardware-based	Hardware-based	—
Firestone et al. (AccelNet) [81]	Hardware-based	Hardware-based	Application-aware
Xing et al. (Bedrock) [135]	In-Network	—	—
Huang et al. (Nomad) [136]	—	—	Guest kernel-assisted
Planeta et al. (MigrOS) [137]	—	—	Transparent

enforcement, and *live resource migration* are the main concerns, because the standard mechanisms to provide them tend to be bypassed for the sake of performance.

The different *network virtualization* strategies we explored in Section VI-B directly influence whether these serviceability aspects are more or less difficult for providers to implement, independently of the specific network acceleration technology adopted (software-based or hardware-based). Since these serviceability aspects are implemented by providers within virtual switches, retrieving and manipulating the state of those switches is the main concern of the approaches we present here: providers face a trade-off between the flexibility and ease of use of software-based solutions and the performance of the hardware-offloaded approaches. The discussion in this Section considers how each work explores this trade-off.

In the following, we classify the approaches proposed in the literature on this topic based on which of these serviceability aspect they mainly focus on, as summarized by Table VI. Within each category, we further distinguish these works as *hardware*, *software*, *hybrid*, and *in-network* approaches, consistently with our proposed taxonomy.

A. Monitoring and Logging

Researchers have explored different strategies to make monitoring and logging tools coexist with the high-performance data paths discussed in Section VI-B. Accordingly, we categorize these approaches as *hardware-based*, *software-based*, *hybrid*, and *in-network*.

1) Hardware-Based Approach: Various industrial and academic initiatives aim at enhancing the capabilities of network cards by exposing, to different degrees, the internal state of hardware-based virtual switches: modern *SmartNICs* [78] and AccelNet [81]. This approach combines the benefits of the hypervisor mediation, typical of *virtual* switches, with the performance advantages associated with their implementation via specialized devices. Major NIC manufacturers increasingly offer *SmartNICs*, enhanced off-the-shelf network cards that provide a built-in implementation of a full-fledged virtual switch. For example, NVIDIA ConnectX-6 [78] is an ASIC-based NIC that comes with a built-in hardware implementation of OVS [63]. The main drawback of this technique is that such devices are generally not programmable, making it hard for providers to update, upgrade, and change in any way the switch implementation.

An interesting attempt to avoid embedding static switch implementation into network card is the FPGA-based, bump-in-the-wire approach introduced in AccelNet [81]. AccelNet builds a custom hardware-based virtual switch with a three-fold goal: first, to enable large-scale monitoring through the collection of a large number of fine-grained metrics from each infrastructure component; second, to make such solution efficient in terms of performance; and third, to enable a frequent and dynamic upgrade of the switch logic when necessary. Although developed *ad-hoc* by a provider through software-hardware co-design, this effort represent the most complete and concrete approach so far to monitoring virtual accelerated network in a large-scale public cloud, by offering an insight on the kind of infrastructure that is required to make virtual accelerated networks serviceable.

2) Software-Based Approach: The availability of a software switch makes it easier to track each connections and their associated traffic, as the hypervisor mediates every network operation and thus enables a high degree of observability on the network resources. OVS-DPDK bypasses the standard cloud observability mechanisms, but it relies on the DPDK observability libraries to recover equivalent functions. Because OVS cannot handle RDMA communications, FreeFlow [56] solves the same challenge for that technology, by introducing a central controller to gather metrics from a cluster of physical hosts, where user containers can communicate through virtual private overlay RDMA channels.

The drawback of these solutions is that the processor must be involved in each network operation in order to gather and log the associated monitoring information. To mitigate this effect, generally it is necessary to introduce sophisticated optimizations, such as the FreeFlow *fast path* commented in Section VI-A, which requires a high resource consumption.

3) Hybrid Approach: According to the hybrid network virtualization strategy (see Section VI-B3), the hypervisor only mediates the control plane of an accelerated communication, leaving the data plane directly interact with the hardware. The consequences of this approach are relevant on the monitoring and logging infrastructure, because the hypervisor keeps visibility only of *per-connection* information, such as the number of active endpoints on a certain host.

X-RDMA [121] and MasQ [54] analyze the implications of this lack of visibility on observability tools. The solutions they propose are specific for RDMA, because standard cloud mechanisms can be used to trace the TCP/IP connections typically adopted by applications that use either standard networking or accelerated networking via XDP or DPDK.

X-RDMA [121] is a significant example of adoption of a hybrid strategy by a major cloud provider. It reports about the deployment of a set of tools for tracing, tuning, and monitoring a large-scale production cluster on top of RDMA networks. Specifically, X-RDMA introduces a monitoring system that integrates three utilities: a tool to retrieve *per-connection* statistics, a *ping* tool to detect latency issues, and a benchmark test to understand possible performance issues. Although X-RDMA does not directly target a cloud environment, the proposed approach can be extended to cloud-oriented solutions that use a hybrid network virtualization strategy. To this end, MasQ [54] proposes the introduction of a feature called *connection tracking* in its solution for network virtualization.

However, the complete transparency of the data plane to the provider cannot be considered an acceptable solution in a cloud platform, because providers lose the possibility to monitor important aspects such as the actual performance a user is experiencing. X-RDMA partially retrieves such information for RDMA networks by using additional metrics such as PFC status, queue drop counters, and buffer utilization, but in a context in which only internal provider services access the RDMA network. Instead, when untrusted external customers are allowed to communicate over an accelerated network, provider control is fundamental. ReDMArk [138] suggests to leverage the rich set of hardware counters exposed by NICs to monitor networking events (see Section VIII).

Yet, no current hybrid solution for network virtualization in cloud platform has adopted similar strategies. *Hybrid* monitoring and logging strategies for accelerated cloud networking might become practically viable only when providers will have the option for a complete visibility of the data plane.

4) In-Network Approach: In-network approaches move the collection point of observability information from the physical host to the physical network: monitoring and logging is performed at the first network hop of the host's traffic. Following this approach, Bedrock [135] provides a practical solution for the efficient monitoring and logging in accelerated cloud environments: through programmable hardware devices and a set of P4 scripts [130], it recovers missing observability functionalities that are typically lost in kernel-bypassing network virtualization, introducing minimal overhead on the data plane. In particular, Bedrock tracks packet flows directly in Top-of-Rack (ToR) switches that keep a state for each active connection (compressed to minimize the memory overhead).

Not only this approach promises to enable a fine-grained monitoring of network operations, but it also appears effective for the detection of a wide range of security attacks that would be impossible to discover when the data plane is offloaded to hardware devices (see Section VIII). Finally, logging is enabled by forwarding relevant packets to a log server.

Overall, *in-network approaches* appear interesting not only *per se*, but potentially as a complement go the hybrid solutions previously commented. Indeed, for providers that do not have the resources to develop custom *hardware-based* solutions, the combination of hybrid and in-network approaches might offer a sufficient degree of programmability, performance efficiency, and observability of the cloud platform.

B. QoS Enforcement

Depending on the SLA negotiated with each customer, providers usually distinguish different classes of services: limited resources are generally available to users for a low cost, whereas more powerful capabilities are generally associated to a higher price. From the network perspective, that level-based differentiation is possible only if the provider is able to enforce heterogeneous Quality of Service (QoS) policies, such as network quotas, caps, and other forms of limitations on network resources. In this Section, we survey the literature proposals to enforce those policies on virtual switches that can be *hardware-based* (modern *SmartNICs* [78] and AccelNet [81]), *software-based* (FreeFlow [56]), and *hybrid* (MasQ [54]).

1) Hardware-Based Approaches: Some hardware manufacturers already enhance NICs with virtualization support, such as forms of *QoS offloading* (e.g., rate limiting) [78]. However, cloud providers cannot let those functions to be accessed directly, because they must control user behavior. Either they use a form of direct device assignment, which offers limited flexibility, or they use these advanced features from software switches, selectively offloading the enforcement of specific policies. AccelNet [81] represents an innovative alternative, although its proprietary hardware-software co-design is hard to reproduce: it proposes to augment the NIC capabilities by integrating a additional FPGA device alongside the network card, which supports the implementation and execution of several software-programmable QoS policies, even non-trivial. For example, AccelNet can support rate limiting policies on a per-flow basis, a much finer granularity compared to the hybrid approach of MasQ [54].

2) Software-Based Approaches: The presence of a software switch makes it easy for cloud providers to reuse existing, standards tools to enforce QoS policies, as the hypervisor can have full control of both the control and the data planes. The obvious drawback of this approach is a heavy performance penalty on network operations, which in this case is even higher than for observability tools. Indeed, typical QoS policies require to limit the data rate on the network to a given value, involving the processor even more into the network operations. Although FreeFlow [56] does not explicitly mention this aspect, their design could easily allow the enforcement of QoS policies within the virtual switch.

3) Hybrid Approaches: In hybrid approaches to network virtualization, the control plane is still under the full control of the hypervisor, so the *control-plane* QoS policies are straightforward to implement (e.g., enforce a quota on the number of active endpoints). However, because the hypervisor has no visibility on data plane operations, the only possible solution to enforce *data-plane* policies (e.g., rate limiting) is to resort to the hardware-based techniques, provided that the network card supports the advanced features required by them.

Alternatively, a limited set of QoS policies can still be enforced through the use of SR-IOV VFs, following an approach introduced by MasQ [54]. This strategy relies on the creation of a set of VFs, each configured with predefined rate limit. Then, each endpoint is configured to use the VF that enforces the desired rate limit. Although this solution might

be effective in the short-term, it comes with some important drawbacks. First, only basic QoS policies, such as a rate limiting to a given set of possible thresholds, can be supported at the end hosts. Second, the configuration of a set of SR-IOV VFs is currently not as easy and flexible as configuring software-based virtual switches, which is the main goal for public cloud providers.

Hence, QoS enforcement remains an open challenge for hybrid virtualization approaches, whose solution might require the combination with in-network techniques (see Section VII-A) to recover the missing serviceability features.

C. Live Migration

Live resource migration is a powerful and defining capability of cloud infrastructures, as discussed in Section II-E3. Network acceleration technologies were not initially designed to support live migration: the state of active communication channels is split between user applications and, in case of RDMA, hardware devices, and thus not visible to providers. Recently, researchers started to investigate possible strategies to achieve that goal, by exploring a trade-off between the *transparency* of the migration process for user applications and the availability of *hardware support* from manufacturers.

By referring to that trade-off, to enable a more precise analysis of these works, this Section will categorize the papers differently from previous Sections. We distinguish three different approaches to enable the migration of active accelerated communication channels: *application-aware*, which sacrifice transparency but does not require changes to cloud infrastructures (X-RDMA [121], MasQ [54]); *guest-kernel assisted*, which leverage a software layer on the data plane (Nomad [136]); and *fully transparent*, which require protocol changes in acceleration technologies (MigrOS [137]).

The complexity of live migration solutions largely depends on the network interface exposed to user applications (see Section V). If the adoption of network acceleration is hidden behind the standard POSIX interface, transparent live migration is not a major issue: SocksDirect [58] and VSocket [57] support this feature for VMs and containers respectively. Indeed, those systems already work by redirecting socket operations to the native interfaces of the acceleration technologies. When live migration is needed, it is sufficient to redirect the data path to a *standard in-kernel TCP/IP connection* and then apply the well-established migration techniques. Once the migration is completed, the traffic is routed back to the new accelerated connection. AccelNet [81] adopts a similar solution: the interface exposed to users is actually a switch between a SR-IOV VF, used during regular operations, and a paravirtualized stack, used during migration (Fig. 9 left).

On the other hand, when the native interface of an acceleration technology is directly exposed to applications, two main issues emerge. Although these issues hold for any kind of network acceleration, they are more concerning for hardware network acceleration and in particular for the RDMA communication model, which is based on different assumptions than TCP/IP networking (see Section III-B). First, the communication state, which resides partially in the virtualized user application and partially in the hardware NIC, is transparent to the hypervisor. For instance, RDMA *one-sided* operations can modify memory pages, or the NIC may silently change its portion of application state. Second, the presence of several location-dependent identifiers (see Section III-E) makes it hard to relocate resources to a different hosting environment.

In the following, we comment how the three different live migration strategies handle these issues.

1) Application-Aware Migration: A first option to support the live migration of accelerated communication channels is to make guest applications aware of the migration operations and let them implement the migration logic. This idea is introduced first in AccelNet [81] by observing that most applications that use network acceleration in their data centers are already programmed to gracefully fall back to kernel-based TCP/IP networking in case of failure of an *accelerated* channel. Thus, if the hypervisor shuts down all the *accelerated* channels, user applications would redirect communication to standard in-kernel channels, which cloud providers can easily handle.

Also X-RDMA [121] and MasQ [54] refer to this strategy as a possible way to support live migration in their respective solutions. This approach has the advantage of enabling live migration even in absence of specific hardware or OS stack support, and appears efficient also in terms of performance. However, it forces developers to take responsibility of the migration logic, a task that logically belongs to the hypervisor.

2) Guest-Kernel-Assisted Migration: If the I/O channels are paravirtualized (see Section VI-A), the software indirection layer can enable a form of live migration transparent to guest applications, in particular for software-based acceleration techniques such as XDP and DPDK. Instead, for hardware-based acceleration there are two main challenges: handling location-dependent resources and checkpointing the connection state, which partially resides on the NIC.

For instance, location-dependent resources in RDMA channels are QPNs, MRNs, memory keys, and the LID (see Section III-E). Similarly to how IDs are assigned to processes by operating systems, those identifiers are sequentially assigned by the local RNIC. This is not an intended feature to support live migration (in Section VIII we will mark this behavior as a security vulnerability), but it can be leveraged in this sense.

Nomad [136] proposes to abstract these IDs by introducing a layer of *namespace virtualization* in the software switch: at resource creation time, this layer returns to applications VM-specific identifiers instead of the actual location-dependent information. Thus, after VM migration, the application code will not observe any change despite the location change. Such virtualization layer is implemented by a *modified frontend driver* in the guest kernel, which keeps the mapping between local and virtual resource identifiers. The same frontend driver plays a crucial role also for the second challenge, i.e., the connection state checkpointing. Because the hardware NIC does not expose its portion of connection state, Nomad proposes that the frontend driver implements a form of *QP*

suspension: after a migration request, the driver suspends all the operations on all the active QPs, waiting for the outstanding send operations to complete. Then, it notifies all the peer QPs to do the same on their side. Additional WRs on both sides are cached by the driver during suspension time.

This simple migration protocol ensures that, even without access to the internal state of the NIC, the connection state eventually becomes deterministic and a consistent checkpoint can be safely performed. After migration, communication can be simply resumed by the frontend driver.

This guest-kernel-assisted approach makes the migration process transparent to the guest applications, but not to the guest operating system, as a modified user library and frontend driver implement the necessary abstractions and protocols. As a consequence, this solution can apply only to virtual communication channels based on paravirtualization.

3) Transparent Migration: A truly transparent solution for the live migration of *accelerated* cloud applications would only be possible with proper support from NIC vendors, especially in case of hardware-based acceleration. To prove that, MigrOS [137] extends RoCE (see Section III-C), an RDMA protocol, to natively support this feature, investigating the overall benefits and costs.

Similarly to Nomad [136], the goal of these changes is to get a deterministic checkpoint of the connection state without the NIC silently changing resources, but this time without the requirement of a software support layer in the guest kernel. Hence, MigrOS introduces two new state to the QP state machine (see Fig. 5(b)) and two new protocol messages. When the migration procedure starts on one node, all the QPs of the target process go into the *Stopped* state: they do not send or receive further messages. When a QP learns that its peer QP is stopped, it transition to the *Paused* state, in which it does not send any further message. A simple reliable migration protocol handles these state transitions. Once every QP is stopped and has its peer paused, it is safe to checkpoint the process state.

MigrOS modifies a popular framework for checkpointing and restoring Linux processes (CRIU [139]), widely used in container runtimes, to include *Verbs* objects in the checkpointed state. To this end, they add two new primitives to the *Verbs* API, one for dumping the *context* of a connection (see Section III) and one for restoring *Verbs* object on the new host. Finally, to handle location-dependent identifiers, a simple virtualization solution like that previously discussed for Nomad [136] would be sufficient. We note that for XDP-based and DPDK-based applications, these changes are not necessary: if applications adopt the TCP/IP protocols, their traffic can be handled by the unmodified CRIU framework.

Overall, these three changes are sufficient to enable the *transparent live migration* of RDMA applications, which is applicable to both VMs and containers and does not require any modification to the user application nor to the hypervisor or container runtime. However, the most significant limitation to this approach is that the RoCEv2 protocol is generally implemented in hardware, so any change must be supported by the card vendors. MigrOS, which implemented the proposed solution on a software version of the protocol, claims that such changes are small and backward compatible.

In Section IX, we discuss how this kind of features should become part of the RDMA ecosystem in order to start offering network acceleration as a commodity in cloud platforms.

D. Key Takeaways

In this Section, we reviewed the proposals to preserve typical serviceability functions while integrating network acceleration technologies into cloud platforms. Our discussion started from the insight that the placement of virtual switches significantly impacts serviceability aspects. We reviewed three of them - monitoring and logging, QoS enforcement, and live resource migration - that we identified in Section II-E as the most impacted by this integration, and considered *hardware*, *software*, *hybrid*, and *in-network* strategies.

On the one hand, hardware-based solutions are the most efficient, but also mostly static and do not allow providers to achieve the level of programmability, flexibly, and dynamicity they need. Some providers started developing their own hardware-software co-designed solutions, but these designs are proprietary. In Section IX, we discuss how making them generally available could be a further step toward the practical availability of *NAaaS*. On the other hand, software-based virtual switches that can introduce significant performance overhead, especially for RDMA-based networks.

Hybrid solutions showcase promising performance, but they provide an excessively coarse-grained visibility on the switch state. In principle, they can be combined with in-network approaches, which act on the network infrastructure, to achieve a sufficient blend of programmability, performance efficiency, and observability of the cloud platform, by recovering in-network the features bypassed through hardware offload.

For the specific aspect of *live migration*, we distinguished between *application-aware*, *guest-kernel-assisted*, and *fully transparent* strategies. The first two approaches have intrinsic limitations. In the first category, applications are required to be aware of virtualization, thus preventing backward compatibility and requiring developers to consider non-trivial migration-related concerns in addition to their business logic. The second category relies on the availability of a guest kernel, hence it applies to VMs only, and on software-based I/O virtualization, which is not the most efficient option for high-performance networking (Section VI-A). Conversely, transparent strategies face mostly practical challenges, such as the need for proper hardware support. In Section IX, we will discuss how closer cooperation between hardware manufacturers and cloud providers might help solve these issues.

VIII. Security

In this Section, we classify the literature on the security of network acceleration technologies in integration with cloud computing infrastructures. These works mainly consider three security properties: *confidentiality, integrity, and authentication*; *access control*; and *isolation*. These aspects are by far the most impacted by the introduction of acceleration techniques,

TABLE VII
A SCHEMATIC OVERVIEW OF THE SECURITY PROPERTIES OF *NAaaS* CONSIDERED IN THE SURVEYED LITERATURE. "*A*" MEANS THAT THE WORK PRESENTED AN ANALYSIS OF THAT TOPIC, OFTEN INCLUDING THE PROPOSAL OF SOME FORM OF VULNERABILITY MITIGATION. "*S*" MEANS THAT THE PAPER PROPOSES A NOVEL SOLUTION FOR THAT SPECIFIC ASPECT, AS DISCUSSED IN THE REMAINDER OF THE SECTION

	Confidentiality Integrity Authenticity	Access Control	Isolation
Lee et al. [140], [141]	S	—	—
Simpson et al. [142]	A	A	—
Taranov et al. (sRDMA) [143]	S	—	—
Xing et al. (Bedrock) [135]	S	S	S
Tsai & Zhang [90]	A	A	A
Tsai et al. (Pythia) [144]	—	—	A
Kurth et al. (NetCAT) [145]	—	—	A
Rothenberger et al. (ReDMArk) [138]	A	A	—
Pinkerton et al. (RFC 5042) [146]	S	—	—

because the standard cloud security mechanisms tend to be bypassed for the sake of performance.

The security challenges we discuss in this Section mainly apply to hardware-based network acceleration technologies. For software-based technologies, like XDP and DPDK, confidentiality, integrity, and authentication are under the responsibility of user applications, which tend to adopt standard protocols to enforce them (e.g., TLS). To support access control and isolation for XDP applications, providers can rely on standard kernel-based mechanisms. DPDK provides its own access control library, whereas the isolation issues described in Section VIII-C do not apply as they are based on one-sided RDMA operations or target specialized hardware devices.

Instead, for hardware-based network acceleration techniques, these challenges originate in the difference between the security context of cloud platforms and the security model for which these technologies were originally intended. RDMA was designed for environments that assume a high degree of trust among users and, when this is not the case, enforce physical resource isolation. In contrast, cloud platforms have an opposite security model: resources are shared among several untrusted users, and the physical infrastructure, although virtualized, is publicly accessible (see Section II-F).

Toward the goal of *NAaaS*, researchers have started to investigate the security implications of accelerated networking in the cloud context, unveiling a dramatic situation. For instance, built-in security mechanisms are largely ineffective against both same-host and in-network attacks, making applications vulnerable to denial of service, packet injection, and side-channel attacks, which could result in devastating consequences such as attackers stealing entire databases with no trace, or silently injecting code in other users' memory areas [135], [138], [140], [142], [144], [145].

Table VII summarizes the papers we survey. They either *assess* the security vulnerabilities of network acceleration technologies in cloud platforms or *propose* novel solutions to achieve stronger security properties in cloud environments. Although the availability of a mature security framework for *NAaaS* appears still far away (see Section IX), these works represent a first important step toward the definition of foundational principles for a *secure* accelerated cloud networking.

In the following, we expand these considerations by analyzing these three security aspects separately. Consistently with the goal of this work, we will limit our overview to the challenges arising from the integration of network acceleration into cloud platforms, adopting RDMA as a reference. A broader coverage of the vulnerabilities of hardware accelerators and of the efforts to mitigate them is out of the scope of this survey and can be found in complementary work [147].

A. Confidentiality, Integrity, and Authentication

A significant number of works has investigated how to provide confidentiality, integrity, and authentication properties to accelerated cloud networks: Lee et al. [140], [141], Simpson et al. [142], sRDMA [143], Bedrock [135], Tsai & Zhang [90], ReDMArk [138], and RFC 5042 [146]. All these contributions agree that only a hardware-based approach can provide the necessary encryption and authentication without significantly harming performance, but there is no convergence on a specific solution yet: some of them implement the proposed mechanisms on the host NIC; others in ToR switches. Before considering those proposals, we summarize the main challenges that accelerated cloud networking faces in terms of confidentiality, integrity, and authentication.

RDMA endpoints exchange data and metadata in plaintext, without any form of encryption and authentication. In a multi-tenant cloud data center, this lack of protection exposes accelerated traffic to several different potential attacks affecting not only data *confidentiality* but also data *integrity*: an eavesdropper located anywhere on the network can not only easily access sensitive information, but also arbitrarily modify the traffic end even forge new packets. In fact, packet alteration of headers and/or payload only requires the attacker to recalculate two checksums whose seeds and algorithms are defined in the Infiniband specification, and little additional effort is required for packet forging. Indeed, the RNIC checks four elements to accept an incoming message: the request must target a valid QPN, a valid MR with a correct *rkey*, and a valid PSN (see Section III-E), and various works have demonstrated that it is only moderately complex to obtain or predict all those numbers [90], [138], [142].

Overall, these vulnerabilities may lead to attacks with potentially dramatic consequences, ranging from silent information

leakage to code injection in remote memory areas [143]. To make the situation worse, attackers located on the same host of an RDMA endpoint, such as cloud users that escaped their confinement in VMs or containers [148], can potentially impersonate the legitimate user of an established connection and access remote memory on any machine of the network [143].

Unfortunately, standard mechanisms to guarantee the confidentiality, integrity, and authentication of data and metadata (e.g., the *TLS* protocol) are not currently supported by RDMA NICs, because they are not part of the standard specification. Software-based implementations of such functions would be impractical, both because one-sided operations bypass the remote CPU, making it impossible to transparently decrypt incoming data, and because the associated performance overhead would negate the benefits of zero-copy transfers [140]. Thus, novel proposals mainly focused on introducing the missing hardware support either on NICs or on ToR switches.

1) Hardware-Based Approach: RFC5042 [146] suggests the use of IPSec [149] to encrypt and authenticate RDMA traffic over IP networks, although it is not straightforward to integrate IPSec into existing RDMA protocols: RDMA headers are not visible outside the IPSec encapsulation and thus network equipment cannot distinguish packets directed to different QP endpoints, making it impossible to achieve a QP-level authentication and thus to avoid packet injection or spoofing [138]. Furthermore, IPSec adds a significant processing overhead which becomes not negligible at RDMA speed [140], [143]. Nevertheless, major NIC vendors have recently started to support IPsec-based encryption of RoCE packets [78]. As a workaround to the problem of QP visibility, the NIC has to store information about QPs and keep an association between them and the corresponding IP address. Alternatively, Kornfeld Simpson et al. [142] suggest the use of the TLS protocol for iWarp, which is based on TCP. For RoCEv2, which relies on UDP, they propose *Datagram TLS* (DTLS) [150] as a solution to provide the same confidentiality and integrity guarantees.

However, both IPSec and DTLS would apply only to IP-based implementations of the RDMA specifications, excluding Infiniband deployments. Although Infiniband equipment is less common in cloud data centers, approaching the problem of RDMA encryption at a higher abstraction level, directly in the RDMA specification, would be ideal, as the solution would apply to all the implementations. To this end, Lee et al. [140], [141] first attempted to introduce changes to the Infiniband specification. In particular, they proposed a form of RDMA packet authentication that replaces the Invariant CRC field with a message authentication code (MAC).

More recently, sRDMA [143] proposed to extend the specification to include built-in encryption and authentication. In particular, that work introduces a new RDMA mode called *Secure Reliable Connection* (SRC) that adopts symmetric cryptography to protect RDMA traffic. The choice of symmetric cryptography minimizes the computational overhead compared to asymmetric cryptography, reducing it by three to five orders of magnitude, and thus is suitable for RDMA networking. Conceptually, a symmetric key is associated to each QP at initialization time together with a protection algorithm. This key will be used by the NIC to perform cryptographic operations. sRDMA also requires a second change to the specification, namely the addition of a new packet header called *Secure Transport Header* (STH). This header contains a message authentication code useful to provide the integrity of the payload, and must be included in any RDMA request.

With the new SRC mode and the new header, sRDMA is able to guarantee both confidentiality, integrity, and authentication for RDMA traffic with minimal performance overhead, as cryptographic operations are performed directly by the NIC. However, this solution assumes that the host trusts the NIC and the internal bus that connects the NIC to the host. If that is not the case, additional security mechanisms should be considered.

2) In-Network Approach: Following the trend discussed in Section VII, modern networking hardware increasingly supports programmable data planes both in switches and NICs [128], [129]. Under a security perspective, this approach makes it possible to define custom datapath protection mechanisms that operate at hardware speed.

Simpson et al. [142] comment that DTLS protection (encryption and authentication) could be integrated in custom bump-on-the-wire devices like those introduced in Catapult and AccelNet [80], [81], but these are highly customized devices. Instead, Bedrock [135] enables in-network source authentication for RDMA traffic using the P4 scripts [130], [131]. By enforcing security in ToR-based switches, this solution is transparent to the end-hosts software and hardware and does not require any protocol or specification change. When end-host actions are required, Bedrock resorts to the eBPF framework [31] to transparently inject behavior to, or extract information from, the Linux kernel.

In particular, Bedrock obtains source authentication by binding RDMA endpoints to *network invariants* of the data center topology: as part of the packet processing pipeline, the ToR switch only allows packets which are directed to a given QP, which come from the expected IP address, and from the right switch ingress port ID. To neutralize same-host attacks, the sender process ID is also considered. However, Bedrock relies on the strong assumption that the cloud provider is trusted: under weaker models, such as that adopted in sRDMA [143], this solution becomes ineffective against rogue cloud providers.

Overall, the popularity of RDMA in cloud environments will soon make confidentiality and integrity of RDMA traffic a top priority for cloud providers. We expect more solutions like sRDMA to call for an extension of the Infiniband specification, so that all the RDMA protocols can include adequate built-in protection mechanisms. However, standardizing those proposals will likely take a long time as it requires a broad support from the whole community. A short-term mitigation may come from proposals like the use of DTSN or IPSec for RoCEv2 traffic, which apply to the most popular RDMA protocol implementation (RoCE) and are starting to get support from hardware vendors. Even more promising are in-network solutions like Bedrock, which can transparently improve security of existing applications and deployments, although this approach requires users to trust cloud providers.

B. Access Control

To enable *NAaaS*, the resources managed through acceleration techniques, such as the memory areas for remote access exposed by RDMA, should be included into the existing cloud access control strategies. However, that integration is particularly difficult to accomplish. In cloud data centers, access control is managed through software-based mechanisms like Access Control Lists (ACL), dynamically reprogrammable lists of permissions (*policies*) that specify which users or system processes can access specific resources and which operations they are allowed to perform. In contrast, the resources associated with network acceleration tend to be managed with native access control mechanisms (see Section III-E3), which in the case of RDMA are fundamentally *insecure* and too *rigid* to be integrated with cloud ACLs.

Nevertheless, access control is essential in cloud infrastructures and any viable solution for *NAaaS* should support secure and flexible mechanisms for that. Hence, researchers explored the possibility to apply, extend, or modify ACL-based techniques to include RDMA resources.

In this Section, we first summarize the major vulnerabilities of the *native RDMA mechanisms* for access control, showing that they are insecure and easily bypassed in a cloud environment. In the second part, we summarize the possible short-term actions that can be immediately adopted to mitigate their impact. Finally, in the third part we discuss the longer-term solutions that have been proposed to overcome the rigidity of these mechanisms and to integrate them into existing cloud access resource control policies, such as ACLs.

1) Security Issues of RDMA Access Control Mechanisms: Recent contributions [90], [135], [138], [142] discovered several vulnerabilities in the built-in RDMA access control mechanisms introduced in Section III. First, these mechanisms make use of insecure tokens, not only exchanged in plaintext on the network, but also *easy to predict*. For example, an attacker could simply guess the correct values of some parameters necessary for a one-sided access to a remote memory area (memory keys, QPNs) because these are generated by the NIC according to a predictable pattern [90], [135], [138], [142].

Secondly, several RDMA-based systems, proposed by industry or academia, tend to *misuse the available access control mechanisms* in favor of improved performance, making predictability even easier. For instance, those systems tend to use a single protection domain for all applications on the same host; do not use memory windows to restrict QP access to MRs; and use a default value for PSN initialization (*fixed-starting PSN*) [135], [138]. As a consequence, an attacker might legitimately require its own QP, register a new MR, and use the associated parameters to guess the corresponding values for co-located applications.

Finally, existing systems tend to allocate objects at *consecutive addresses* starting from a randomly-assigned base provided by the operating system [138]. Hence, although it is hard for attackers to predict the exact location of objects in remote memory areas, this task becomes much easier if they know the memory position of even just one of them, as they can guess the relative position of the others. This problem is exacerbated by the common adoption of the Implicit On-Demand Paging (ODP) technique for improved performance. Instead of registering many small memory portions for DMA access, which would introduce non-negligible performance overhead, developers often use this feature to register for DMA the complete memory address space of a process. The security implication of this choice, however, is that an attacker can potentially obtain access to the entire process memory [138].

2) Short-Term Mitigation Proposals: Despite initial efforts, such as avoiding the plaintext transmission of memory keys [140], [141], the security concerns about the native RDMA access control mechanisms have remained unaddressed for years. The seminal ReDMArk [138] paper recently proposed a set of possible short-term mitigation mechanisms that can be adopted without disrupting existing RDMA deployments. Overall, these mechanisms aim at making harder for attackers to guess either memory access tokens or the memory addresses of RDMA objects, and are intended to be used in combination with encryption (see Section VIII-A).

First, ReDMArk aims at *reducing the predictability* of the tokens involved in the communications, especially memory keys and QPNs, by randomizing their generation. Second, it also recommends that RDMA developers start to *make a meaningful use* of the existing mechanisms, in particular of PDs and MWs, and randomize the location of RDMA objects in their memory address space. ReDMArk also suggests to require code attestations for any library loaded with application binaries, because malicious versions could have been silently injected by attackers accessing memory without authorization.

3) Integration With Cloud Access Control Mechanisms: The research on the integration of network accelerators into cloud access control mechanisms mainly focused on two aspects: preventing unauthorized traffic injection into the physical network fabric, and managing RDMA resources with ACLs.

The first line of research aims at enabling even accelerated traffic to be managed through *network security rules*. In commodity virtual networks, this problem is generally solved by making a virtual switch apply security rules chain to each packet. When that is not possible for accelerated traffic, such as for *hybrid* and *hardware* virtualization strategies (see Section VI-B3), a *per-connection* approach can be adopted as proposed by MasQ [54], VSocket [57], and SocksDirect [58]: before creating an RDMA connection, a corresponding TCP connection should generally be created first. If that is allowed by the current security rules, then also the RDMA connection obeys those security rules for connection creation. Should a security rule change during an active transmission such that the transmission is no more allowed, the corresponding QP can be moved to a connection error, just like when a network error occurs [54]. This approach has the advantage of being immediately available for existing RDMA deployments, but it is far from ideal as it maps network permission changes to network errors.

In the context of live resource migration (see Section VII-C), MigrOs [137] proposes an extension to the RoCEv2 protocol to add new states to the QP state machine: although that proposal does not consider security, the same strategy could be extended to include a specific state to signal that a connection has been interrupted because of changing security rules. However, the overall effort of these works is mainly focused only on protecting network access, whereas access to RDMA local objects such as QP or MR is still demanded to the basic RDMA access control mechanisms.

Within the second line of research, Bedrock [135] proposes a finer-grained access control for RDMA resources designed to expose software-programmable ACLs customizable with network or application-specific policies. This work proposes to offload the enforcement of access control to programmable data planes (using eBPF [31] and P4 [130], [131]) located on ToR switches [128], [129].

With this *in-network* approach, cloud providers and users obtain the possibility to control, modify or customize ACL policies for CPU-bypassing traffic without performance penalties due to software intervention and without modifications to existing applications. User configurations are forwarded to a daemon on the closest ToR switch, which translates them to a P4 program to enforce access control in the data plane. Access policies can take into account various RDMA object parameters, such as memory ranges, opcodes, QPNs, but also process ids for local communication. ACLs are implemented in the form of match/action tables on the switch, maintained in a compressed form to reduce memory overhead. Advanced ACL configuration is also possible by taking into account real-time monitoring data, as we discussed in Section VII-A.

The Bedrock solution cannot mitigate all the current limitations of RDMA access control, e.g., local resource exhaustion attacks, because it monitors traffic in switches [151]. Nevertheless, the integration of a software-based ACL configuration mechanism for RDMA represents a major contribution to the goal of *NAaaS* in cloud infrastructures.

C. Isolation

Under the original security model of acceleration technologies, physical resource isolation guarantees no interference among untrusted users. Because this strategy is not economically sustainable for cloud data centers, providers are looking for different isolation techniques to guarantee that users cannot interfere with each other, and the solutions discussed in Sections VIII-A and VIII-B about confidentiality and access control for RDMA networks answer the primary need of ensuring traffic isolation among different tenants.

In this Section, we report about *side-channel attacks*, which are forms of violation of the tenant isolation, and report the strategies proposed to prevent them. In a side-channel attack, an attacker might obtain information about a running program by probing the CPU cache (or other caching systems) and observe how it behaves in consequence of the other program actions. Although malicious users cannot directly read data belonging to the target program, they can still obtain so-called *side information* such as the application access patterns to certain data, which can be the basis for more complex attacks.

Recent work identified at least two possible side-channel attacks to RDMA applications, both related to the use of the processor caches and one-sided operations: Pythia [144] and NetCAT [145]. Pythia [144] detected a set of side-channel attacks that allow a malicious user to learn how clients on other machines access data exposed by a remote RDMA service. This attack, invisible to either the victim client and server due to the use of *one-sided* RDMA operations, is based on a evict+reload strategy [152] on the server RNIC. RNICs cache *hot* metadata for performance reasons, but in case of cache misses they must fetch information from main memory, adding extra latency to that request. Hence, a malicious user might try to guess whether a victim is using a specific set of data by forcing specific metadata to be cached on the NIC and measuring differences in the server response time. For example, Pythia demonstrates a successful detection of key-value pair access patterns on a storage system [153].

NetCAT [145] unveiled a different form of side-channel attack that targets the use of hardware-based performance optimizations, in particular of the Intel Data-Direct I/O (DDIO), a feature available on recent Intel processors that enables peripherals (e.g., an RNIC) to perform Direct Cache Access (DCA) instead of Direct Memory Access (DMA). Using a similar approach to Pythia, NetCAT allows a malicious user to detect the activity of a remote processor, as well as of the other network clients interacting with it. To demonstrate the effectiveness of this technique, NetCAT performed a keystroke timing attack, recovering the words typed on a SSH session by a remote client with the target server. In this case the reconstruction of data access patterns from other clients allows attackers to leak sensitive information by just acting as legitimate clients of the same remote service.

As a general consideration, the risks associated to side-channel attacks are not particularly high in a noisy environment with many co-located applications and many users, such as a public cloud infrastructure [54]. Hence, although these attacks are certainly a source of concern for cloud providers, the urgency of mitigation for cloud provider is much less than the previously considered vulnerabilities [138], [142].

Nevertheless, possible mitigation strategies were proposed even against this kind of attacks. Pythia proposes a form of traffic monitoring to identify and block malicious traffic that tries to launch side-channel attacks [144], and Bedrock [135] includes that capability into its in-network security suite for RDMA networks. Other possibilities include adding intentional random latency overhead on the network, disabling DDIO, or partition the processor cache per-user [144], [145]. However, the latter approach contrasts with the goal of maximize network performance. Therefore, as all these works suggest, the most effective approach would be to introduce countermeasures directly in the hardware equipment.

D. Key Takeaways

This Section reviewed the major contributions to the investigation of the security properties of network acceleration

technologies and of the security implications of their integration in cloud platforms in terms of *confidentiality, integrity, and authentication*, *access control*, and *isolation*. Whereas for XDP-based and DPDK-based applications these aspects are either not under the provider responsibility or quite straightforward to implement, the picture we drew for RDMA is quite concerning: that technology was designed for a security model based on physical resource isolation. Only recently researchers have started to assess the vulnerabilities related to its adoption in shared cloud infrastructures.

When considering possible solutions, the majority of these works agrees that hardware-based approaches (e.g., embedded in the NIC) represent the only practical option to add security mechanisms to network acceleration technologies, in particular if compute-intensive tasks are involved (e.g., data encryption). In-network strategies, such as ToR-enforced policies are also considered effective, especially for access control tasks.

Overall, the proposed solutions are either temporary mitigation actions or claims for longer-term changes. Short-term actions mainly consist in recommendations for developers to properly use the few existing security mechanisms, for instance by randomizing sequence numbers and memory addresses to reduce predictability. Conversely, longer-term solutions involve changes to protocol specifications (e.g., Infiniband [33]) to provide built-in support to all these three security aspects and to favor their hardware implementation by manufacturers. Nonetheless, security remains one of the main open challenges for *NAaaS*, as we discuss in Section IX.

IX. Open Challenges and Future Directions

In the previous Sections, we have provided a literature overview about the integration of network acceleration technologies in cloud platforms, with the primary objective of presenting a cohesive and integrated perspective on this challenge and to overcome the fragmentation resulting from isolated contributions. The majority of the surveyed studies focused on novel concepts and design principles, whereas the realization of production-grade solutions that can efficiently and securely accelerate overlay networks in large-scale cloud infrastructures is still not answered, especially for those technologies that partially or totally offload network operations to specialized hardware devices, such as RDMA.

In this Section, we first summarize the lessons learned from our literature overview (Section IX-A). Then, we present the key open challenges that currently obstacle the development of mature industry-grade solutions (Section IX-B). Finally, we conclude by discussing the most relevant future research directions that need to be investigated to achieve the maturity of the *NAaaS* paradigm (Section IX-C).

A. Lessons Learned

In this survey, we considered four orthogonal aspects of cloud networking (*access interfaces*, *virtualization techniques*, *serviceability*, and *security*). For each aspect, we distinguished *hardware*, *software*, *hybrid*, and *in-network* approaches to integrate acceleration technologies. As a preliminary step before introducing our insights about open challenges and future research directions, in this Section we observe that these four cloud aspects are interconnected and reciprocally influenced. Hence, we first need to discuss whether the principles and the solutions proposed in the surveyed literature can be combined toward an integrated evolution of the cloud ecosystem.

We begin by observing that many interfaces for accelerated networking (Section V) assume the availability of a *virtual accelerated NIC* (e.g., a virtual RDMA NIC), or its equivalent for software-based techniques (e.g., a special PMD for DPDK). Such assumption has a significant impact on both the I/O virtualization and serviceability dimensions: if applications did not need direct visibility of the NIC, providers would be able to either use standard solutions or design more efficient ones to support accelerated cloud networking [57], [112], [113].

Similarly, different approaches to I/O virtualization impact the implementation options for virtual switches, core components for network virtualization, serviceability, and security: for instance, offloading I/O virtualization to hardware entails a hardware-based virtual switch, thus excluding the software alternatives discussed in Section VII. The same holds for the other dimensions: different serviceability techniques, e.g., for monitoring and logging, influence the granularity and the timeliness of security operations such as ACL enforcement (see Section VIII).

Overall, the insight that clearly emerges from our comprehensive survey effort is that the selection of an approach for a certain cloud aspect practically affects all the others. In particular, we highlight a crucial trade-off between the network performance offered to cloud customers and the ease of implementation of *NAaaS* solutions for providers. Software-based solutions, although still improving performance over standard mechanisms, tend to introduce a significant overhead on the native acceleration technologies, making them less appealing to cloud users. Conversely, hardware-based solutions preserve the native performance and are more attractive for customers, but the limitations we discussed in terms of programmability and flexibility make them challenging to adopt by providers.

An alternative approach to pure software and pure hardware approaches that emerged from our review is the combination of hybrid and in-network techniques. This option offers an alternative balance between the flexibility of a software-based control plane and the performance of an accelerated data plane: in-network solutions can recover the serviceability and security mechanisms bypassed by the hybrid solutions. However, that combination would be a short-term solution to a fundamental shortcoming of the network acceleration ecosystem: the lack of proper *cloudification* support from hardware manufacturers.

In light of these considerations, in the remainder of the Section we will discuss how a longer-term approach to this trade-off must be designed through a joint effort from different actors, including cloud providers and hardware manufacturers.

B. Open Challenges

The integration of network acceleration technologies into existing cloud infrastructures, shared among multiple untrusted

TABLE VIII
A SUMMARY OF THE OPEN CHALLENGES FOR THE INTEGRATION OF NETWORK ACCELERATION IN CLOUD PLATFORMS

Open Challenges
Network Acceleration Interfaces The definition of the right abstraction level for a set of standard, easy-to-use primitives to uniformely access accelerated networking.
Virtualization Techniques The definition of a standardized approach to the virtualization of network acceleration that takes into account not only efficiency and programmability, but also open technology availability and economical factors.
Serviceability The definition of new solutions for the practical scalability of network acceleration technologies and their deployment in large-scale clouds.
Security The definition of *secure-by-design* acceleration options, starting from the building blocks already in-place, through the cooperation of stakeholders.

tenants with demanding requirements of flexibility and security, is the most significant challenge for next-generation cloud platforms. The reviewed literature has introduced solutions in the form of design principles and prototypes and the first commercial products are starting to emerge [94], [95], [96]. However, the broad application of these innovations to real-world infrastructures presents several additional technical problems, still largely unaddressed. As a result, the industry has not yet achieved production-ready, large-scale solutions that can leverage *NAaaS*. Here we summarize the most relevant technical challenges that must be overcome to achieve this goal. We organize our discussion in four paragraphs, corresponding to the four aspects of cloud computing considered in this survey.

1) Network Access Interface: Although the POSIX APIs are pervasively adopted in any application domain and offer a standard and easy-to-use interface, they are also designed on opposite principles than modern network acceleration technologies, thus making them difficult to adapt to the high performance of these options (Section V). However, the native interfaces of these technologies do not currently offer an equally easy-to-use programming option. The proposals reviewed in Section V-C represent attempts in this direction, but today, they are still at the level of academic proposals and are a long way from becoming mature products. *Therefore, we consider as an open research challenge the definition of the right abstraction level for a set of standard, easy-to-use primitives to access high-performance networking.*

2) Virtualization Techniques: We surveyed several proposals on the I/O and network virtualization of accelerated networking, all striving to balance the need to preserve the native performance with software-like programmability. This trade-off is especially evident when offloading network operations to hardware devices, but even the performance of software-based technologies such as XDP and DPK risks to be jeopardized by inefficient virtualization techniques.

From our discussion, none of the proposed virtualization solutions emerged as a clear winner. Approaches based on software communication channels and virtual switches introduce CPU overhead into the data path, slowing down performance and wasting valuable resources. Hybrid approaches reduce this overhead but also the providers' control on the data plane, which would likely be unacceptable in public clouds. Both the above solutions are cost-effective and retain the flexibility advantages of a software-based control path, but at the price of a significant performance penalty.

On the contrary, solutions based on specially-designed hardware, such as AccelNet [81] or AWS Nitro [126], offload the data plane operations to a specialized hardware device while also providing a high degree of control and programmability. As a consequence, these solutions are already part of different commercial services such as DPDK-enabled VMs [94], [95], [96]. However, these approaches require a high upfront investment for research, development, and deployment that only major providers can afford, and propose highly-customized solutions difficult to generalize. *Ultimately, we believe that further research should be dedicated to the definition of a standardized approach to the virtualization of network acceleration that takes into account not only efficiency and programmability, but also open technology availability and economical factors.*

3) Serviceability: In Section VII we extensively commented about the key role of *virtual switches* for the provisioning of serviceability aspects. In this paragraph we consider the *scalability* of these devices, which the reviewed literature do not systematically consider despite being a known concern in any data center setting [26], [54], [103], [120].

A fundamental issue arises when virtual switches are implemented by hardware devices for the sake of performance: these devices cache part of the connection state needed for serviceability directly in the *on-board device memory*. Because that memory is typically limited, also the total number of manageable connections is limited. This issue is exacerbated in cloud settings, where a device on the same physical host would be shared by multiple users. Existing countermeasures proved overall ineffective: for example, sharing on-board connection descriptors among many CPU cores might increase the number of managed connections, but it also significantly reduces performance and network isolation [98], [103], [132]. Even worse, the scarce scalability of these devices represents also a security vulnerability (see Section VIII).

Nevertheless, several proposals for NAaaS increase the amount of per-connection data to store on NICs, for example to better support network virtualization or information confidentiality. The same considerations apply to the *smart switches* adopted to carry on in-network packet processing. While some researchers are confident that this problem will eventually be solved through the continuous improvement in NIC hardware [154], [155], other studies question this argument, noting that such improvement is in fact much slower than expected [103]. *Overall, we believe that more research is needed to make network acceleration technologies really scalable, in particular in the context of cloud platforms.*

4) Security: In Section VIII, we separately considered software-based (e.g., XDP, DPDK) and hardware-based network acceleration technologies (e.g., RDMA). The former generally adopt standard protocols and cloud mechanisms to guarantee communication security, or provide their own

libraries when these cannot be reused. Instead, the latter technologies are generally based on a different communication model and on different hardware devices, for which historically security has not been a key design principle.

In particular, we drew a concerning picture of RDMA security. Recent studies found that the built-in security mechanisms of RDMA are inadequate for the cloud security model where multiple untrusted users share the same physical infrastructure. Furthermore, the few available security mechanisms are often misused by application developers for performance gains.

Consequently, security concerns pose a significant obstacle to the widespread availability of *NAaaS*, as the risk of untrusted users accessing remote memory is too high. Although we lack insights into how providers offering *bare-metal* RDMA instances are dealing with this issue, it is likely that they isolate each user within a secure environment. After the initial excitement surrounding the performance potential of network acceleration technologies, which led to a lack of attention to security aspects, the recent publication of the first systematic security assessments demonstrates an increasing maturity of the network acceleration ecosystem. Although only few proposals have been advanced so far, *hardware-based [143] or in-network [135] approaches might eventually evolve toward economically sustainable solutions: the building blocks for high-performance, secure accelerated networking are in place, although it involves the joint cooperation of several stakeholders as we further comment in the next Section*.

C. Future Research Directions

The increasing availability of powerful network acceleration technologies as a commodity holds the potential for a breakthrough in the communication domain. The possibility to significantly upgrade conventional communication infrastructures with high-performance networking options opens new perspectives for customized and real-time decision-making, innovative adaptive services, and advanced forms of automation in several application domains. At the same time, the success of expensive, specialized but fragmented technologies is leading to a broader *decline of general-purpose technology* [12].

Embracing this trend, we believe that an integration of such specialized technologies *as a service* into next-generation cloud platforms will play a crucial role in avoiding an excessive fragmentation of the networking landscape and, at the same time, in broadening the access to these modern options. The availability of specialized, heterogeneous technology through a standard, easily accessible, and widely used computing paradigm would be beneficial for many stakeholders. On the one hand, it would enable organizations to access the increasingly heterogeneous hardware landscape through a homogeneous point of access, without committing to massive upfront investments. On the other hand, this integration would also give cloud providers the economical power to drive the evolution of hardware accelerators according to their needs.

Consistently with these considerations, we believe that future research on *NAaaS* will need to investigate four main directions, summarized in Table IX. First, the extension of the cloud paradigm outside data centers, discussed in Section II-B, holds the potential to significantly expand the customer base for *NAaaS*, which could be used as a high-performance support for next-generation 5G and beyond telco networks. Second, emerging cloud service models, such as *serverless computing* [156], [157], promote a radical separation between the user code and the underlying resources, thus potentially enabling providers to offer acceleration options without giving users direct visibility of their infrastructure. Third, the fast-paced evolution of hardware accelerators is blurring the distinction between different forms of I/O. New devices such as Data Processing Units (DPUs) go even beyond the concept of *NAaaS* by offloading any I/O operation, toward the more general idea of *Accelerated I/O as a Service*. Finally, we observe that a higher degree of maturity of *NAaaS* solutions can be achieved only through the cooperation of all the involved stakeholders, including cloud providers, telco operators, hardware manufacturers, and standardization bodies.

TABLE IX
A SUMMARY OF THE FUTURE RESEARCH DIRECTIONS FOR THE INTEGRATION OF NETWORK ACCELERATION IN CLOUD PLATFORMS

Future Research Directions
Extension of NAaaS solutions outside data centers (1) A careful integration of *NAaaS* solutions with the standard interfaces and specifications of next-generation networks (5G and beyond). (2) A systematic consideration of the implications of *NAaaS* deployment outside data centers, with particular regard to energy consumption aspects.
Integration with emerging cloud service models New service models, such as serverless and Function as a Service (FaaS), reduce the visibility cloud users have on the infrastructure, thus potentially easing the adoption of accelerated networking in public cloud platforms.
Extension of the *NAaaS* approach to other forms of I/O Different forms of I/O memory (persistent memory, NIC memory, onboard device memory) are increasingly managed together by new accelerators, such as Data Processing Units (DPUs). A framework to access these I/O capabilities *as a Service* in public clouds is yet to be defined.
Increased cooperation among *NAaaS* stakeholders An active cooperation among all the involved stakeholders (researchers, hardware manufacturers, telco operators, cloud providers) is necessary to ensure the availability of *NAaaS* in next-generation cloud platforms, within but also outside large-scale centralized data centers.

1) Extension of NAaaS Solutions Outside Data Centers: The majority of the contributions surveyed in this work focused on centralized data centers. However, as discussed in Section II-B, in the last decade cloud computing has embraced a decentralization trend, making cloud resources available within the infrastructure of telco operators and even at the network edge.

In the telco ecosystem, *NAaaS* has the potential to complement NFV solutions and become a crucial enabler for the emerging services in 5G and beyond mobile networks, such as NetApps (see Section I-A). That is because *NAaaS* solutions minimize the performance overhead of the cloud mechanisms used by virtualized applications to access network acceleration technologies, even when special hardware devices are involved. With *NAaaS*, virtual network functions, as well as any application deployed on the telco infrastructure, could more efficiently access the underlying equipment. That would more effectively separate the software functions from the hardware implementing them: for instance, a *NAaaS*-based solution

would greatly simplify the development of novel solutions for the disaggregated Radio Access Network (RAN) [39].

However, adopting the techniques reviewed in this survey on the telco edge would require, in practice, a careful integration with existing standard interfaces and specifications that currently has not yet been explored in the literature. We believe that *future research on NAaaS should focus more on these extensions of cloud computing outside data centers, also because many performance-demanding applications are increasingly hosted there. That would have the effect of widening the customer base for these services, making it even more sustainable for providers to design mature technical solutions, as we discuss in the next paragraph.*

At the network edge, it is not uncommon to find small-scale data centers equipped with fairly powerful resources, qualitatively comparable to those in centralized clouds. Even there *NAaaS* approaches could be effective, because many of issues of public cloud deployments (in particular, scalability and security isolation) are generally less concerning for those environments. However, those settings typically have different constraints than data centers and might bring out different concerns, such as *energy efficiency* [158], [159]. Although executing functions on specialized hardware requires less energy than on general-purpose processors, the availability of hardware accelerators at the network edge is not always guaranteed. Conversely, software-based acceleration technologies might require *polling* techniques that induce a high resource usage [14]. Therefore, *the implications of NAaaS outside data centers should be carefully considered by future research, with particular regard to the energy consumption aspects that will assume increasing importance in next-generation edge computing infrastructures.*

2) Integration With Emerging Cloud Service Models: In Section IX-B, we identified the lack of a uniform, easy-to-use interface as an open challenge for *NAaaS*. To this end, the emerging concept of *resource disaggregation*, which proposes to radically separate user code from the underlying hardware, represents a promising research direction [160], [161].

In cloud scenarios, this trend led to the definition of *serverless* computing and the *Function as a Service* (FaaS) model: users submit code as stateless *functions* to be executed on-demand, thus completely delegating to providers the control of resource management. The FaaS model yields several advantages, such as scale-to-zero capabilities and the possibility to create applications as function pipelines (*function chaining*) [156], [157].

However, mainly for security concerns, current platforms still execute user functions within isolated environments, such as containers and lightweight VMs, just like in other cloud models. In addition, the short life of functions worsens the obstacles for *NAaaS*: not only do functions still use standard I/O interfaces that require a NIC, but the overhead of communication setup (e.g., for data exchange with databases) becomes dominant [162], [163].

Recent work is starting to promote a more radical separation between the computation and communication of functions (*computation-centric networking*) [110], [164]. The key insight of this approach is to let users explicitly declare their communication flows but to leave to providers the implementation of the actual I/O operations. In the edge cloud, a similar concept has been recently introduced to decouple applications from special-purpose acceleration interfaces [113]. Because user code does not need to directly access I/O resources, *providers can then enforce lighter forms of isolation and transparently optimize I/O operations, without designing complex solutions for virtualization, serviceability, and security. Hence, we believe that this approach, and more generally the serverless paradigm, is a promising direction for researchers to design an easy-to-use, uniform point of access to I/O acceleration in next-generation cloud platforms.*

3) Extension of the NAaaS Approach to Other Forms of I/O: The hardware acceleration landscape is rapidly evolving. In this survey, we have focused on network acceleration options as they represent the most relevant example of this trend. However, several applications increasingly need efficient ways to handle any form of I/O. For instance, the AI/ML applications that are popular today in centralized clouds typically employ GPUs for parallel processing, requiring frequent and heavy data transfers between CPU memory, GPU memory, and persistent storage [15]. Similar communication patterns also emerge in the edge cloud, whereby data produced by local devices (e.g., cameras) are increasingly processed on-site to obtain low-latency responses [28], [165]. Modern hardware-based techniques like NVLink and GPUDirect already enable offloading those data movements, even between GPUs and remote storage devices via direct RDMA transfers [166].

To further optimize data movement, the concept of Data Processing Units (DPUs) was recently introduced to put together different hardware accelerators for I/O operations (network and storage) [167], [168]. With dedicated CPU cores embedded on the same board, these devices completely offload I/O operations from the general-purpose CPU that can be preserved for other types of processing. The potential benefits of DPUs are promising especially for cloud providers, which can use them to implement the virtualization, serviceability, and security aspects that we have previously discussed without spending CPU cycles for them.

However, due to the novelty of these options, there are several aspects still to be investigated before their successful adoption in cloud platforms. For instance, we still lack suitable software frameworks to easily interface with these devices, while it is still unclear which I/O virtualization techniques are appropriate for them. *Therefore, we suggest that further research should be dedicated to the proper integration of these new devices within cloud platforms. Given the extended capabilities of these hardware options, we envision that they can be employed to offer users not only NAaaS but the even more general option of Accelerated I/O as a Service.*

4) Increased Cooperation Among NAaaS Stakeholders: We believe that only the close cooperation among all involved stakeholders can solve the open challenges that still clash with the effective integration of network acceleration technologies into the cloud paradigm. Cloud providers, telco operators, hardware manufacturers, and standardization bodies are called to cooperate and re-design their products and services to support the needs of next-generation cloud platforms.

On the one hand, *cloud providers and telco operators need to properly integrate network acceleration technologies with their infrastructures* by enforcing a more radical separation between control and data plane. The control plane, running on CPUs, must orchestrate a data plane that should be almost completely offloaded to acceleration technologies, either in software (XDP, DPDK) or in hardware (RDMA SmartNICs, programmable hardware such as FPGA). On the other hand, to preserve programmability and flexibility, *hardware manufacturers should include an explicit support for the cloudification of their products*, by enhancing device features with built-in support to crucial aspects for cloud infrastructures, such as live migration, network virtualization, and security.

For example, considering RDMA, in recent years the definition of the RoCEv2 protocol [35] and its support by major hardware vendors triggered a process of wide dissemination of that technology even in Ethernet-based data centers. *We believe that a similar change is now necessary to properly support network acceleration technologies in cloud platforms*.

Most research we reviewed propose changes to protocols, semantics, interface, or objects, sometimes with explicit reference to a needed change in the RDMA specification and in its hardware implementations [137], [138]. At the current early adoption stage of acceleration technologies in cloud infrastructures, we believe that such changes are still sustainable, technically and economically, and mainly motivated by an exponential growth in user demand for these services. Hyperscaler cloud service providers are already co-designing custom hardware devices to better interoperate with their software infrastructure [81], [126]. These solutions already enable some of them to offer *NAaaS* through software techniques like DPDK [94], [95], [96], but security concerns still prevent them from enabling the full potential of hardware-based solutions (see Section VIII). To solve these issues, the convergence of several stakeholders, including researchers, hardware manufacturers, telco operators, and cloud providers on the definition of standard solutions is necessary. Despite the different economical and industrial interests, *that cooperation is a necessary and unavoidable step to ensure and effective availability of NAaaS in next-generation cloud platforms, within but also outside large-scale centralized data centers*.

X. Conclusion

The increasingly pervasive availability of connected devices in virtually any application domain is pushing the well-established cloud computing model to show its inherent limitations. In the last decade, cloud workloads have progressively shifted toward *interactive, data-intensive applications* that need to move huge volumes of data coming from sparse devices and to process them within tight deadlines. To meet these increasingly demanding performance requirements, developers have started to rely on *network acceleration technologies*, which, often by means of specialized hardware, can substantially outperform standard networking stacks and protocols.

However, the adoption of network acceleration in cloud platforms faces several technical challenges, because the nature of acceleration technologies cannot easily conciliate with the architecture of standard cloud infrastructures: usually, these options bypass the existing software stacks and create a trade-off between the typical cloud flexibility and network performance. This survey reviewed the literature on *Network Acceleration as a Service* and focused on the goal of integrating accelerated networking into the cloud computing platforms, adopting the popular Remote Direct Memory Access (RDMA) as the reference communication model.

We discussed the existing proposals by systematically organizing them in a taxonomy based on the aspects of cloud networking most impacted by this integration: *network access interfaces*, *virtualization techniques*, *serviceability*, and *security*. For each dimension, we presented different approaches ranging from software-based solutions, which sacrifice performance in favor of flexibility, to hardware-based solutions, which maximize performance at the price of reduced flexibility. Finally, we identified the key challenges that providers still need to address to effectively achieve the goal of enabling *NAaaS* in cloud platforms and discussed the crucial research directions that must be investigated for a mature and widespread adoption of this concept.

Ultimately, we envision that the integration of network acceleration technologies will become essential for cloud providers to meet the increasingly demanding performance requirements of new *interactive, data-intensive applications* in several application domains. Although the literature reviewed in this survey mainly focused on centralized data centers, we believe that this integration will become paramount also for the cloud extensions outside data centers. In particular, the evolution of cloud-based architectures and technologies in support to the next generation communication infrastructures (e.g., 5G and beyond) will increasingly require *NAaaS* solutions.

Despite these compelling motivations, such integration will only be successful as a result of a joint effort from different stakeholders. On the one hand, cloud providers and telco operators should redesign their interfaces and infrastructures to enable scalable, efficient, and secure data paths. On the other hand, hardware manufacturers should enhance device features with built-in, low-overhead support to crucial aspects such as application live migration, network virtualization, and security. With this survey, we aimed to provide a comprehensive perspective in the direction of achieving that goal, by establishing a shared knowledge base toward the design of next-generation cloud platforms and communication infrastructures.

Acknowledgment

The authors would like to express their gratitude to the editor and to the anonymous reviewers for the careful and thorough examination of our manuscript and for their constructive comments and suggestions.

References

[1] L. A. Barroso, U. Hölzle, and P. Ranganathan, *The Datacenter as a Computer: Designing Warehouse-Scale Machines*. Cham, Switzerland: Springer, 2019.

[2] T. Erl, R. Puttini, and Z. Mahmood, *Cloud Computing: Concepts, Technology & Architecture*, 1st ed. Hoboken, NY, USA: Prentice Hall, 2013.

[3] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128610001568

[4] K. Birman, B. Hariharan, and C. De Sa, "Cloud-hosted intelligence for real-time IoT applications," *Oper. Syst. Rev.*, vol. 53, no. 1, pp. 7–13, Jul. 2019. [Online]. Available: https://doi.org/10.1145/3352020.3352023

[5] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Towards low-latency service delivery in a continuum of virtual resources: State-of-the-art and research directions," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 4, pp. 2557–2589, 4th Quart., 2021.

[6] E. Baccour et al., "Pervasive AI for IoT applications: A survey on resource-efficient distributed artificial intelligence," *IEEE Commun. Surveys Tuts.*, vol. 24, no. 4, pp. 2366–2418, 4th Quart., 2022.

[7] Z. Xiang, F. Gabriel, E. Urbano, G. T. Nguyen, M. Reisslein, and F. H. P. Fitzek, "Reducing latency in virtual machines: Enabling tactile Internet for human-machine co-working," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 1098–1116, May 2019.

[8] A. Garbugli, L. Rosa, L. Foschini, A. Corradi, and P. Bellavista, "A framework for TSN-enabled virtual environments for ultra-low latency 5G scenarios," in *Proc. IEEE Int. Conf. Commun.*, 2022, pp. 5023–5028.

[9] C.-X. Wang, M. D. Renzo, S. Stanczak, S. Wang, and E. G. Larsson, "Artificial intelligence enabled wireless networking for 5G and beyond: Recent advances and future challenges," *IEEE Wireless Commun.*, vol. 27, no. 1, pp. 16–23, Feb. 2020.

[10] Y.-X. Huang and J. Chou, "A survey of NFV network acceleration from ETSI perspective," *Electronics*, vol. 11, no. 9, p. 1457, 2022. [Online]. Available: https://www.mdpi.com/2079-9292/11/9/1457

[11] G. S. Niemiec, L. M. S. Batista, A. E. Schaeffer-Filho, and G. L. Nazar, "A survey on FPGA support for the feasible execution of virtualized network functions," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 1, pp. 504–525, 1st Quart., 2020.

[12] N. C. Thompson and S. Spanuth, "The decline of computers as a general purpose technology," *Commun. ACM*, vol. 64, no. 3, pp. 64–72, Feb. 2021. [Online]. Available: https://doi.org/10.1145/3430936

[13] B. Peccerillo, M. Mannino, A. Mondelli, and S. Bartolini, "A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives," *J. Syst. Archit.*, vol. 129, Aug. 2022. Art. no. 102561. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762122001138

[14] E. Freitas, A. T. de Oliveira Filho, P. R. do Carmo, D. Sadok, and J. Kelner, "A survey on accelerating technologies for fast network packet processing in Linux environments," *Comput. Commun.*, vol. 196, pp. 148–166, Dec. 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366422003917

[15] C.-H. Hong, I. Spence, and D. S. Nikolopoulos, "GPU Virtualization and scheduling methods: A comprehensive survey," *ACM Comput. Survey*, vol. 50, no. 3, pp. 1–37, Jun. 2017. [Online]. Available: https://doi.org/10.1145/3068281

[16] C. Bobda et al., "The future of FPGA acceleration in datacenters and the cloud," *ACM Trans. Reconfig. Technol. Syst.*, vol. 15, no. 3, pp. 1–42, Feb. 2022. [Online]. Available: https://doi.org/10.1145/3506713

[17] R. Buyya et al., "A manifesto for future generation cloud computing: Research directions for the next decade," *ACM Comput. Survey*, vol. 51, no. 5, pp. 1–38, Nov. 2018. [Online]. Available: https://doi.org/10.1145/3241737

[18] "Amazon AWS high performance computing." Amazon. Accessed: Apr. 4, 2024. [Online]. Available: https://aws.amazon.com/hpc

[19] "Microsoft azure high performance computing." Microsoft. [Online]. Available: https://azure.microsoft.com/solutions/high-performance-computing

[20] "Google cloud high performance computing." Google. Accessed: Apr. 4, 2024. [Online]. Available: https://cloud.google.com/solutions/hpc

[21] (Ethernet Alliance, Silicon Valley, CA, USA). *Ethernet Roadmap*. 2022. [Online]. Available: https://ethernetalliance.org/technology/ethernet-roadmap

[22] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. Amsterdam, The Netherlands: Elsevier, 2017.

[23] Q. Cai, S. Chaudhary, M. Vuppalapati, J. Hwang, and R. Agarwal, "Understanding host network stack overheads," in *Proc. ACM SIGCOMM Conf.*, 2021, pp. 65–77. [Online]. Available: https://doi.org/10.1145/3452296.3472888

[24] L. Barroso, M. Marty, D. Patterson, and P. Ranganathan, "Attack of the killer microseconds," *Commun. ACM*, vol. 60, no. 4, pp. 48–54, Mar. 2017. [Online]. Available: https://doi.org/10.1145/ 3015146

[25] A. Kaufmann, T. Stamler, S. Peter, N. K. Sharma, A. Krishnamurthy, and T. Anderson, "TAS: TCP acceleration as an OS service," in *Proc. 14th EuroSys Conf.*, 2019, pp. 1–16. [Online]. Available: https://doi.org/10.1145/3302424.3303985

[26] A. Dragojević, D. Narayanan, M. Castro, and O. Hodson, "FaRM: Fast remote memory," in *Proc. 11th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2014, pp. 401–414. [Online]. Available: https://www.usenix.org/conference/nsdi14/technical-sessions/dragojevi

[27] A. Kalia, M. Kaminsky, and D. G. Andersen, "Using RDMA efficiently for key-value services," in *Proc. ACM Conf. SIGCOMM*, 2014, pp. 295–306. [Online]. Available: https://doi.org/10.1145/2619239.2626299

[28] W. Song et al., "Cascade: A platform for delay-sensitive edge intelligence," 2023, *arXiv:2311.17329*.

[29] S. Jha et al., "Derecho: Fast state machine replication for cloud services," *ACM Trans. Comput. Syst.*, vol. 36, no. 2, pp. 1–49, Apr. 2019.

[30] A. Kalia, M. Kaminsky, and D. G. Andersen, "Design guidelines for high performance RDMA systems," in *Proc. USENIX Annu. Tech. Conf.*, 2016, pp. 437–450. [Online]. Available: https://www.usenix.org/conference/atc16/technical-sessions/presentation/kalia

[31] M. A. M. Vieira, M. S. Castanho, R. D. G. Pacífico, E. R. S. Santos, E. P. M. C. Júnior, and L. F. M. Vieira, "Fast packet processing with EBPF and XDP: Concepts, code, challenges, and applications," *ACM Comput. Survey*, vol. 53, no. 1, pp. 1–36, Feb. 2020.

[32] (Linux Found., San Francisco, CA, USA). *The Data Plane Development Kit*. 2023. [Online]. Available: https://www.dpdk.org/

[33] *InfiniBand Architecture Specification, Release 1.0*, InfiniBand Trade Assoc., Beaverton, OR, USA, 2000,

[34] T. von Eicken, A. Basu, V. Buch, and W. Vogels, "U-net: A user-level network interface for parallel and distributed computing," in *Proc. 15TH ACM Symp. Oper. Syst. Princ.*, 1995, pp. 40–53. [Online]. Available: https://doi.org/10.1145/224056.224061

[35] *Supplement to InfiniBand Architecture Specification, Release 1.2.2 Annex A16: RDMA Over Converged Ethernet (RoCE)*, InfiniBand Trade Assoc., Beaverton, OR, USA, 2010.

[36] L. Linguaglossa et al., "Survey of performance acceleration techniques for network function virtualization," *Proc. IEEE*, vol. 107, no. 4, pp. 746–764, Apr. 2019.

[37] P. Shantharama, A. S. Thyagaturu, and M. Reisslein, "Hardware-accelerated platforms and infrastructures for network functions: A survey of enabling technologies and research studies," *IEEE Access*, vol. 8, pp. 132021–132085, 2020.

[38] A. S. Thyagaturu, P. Shantharama, A. Nasrallah, and M. Reisslein, "Operating systems and hypervisors for network functions: A survey of enabling technologies and research studies," *IEEE Access*, vol. 10, pp. 79825–79873, 2022.

[39] M. Polese, L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "Understanding O-RAN: Architecture, interfaces, algorithms, security, and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 2, pp. 1376–1411, 2nd Quart., 2023.

[40] P. Mell and T. Grance, "The NIST definition of cloud computing," US Dept. Comm., Nat. Instit. Stand. Technol., Gaithersburg, MD, USA, Rep. SP 800-145, 2011.

[41] Y. Duan, G. Fu, N. Zhou, X. Sun, N. C. Narendra, and B. Hu, "Everything as a service (XaaS) on the cloud: Origins, current and future trends," in *Proc. IEEE 8th Int. Conf. Cloud Comput.*, 2015, pp. 621–628.

[42] A. Randal, "The ideal versus the real: Revisiting the history of virtual machines and containers," *ACM Comput. Survey*, vol. 53, no. 1, pp. 1–31, 2020.

[43] I. Nadareishvili, R. Mitra, M. McLarty, and M. Amundsen, *Microservice Architecture: Aligning Principles, Practices, and Culture*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2016.

[44] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 450–465, Feb. 2018.

[45] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan. 2017.

[46] M. Agiwal, A. Roy, and N. Saxena, "Next generation 5G wireless networks: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 3, pp. 1617–1655, 3rd Quart., 2016.

[47] B. Sayadi et al., "5G-PPP software network working group—Network applications: Opening up 5G and beyond networks," 5 GPPP Heidelberg, Germany, White Paper, Jul. 2023.

[48] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st Ed. MCC Workshop Mobile Cloud Comput.*, 2012, pp. 13–16. [Online]. Available: https://doi.org/10.1145/2342509.2342513

[49] P. Bellavista, J. Berrocal, A. Corradi, S. K. Das, L. Foschini, and A. Zanni, "A survey on fog computing for the Internet of Things," *Pervasive Mobile Comput.*, vol. 52, pp. 71–99, Jan. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574119218301111

[50] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84987842183&doi=10.1109

[51] L. Bittencourt et al., "The Internet of Things, fog and cloud continuum: Integration and challenges," *Internet Things*, vols. 3–4, pp. 134–155, Oct. 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2542660518300635

[52] P. Salva-Garcia, R. Ricart-Sanchez, E. Chirivella-Perez, Q. Wang, and J. M. Alcaraz-Calero, "XDP-based SmartNIC hardware performance acceleration for next-generation networks," *J. Netw. Syst. Manag.*, vol. 30, no. 4, p. 75, Oct. 2022. [Online]. Available: https://doi.org/10.1007/s10922-022-09687-z

[53] J. Pfefferle, P. Stuedi, A. Trivedi, B. Metzler, I. Koltsidas, and T. R. Gross, "A hybrid I/O virtualization framework for RDMA-capable network interfaces," *ACM SIGPLAN Not.*, vol. 50, no. 7, pp. 17–30, Mar. 2015.

[54] Z. He et al., "MasQ: RDMA for virtual private cloud," in *Proc. Annu. Conf. ACM Spec. Interest Group Data Commun. Appl., Technol., Archit., Protoc. Comput. Commun.*, 2020, pp. 1–14.

[55] R. Russell, "Virtio: Towards a de-facto standard for virtual I/O devices," *ACM SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 95–103, Jul. 2008.

[56] D. Kim et al., "FreeFlow: Software-based virtual RDMA networking for containerized clouds," in *Proc. 16th USENIX Symp. Netw. Syst. Design Implement. (NSDI 19)*, 2019, pp. 113–126.

[57] D. Wang, B. Fu, G. Lu, K. Tan, and B. Hua, "VSocket: Virtual socket interface for RDMA in public clouds," in *Proc. 15th ACM SIGPLAN/SIGOPS Int. Conf. Virtual Exec. Environ.*, 2019, pp. 179–192.

[58] B. Li, T. Cui, Z. Wang, W. Bai, and L. Zhang, "Socksdirect: Datacenter sockets can be fast and compatible," in *Proc. ACM Spec. Interest Group Data Commun.*, 2019, pp. 90–103.

[59] A. Garbugli, L. Rosa, A. Bujari, and L. Foschini, "KuberneTSN: A deterministic overlay network for time-sensitive Containerized environments," in *Proc. IEEE Int. Conf. Commun.*, 2023, pp. 1494–1499.

[60] "PCI-SIG single root I/O virtualization." Accessed: Apr. 4, 2024. [Online]. Available: https://pcisig.com/specifications/iov/single_root/

[61] M. Mahalingam et al., "Virtual extensible local area network (VXLAN): A framework for overlaying virtualized layer 2 networks over layer 3 networks," Internet Eng. Task Force, RFC 7348, 2014. [Online]. Available: https://www.rfc-editor.org/info/rfc7348

[62] D. Firestone, "VFP: A virtual switch platform for host SDN in the public cloud," in *Proc. 14th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2017, pp. 315–328.

[63] "Open vSwitch." 2018. [Online]. Available: http://openvswitch.org/

[64] R. Picoreti, A. Pereira do Carmo, F. Mendonça de Queiroz, A. Salles Garcia, R. Frizera Vassallo, and D. Simeonidou, "Multilevel observability in cloud orchestration," in *Proc. IEEE 16th Intl Conf Dependable, Auton. Secure Comput., 16th Intl Conf Pervasive Intell. Comput., 4th Intl Conf Big Data Intell. Comput. Cyber Sci. Technol. Congr.(DASC/PiCom/DataCom/CyberSciTech)*, 2018, pp. 776–784.

[65] G. Aceto, A. Botta, W. de Donato, and A. Pescapè, "Cloud monitoring: A survey," *Comput. Netw.*, vol. 57, no. 9, pp. 2093–2115, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128613001084

[66] "Regulation (EU) 2016/679 of the European parliament and of the council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/EC (general data protection regulation)," Eur. Union, Belgium, U.K., document 32016R0679, 2016. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2016/679/oj/eng

[67] *California Consumer Privacy Act*, California State Legis., Sacramento, CA, USA, 2018.

[68] "Amazon VPC quotas." Amazon. 2022. [Online]. Available: https://docs.aws.amazon.com/vpc/latest/userguide/amazon-vpc-limits.html

[69] "Google cloud network pricing." Google. 2022. [Online]. Available: https://cloud.google.com/vpc/network-pricing

[70] "Azure bandwidth pricing." Microsoft. 2022. [Online]. Available: https://azure.microsoft.com/en-us/pricing/details/bandwidth

[71] W. Huang, A. Ganjali, B. H. Kim, S. Oh, and D. Lie, "The state of public infrastructure-as-a-service cloud security," *ACM Comput. Survey*, vol. 47, no. 4, pp. 1–31, Jun. 2015. [Online]. Available: https://doi.org/10.1145/2767181

[72] F. Chen, D. Luo, T. Xiang, P. Chen, J. Fan, and H.-L. Truong, "IoT cloud security review: A case study approach using emerging consumer-oriented applications," *ACM Comput. Survey*, vol. 54, no. 4, pp. 1–36, May 2021. [Online]. Available: https://doi.org/10.1145/3447625

[73] C. Delimitrou and C. Kozyrakis, "Bolt: I know what you did last summer$\cdots$ in the cloud," in *Proc. 23rd Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2017, pp. 599–613. [Online]. Available: https://doi.org/10.1145/3037697.3037703

[74] J. Khalid et al., "Iron: Isolating network-based CPU in container environments," in *Proc. 15th USENIX Conf. Netw. Syst. Des. Implement.*, 2018, pp. 313–328.

[75] V. Del Piccolo, A. Amamou, K. Haddadou, and G. Pujolle, "A survey of network isolation solutions for multi-tenant data centers," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 4, pp. 2787–2821, 4th Quart., 2016.

[76] S. Peter et al., "Arrakis: The operating system is the control plane," *ACM Trans. Comput. Syst.*, vol. 33, no. 4, pp. 1–30, Nov. 2015. [Online]. Available: https://doi.org/10.1145/2812806

[77] A. Belay et al., "The IX operating system: Combining low latency, high throughput, and efficiency in a protected dataplane," *ACM Trans. Comput. Syst.*, vol. 34, no. 4, pp. 1–39, Dec. 2016. [Online]. Available: https://doi.org/10.1145/2997641

[78] (NVIDIA, Santa Clara, CA, USA). *ConnectX-6 dx*. (2019). [Online]. Available: https://www.nvidia.com/en-us/networking/ethernet/connectx-6-dx

[79] J. Liu, C. Maltzahn, C. D. Ulmer, and M. L. Curry, "Performance characteristics of the BlueField-2 SmartNIC," 2021, *arXiv:2105.06619*.

[80] A. M. Caulfield et al., "A cloud-scale acceleration architecture," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2016, pp. 1–13.

[81] D. Firestone et al., "Azure accelerated networking: SmartNICs in the public cloud," in *Proc. 15th USENIX Conf. Netw. Syst. Des. Implement.*, 2018, pp. 51–64.

[82] A. Trivedi, B. Metzler, and P. Stuedi, "A case for RDMA in clouds: Turning supercomputer networking into commodity," in *Proc. 2nd Asia-Pac. Workshop Syst.*, 2011, pp. 1–5. [Online]. Available: https://doi.org/10.1145/2103799.2103820

[83] M. Wadekar, "InfiniBand, iWARP, and RoCE," in *Handbook of Fiber Optic Data Communication*, 4th ed., C. DeCusatis, Ed. Oxford, U.K.: Academic, 2013, pp. 267–287. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780124016736000118

[84] C. Guo et al., "RDMA over commodity Ethernet at scale," in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 202–215. [Online]. Available: https://doi.org/10.1145/2934872.2934908

[85] R. Mittal et al., "Revisiting network support for RDMA," in *Proc. Conf. ACM Spec. Interest Group Data Commun.*, 2018, pp. 313–326. [Online]. Available: https://doi.org/10.1145/3230543.3230557

[86] R. J. Recio, P. R. Culley, D. Garcia, B. Metzler, and J. Hilland, "A remote direct memory access protocol specification," Internet Eng. Task Force, RFC 5040, Oct. 2007. [Online]. Available: https://www.rfc-editor.org/info/rfc5040

[87] M. J. Rashti, R. E. Grant, A. Afsahi, and P. Balaji, "iWARP redefined: Scalable connectionless communication over high-speed Ethernet," in *Proc. Int. Conf. High Perform. Comput.*, 2010, pp. 1–10.

[88] Open Fabric Alliance, Palo Alto, CA, USA. *Verbs API*. Accessed: Apr. 4, 2024. [Online]. Available: https://www.openfabrics.org/ofa-overview

[89] Open Fabric Alliance, Palo Alto, CA, USA. *RDMA Connection Manager*. Accessed: Apr. 4, 2024. [Online]. Available: https://github.com/ofiwg/librdmacm

[90] S.-Y. Tsai and Y. Zhang, "A double-edged sword: Security threats and opportunities in one-sided network communication," in *Proc. 11th USENIX Workshop Hot Topics Cloud Comput. (HotCloud)*, 2019, p. 8. [Online]. Available: https://www.usenix.org/conference/hotcloud19/presentation/tsai

[91] (Amazon Inc., Bellevue, WA, USA). *Amazon EC2 P4 Instances*. Accessed: Apr. 4, 2024. [Online]. Available: https://aws.amazon.com/ec2/instance-types/p4

[92] (Google, Mountain View, CA, USA). *Google Cloud GPU*. Accessed: Apr. 4, 2024. [Online]. Available: https://cloud.google.com/gpu

[93] (Microsoft, Redmond, WA, USA). *Microsoft Azure N-Series Virtual Machines*. Accessed: Apr. 4, 2024. [Online]. Available: https://azure.microsoft.com/pricing/details/virtual-machines/series

[94] (Microsoft, Redmond, WA, USA). *DPDK in an Azure linux VM*. (2023). [Online]. Available: https://learn.microsoft.com/en-us/azure/virtual-network/setup-dpdk

[95] (Google, Mountain View, CA, USA). *DPDK in an Google VM*. (2023). [Online]. Available: https://cloud.google.com/compute/docs/networking/use-dpdk

[96] (Amazon Inc., Bellevue, WA, USA). *DPDK in an AWS VM*. (2023). [Online]. Available: https://aws.amazon.com/blogs/industries/automate-packet-acceleration-configuration-using-dpdk-on-amazon-eks/

[97] S. Ma, T. Ma, K. Chen, and Y. Wu, "A survey of storage systems in the RDMA era," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 4395–4409, Dec. 2022.

[98] A. Kalia, M. Kaminsky, and D. G. Andersen, "FaSST: Fast, scalable and simple distributed transactions with two-sided (RDMA) datagram RPCs," in *Proc. 12th USENIX Symp. Oper. Syst. Des. Implement. (OSDI)*, 2016, pp. 185–201. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/kalia

[99] W. Song et al., "Cascade: An edge computing platform for real-time machine intelligence," in *Proc. Workshop Adv. Tools, Program. Lang., PLatforms Implement. Eval. Algorithm. Distrib. Syst.*, 2022, pp. 2–6. [Online]. Available: https://doi.org/10.1145/3524053.3542741

[100] Y. Gao et al., "When cloud storage meets RDMA," in *Proc. 18th USENIX Symp. Netw. Syst. Des. Implement. (NSDI)*, 2021, pp. 519–533. [Online]. Available: https://www.usenix.org/conference/nsdi21/presentation/gao

[101] W. Bai et al., "Empowering azure storage with RDMA," in *Proc. 20th USENIX Symp. Netw. Syst. Des. Implement. (NSDI)*, 2023, pp. 49–67. [Online]. Available: https://www.usenix.org/conference/nsdi23/presentation/bai

[102] Z. Li, N. Liu, and J. Wu, "Toward a production-ready general-purpose RDMA-enabled RPC," in *Proc. ACM SIGCOMM Conf. Posters Demos*, 2019, pp. 27–29. [Online]. Available: https://doi.org/10.1145/3342280.3342296

[103] A. Kalia, M. Kaminsky, and D. Andersen, "Datacenter RPCs can be general and fast," in *Proc. 16th USENIX Symp. Netw. Syst. Des. Implement. (NSDI)*, 2019, pp. 1–16. [Online]. Available: https://www.usenix.org/conference/nsdi19/presentation/kalia

[104] N. Lazarev, N. Adit, S. Xiang, Z. Zhang, and C. Delimitrou, "Dagger: Towards efficient RPCs in cloud Microservices with near-memory reconfigurable NICs," *IEEE Comput. Archit. Lett.*, vol. 19, no. 2, pp. 134–138, Jul.–Dec. 2020.

[105] L. Rosa, W. Song, L. Foschini, A. Corradi, and K. Birman, "DerechoDDS: Strongly consistent data distribution for mission-critical applications," in *Proc. IEEE Mil. Commun. Conf. (MILCOM)*, 2021, pp. 684–689.

[106] A. Sabbioni, L. Rosa, A. Bujari, L. Foschini, and A. Corradi, "A shared memory approach for function chaining in Serverless platforms," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, 2021, pp. 1–6.

[107] H. Hoang, B. Cassell, T. Brecht, and S. Al-Kiswany, "RocketBufs: A framework for building efficient, in-memory, message-oriented Middleware," in *Proc. 14th ACM Int. Conf. Distrib. Event Based Syst.*, 2020, pp. 121–132. [Online]. Available: https://doi.org/10.1145/3401025.3401744

[108] M. Copik, K. Taranov, A. Calotoiu, and T. Hoefler, "rFaaS: Enabling high performance serverless with RDMA and decentralization," 2021, *arXiv:2106.13859*.

[109] A. Sabbioni, L. Rosa, A. Bujari, L. Foschini, and A. Corradi, "DIFFUSE: A distributed and decentralized platform enabling function composition in serverless environments," *Comput. Netw.*, vol. 210, Jun. 2022, Art. no. 108993. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S138912862200161X

[110] T. Kuchler, M. Giardino, T. Roscoe, and A. Klimovic, "Function as a function," in *Proc. ACM Symp. Cloud Comput.*, 2023, pp. 81–92. [Online]. Available: https://doi.org/10.1145/3620678.3624648

[111] I. Zhang et al., "The demikernel datapath OS architecture for microsecond-scale datacenter systems," in *Proc. ACM SIGOPS 28th Symp. Oper. Syst. Princ.*, 2021, pp. 195–211. [Online]. Available: https://doi.org/10.1145/3477132.3483569

[112] N. Bonelli, F. D. Vigna, A. Fais, G. Lettieri, and G. Procissi, "Programming socket-independent network functions with Nethuns," *SIGCOMM Comput. Commun. Rev.*, vol. 52, no. 2, pp. 35–48, Jun. 2022. [Online]. Available: https://doi.org/10.1145/3544912.3544917

[113] L. Rosa, A. Garbugli, A. Corradi, and P. Bellavista, "INSANE: A unified middleware for QoS-aware network acceleration in edge cloud computing," in *Proc. 24th Int. Middlew. Conf.*, 2023, pp. 57–70. [Online]. Available: https://doi.org/10.1145/3590140.3629105

[114] "rsocket(7)." 2019. [Online]. Available: https://linux.die.net/man/7/rsocket

[115] A. Cohen, "A performance analysis of the sockets direct protocol (SDP) with asynchronous I/O over 4X InfiniBand," in *Proc. IEEE Int. Conf. Perform., Comput., Commun.*, 2004, pp. 241–246.

[116] R. Russell, "The extended sockets interface for accessing RDMA hardware," in *Proc. 20th IASTED Int. Conf. Parallel Distrib. Comput. Syst. (PDCS)*, 2008, pp. 279–284.

[117] (Mellanox Technol. Ltd., Sunnyvale, CA, USA). *Mellanox Messaging Accelerator*. [Online]. Available: http://www.mellanox.com/page/software_vma

[118] M. K. Aguilera et al., "Remote regions: A simple abstraction for remote memory," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2018, pp. 775–787. [Online]. Available: https://www.usenix.org/conference/atc18/presentation/aguilera

[119] "lwIP—A lightweight TCP/IP stack." Accessed: Apr. 4, 2024. [Online]. Available: https://savannah.nongnu.org/projects/lwip

[120] S.-Y. Tsai and Y. Zhang, "LITE kernel RDMA support for datacenter applications," in *Proc. 26th Symp. Oper. Syst. Princ.*, 2017, pp. 306–324.

[121] T. Ma et al., "X-RDMA: Effective RDMA Middleware in large-scale production environments," in *Proc. IEEE Int. Conf. Clust. Comput. (CLUSTER)*, 2019, pp. 1–12.

[122] (Linux Found., San Francisco, CA, USA). *Poll Mode Driver for Emulated Virtio NIC*. (2023). [Online]. Available: https://doc.dpdk.org/guides/nics/virtio.html

[123] S. Fan, F. Chen, H. Rauchfuss, N. Har'El, U. Schilling, and N. Struckmann, "Towards a lightweight RDMA para-virtualization for HPC," in *Proc. COSH/VisorHPC@HiPEAC*, 2017, pp. 1–6.

[124] A. Mouzakitis et al., "Lightweight and generic RDMA engine para-virtualization for the KVM hypervisor," in *Proc. Int. Conf. High Perform. Comput. Simul. (HPCS)*, 2017, pp. 737–744.

[125] (Google, Mountain View, CA, USA). *Google Virtual NIC*. (2023). [Online]. Available: https://cloud.google.com/compute/docs/networking/using-gvnic

[126] (Amazon Inc., Seattle, WA, USA). *Amazon Nitro*. (2017). [Online]. Available: https://aws.amazon.com/ec2/nitro

[127] M. Arumugam et al., "Bluebird: High-performance SDN for bare-metal cloud services," in *Proc. 19th USENIX Symp. Netw. Syst. Des. Implement. (NSDI)*, 2022, pp. 355–370. [Online]. Available: https://www.usenix.org/conference/nsdi22/presentation/arumugam

[128] (Intel Semicond. Corp., Santa Clara, CA, USA). *Intel Tofino 2*. (2021). [Online]. Available: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-2-series.html

[129] (Broadcom Inc., San Jose, CA, USA). *Trident SmartToR*. (2021). [Online]. Available: https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/smarttor

[130] P. Bosshart et al., "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014. [Online]. Available: https://doi.org/10.1145/2656877.2656890

[131] P. Bosshart et al., "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in *Proc. ACM SIGCOMM Conf. SIGCOMM*, 2013, pp. 99–110. [Online]. Available: https://doi.org/10.1145/2486001.2486011

[132] Y. Chen, Y. Lu, and J. Shu, "Scalable RDMA RPC on reliable connection with efficient resource sharing," in *Proc. 14th EuroSys Conf.*, 2019, pp. 1–14. [Online]. Available: https://doi.org/10.1145/3302424.3303968

[133] P. M. Phothilimthana, M. Liu, A. Kaufmann, S. Peter, R. Bodik, and T. Anderson, "Floem: A programming system for NIC-accelerated network applications," in *Proc. 13th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2018, pp. 663–679. [Online]. Available: https://www.usenix.org/conference/osdi18/presentation/phothilimthana

[134] S. Grant, A. Yelam, M. Bland, and A. C. Snoeren, "SmartNIC performance isolation with FairNIC: Programmable networking for the cloud," in *Proc. Annu. Conf. ACM Spec. Interest Group Data Commun. Appl., Technol., Archit., Protocols Comput. Commun.*, 2020, pp. 681–693. [Online]. Available: https://doi.org/10.1145/3387514.3405895

[135] J. Xing, K.-F. Hsu, Y. Qiu, Z. Yang, H. Liu, and A. Chen, "Bedrock: Programmable network support for secure RDMA systems," in *Proc. 31st USENIX Security Symp. (USENIX Security)*, 2022, pp. 2585–2600. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/xing

[136] W. Huang, J. Liu, M. Koop, B. Abali, and D. Panda, "Nomad: Migrating OS-bypass networks in virtual machines," in *Proc. 3rd Int. Conf. Virtual Execut. Environ.*, 2007, pp. 158–168. [Online]. Available: https://doi.org/10.1145/1254810.1254833

[137] M. Planeta, J. Bierbaum, L. S. D. Antony, T. Hoefler, and H. Härtig, "MigrOS: Transparent live-migration support for Containerised RDMA applications," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2021, pp. 47–63. [Online]. Available: https://www.usenix.org/conference/atc21/presentation/planeta

[138] B. Rothenberger, K. Taranov, A. Perrig, and T. Hoefler, "ReDMArk: Bypassing RDMA security mechanisms," in *Proc. 30th USENIX Security Symp. (USENIX Security)*, 2021, pp. 4277–4292.

[139] "CRIU. Checkpoint/restore in userspace." 2011. [Online]. Available: https://criu.org
[140] M. Lee, E. J. Kim, and M. Yousif, "Security enhancement in InfiniBand architecture," in *Proc. 19th IEEE Int. Parallel Distrib. Process. Symp.*, 2005, p. 10.
[141] M. Lee and E. J. Kim, "A comprehensive framework for enhancing security in InfiniBand architecture," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 10, pp. 1393–1406, Oct. 2007.
[142] A. K. Simpson, A. Szekeres, J. Nelson, and I. Zhang, "Securing RDMA for high-performance datacenter storage systems," in *Proc. 12th USENIX Workshop Hot Topics Cloud Comput. (HotCloud)*, 2020, p. 11.
[143] K. Taranov, B. Rothenberger, A. Perrig, and T. Hoefler, "sRDMA—Efficient NIC-based authentication and encryption for remote direct memory access," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2020, pp. 691–704.
[144] S.-Y. Tsai, M. Payer, and Y. Zhang, "Pythia: Remote oracles for the masses," in *Proc. 28th USENIX Security Symp. (USENIX Security)*, 2019, pp. 693–710.
[145] M. Kurth, B. Gras, D. Andriesse, C. Giuffrida, H. Bos, and K. Razavi, "NetCAT: Practical cache attacks from the network," in *Proc. IEEE Symp. Security Privacy (SP)*, 2020, pp. 20–38.
[146] J. Pinkerton and E. Deleganes, "Direct data placement protocol (DDP) / remote direct memory access protocol (RDMAP) security," Internet Eng. Task Force, RFC 5042, Oct. 2007. [Online]. Available: https://www.rfc-editor.org/info/rfc5042
[147] H. Naghibijouybari, E. M. Koruyeh, and N. Abu-Ghazaleh, "Microarchitectural attacks in heterogeneous systems: A survey," *ACM Comput. Survey*, vol. 55, no. 7, pp. 1–40, Dec. 2022. [Online]. Available: https://doi.org/10.1145/3544102
[148] (VMware Cloud Comput. Co., Palo Alto, CA, USA). *ESXi VM and Hypervisor Escape Advisory*. (2018). [Online]. Available: https://www.vmware.com/security/advisories/VMSA-2018-0027.html
[149] N. Doraswamy and D. Harkins, *IPSec: The New Security Standard for the Internet, Intranets, and Virtual Private Networks*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2003.
[150] E. Rescorla and N. Modadugu, "Datagram transport layer security version 1.2," Internet Eng. Task Force, RFC 6347, Jan. 2012. [Online]. Available: https://www.rfc-editor.org/info/rfc6347
[151] K. Taranov, B. Rothenberger, D. De Sensi, A. Perrig, and T. Hoefler, "NeVerMore: Exploiting RDMA mistakes in NVMe-oF storage applications," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2022, pp. 2765–2778. [Online]. Available: https://doi.org/10.1145/3548606.3560568
[152] D. Gruss, R. Spreitzer, and S. Mangard, "Cache template attacks: Automating attacks on inclusive last-level caches," in *Proc. 24th USENIX Security Symp. (USENIX Security)*, 2015, pp. 897–912. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/gruss
[153] P. Stuedi et al., "Crail: A high-performance I/O architecture for distributed data processing," *IEEE Data Eng. Bull.*, vol. 40, no. 1, pp. 38–49, Mar. 2017. [Online]. Available: http://sites.computer.org/debull/A17mar/p38.pdf
[154] E. Zamanian, C. Binnig, T. Harris, and T. Kraska, "The end of a myth: Distributed transactions can scale," *Proc. VLDB Endow.*, vol. 10, no. 6, pp. 685–696, Feb. 2017. [Online]. Available: https://doi.org/10.14778/3055330.3055335
[155] A. Dragojevic, D. Narayanan, and M. Castro, "RDMA reads: To use or not to use?" *IEEE Data Eng. Bull.*, vol. 40, pp. 3–14, Mar. 2017.
[156] J. Wen, Z. Chen, X. Jin, and X. Liu, "Rise of the planet of serverless computing: A systematic review," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 5, pp. 1–61, Jul. 2023. [Online]. Available: https://doi.org/10.1145/3579643
[157] H. Shafiei, A. Khonsari, and P. Mousavi, "Serverless computing: A survey of opportunities, challenges, and applications," *ACM Comput. Survey*, vol. 54, no. 11s, pp. 1–32, Nov. 2022. [Online]. Available: https://doi.org/10.1145/3510611
[158] E. Baccarelli, P. G. V. Naranjo, M. Scarpiniti, M. Shojafar, and J. H. Abawajy, "Fog of everything: Energy-efficient networked computing architectures, research challenges, and a case study," *IEEE Access*, vol. 5, pp. 9882–9910, 2017.
[159] Z. Xiang, M. Höweler, D. You, M. Reisslein, and F. H. P. Fitzek, "X-MAN: A non-intrusive power manager for energy-adaptive cloud-native network functions," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 2, pp. 1017–1035, Jun. 2022.
[160] M. Ewais and P. Chow, "Disaggregated memory in the datacenter: A survey," *IEEE Access*, vol. 11, pp. 20688–20712, 2023.
[161] S. Legtchenko et al., "Understanding rack-scale disaggregated storage," in *Proc. 9th USENIX Workshop Hot Topics Storage File Syst. (HotStorage)*, 2017, pp. 1–6. [Online]. Available: https://www.usenix.org/conference/hotstorage17/program/presentation/legtchenko
[162] M. Shahrad et al., "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2020, pp. 205–218. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/shahrad
[163] A. Mahgoub, E. B. Yi, K. Shankar, S. Elnikety, S. Chaterji, and S. Bagchi, "ORION and the three rights: Sizing, bundling, and prewarming for Serverless DAGs," in *Proc. 16th USENIX Symp. Oper. Syst. Des. Implement. (OSDI)*, 2022, pp. 303–320. [Online]. Available: https://www.usenix.org/conference/osdi22/presentation/mahgoub
[164] Y. Deng, A. Montemayor, A. Levy, and K. Winstein, "Computation-centric networking," in *Proc. 21st ACM Workshop Hot Topics Netw.*, 2022, pp. 167–173. [Online]. Available: https://doi.org/10.1145/3563766.3564106
[165] Q. Zhang, H. Sun, X. Wu, and H. Zhong, "Edge video analytics for public safety: A review," *Proc. IEEE*, vol. 107, no. 8, pp. 1675–1696, Aug. 2019.
[166] A. Li et al., "Evaluating modern GPU interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 1, pp. 94–110, Jan. 2020.
[167] I. Burstein, "Nvidia data Center processing unit (DPU) architecture," in *Proc. IEEE Hot Chips 33 Symp. (HCS)*, 2021, pp. 1–20.
[168] B. Burres et al., "Intel's hyperscale-ready infrastructure processing unit (IPU)," in *Proc. IEEE Hot Chips 33 Symp. (HCS)*, 2021, pp. 1–16.

Lorenzo Rosa (Graduate Student Member, IEEE) is currently pursuing the Ph.D. degree in computer science and engineering with the University of Bologna, Italy. His research is in computer systems. He focuses on the integration of network acceleration protocols (e.g., DPDK and RDMA) and devices (e.g., SmartNIC and programmable hardware) in general-purpose cloud and edge cloud platforms. He is also interested in serverless computing.

Luca Foschini (Senior Member, IEEE) received the Ph.D. degree in computer science engineering from the University of Bologna, Italy, in 2007 where is an Associate Professor. His interests span from integrated management of distributed systems and services to wireless pervasive computing, and scalable context data distribution infrastructures and context-aware services.

Antonio Corradi (Life Senior Member, IEEE) graduated from the University of Bologna, Italy, and received the M.S. degree in electrical engineering from Cornell University, USA. He is currently a Full Professor of Computer Engineering with the University of Bologna. His research interests include all aspects of cloud and middleware, interoperability and innovation, and novel aspects of cloud continuum interactions.

Open Access funding provided by 'Alma Mater Studiorum - Università di Bologna' within the CRUI CARE Agreement