CONTRIBUTED
P A P E R

Software-Defined Networking: A Comprehensive Survey

This paper offers a comprehensive survey of software-defined networking covering its context, rationale, main concepts, distinctive features, and future challenges.

By Diego Kreutz, *Member IEEE*, Fernando M. V. Ramos, *Member IEEE*, Paulo Esteves Veríssimo, *Fellow IEEE*, Christian Esteve Rothenberg, *Member IEEE*, Siamak Azodolmolky, *Senior Member IEEE*, and Steve Uhlig, *Member IEEE*

ABSTRACT | The Internet has led to the creation of a digital society, where (almost) everything is connected and is accessible from anywhere. However, despite their widespread adoption, traditional IP networks are complex and very hard to manage. It is both difficult to configure the network according to predefined policies, and to reconfigure it to respond to faults, load, and changes. To make matters even more difficult, current networks are also vertically integrated: the control and data planes are bundled together. Software-defined networking (SDN) is an emerging paradigm that promises to change this state of affairs, by breaking vertical integration, separating the network's control logic from the underlying routers and switches, promoting (logical) centralization of network control, and introducing the ability to program the network. The separation of concerns, introduced between the definition of network policies, their implementation in switching hardware, and the forwarding of traffic, is key to the desired flexibility: by breaking the network control problem into tractable pieces, SDN makes it easier to create and introduce new abstractions in networking, simplifying network management and facilitating network evolution. In this paper, we present a comprehensive survey on SDN. We start by introducing the motivation for SDN, explain its main concepts and how it differs from traditional networking, its roots, and the standardization activities regarding this novel paradigm. Next, we present the key building blocks of an SDN infrastructure using a bottom-up, layered approach. We provide an in-depth analysis of the hardware infrastructure, southbound and northbound application programming interfaces (APIs), network virtualization layers, network operating systems (SDN controllers), network programming languages, and network applications. We also look at cross-layer problems such as debugging and troubleshooting. In an effort to anticipate the future evolution of this new paradigm, we discuss the main ongoing research efforts and challenges of SDN. In particular, we address the design of switches and control platforms—with a focus on aspects such as resiliency, scalability, performance, security, and dependability—as well as new opportunities for carrier transport networks and cloud providers. Last but not least, we analyze the position of SDN as a key enabler of a software-defined environment.

KEYWORDS | Carrier-grade networks; dependability; flow-based networking; network hypervisor; network operating systems (NOSs); network virtualization; OpenFlow; programmable networks; programming languages; scalability; software-defined environments; software-defined networking (SDN)

Manuscript received June 15, 2014; revised October 6, 2014; accepted November 10, 2014. Date of current version December 18, 2014.

D. Kreutz and **F. M. V. Ramos** are with the Department of Informatics of Faculty of Sciences, University of Lisbon, 1749-016 Lisbon, Portugal (e-mail: kreutz@ieee.org; fvramos@fc.ul.pt).
P. E. Veríssimo is with the Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg, L-2721 Walferdange, Luxembourg (e-mail: paulo.verissimo@uni.lu).
C. E. Rothenberg is with the School of Electrical and Computer Engineering (FEEC), University of Campinas, Campinas 13083-970, Brazil (e-mail: chesteve@dca.fee.unicamp.br).
S. Azodolmolky is with the Gesellschaft für Wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG), 37077 Göttigen, Germany (e-mail: siamak.azodolmolky@gwdg.de).
S. Uhlig is with Queen Mary University of London, London E1 4NS, U.K. (e-mail: steve@eecs.qmul.ac.uk).

Digital Object Identifier: 10.1109/JPROC.2014.2371999

I. INTRODUCTION

The distributed control and transport network protocols running inside the routers and switches are the key technologies that allow information, in the form of digital packets, to travel around the world. Despite their widespread adoption, traditional IP networks are complex and hard to manage [1]. To express the desired high-level network policies, network operators need to configure each individual network device separately using low-level and often vendor-specific commands. In addition to the configuration complexity, network environments have to endure the dynamics of faults and adapt to load changes. Automatic reconfiguration and response mechanisms are virtually nonexistent in current IP networks. Enforcing the required policies in such a dynamic environment is therefore highly challenging.

To make it even more complicated, current networks are also vertically integrated. The control plane (that decides how to handle network traffic) and the data plane (that forwards traffic according to the decisions made by the control plane) are bundled inside the networking devices, reducing flexibility and hindering innovation and evolution of the networking infrastructure. The transition from IPv4 to IPv6, started more than a decade ago and still largely incomplete, bears witness to this challenge, while in fact IPv6 represented merely a protocol update. Due to the inertia of current IP networks, a new routing protocol can take five to ten years to be fully designed, evaluated, and deployed. Likewise, a clean-slate approach to change the Internet architecture (e.g., replacing IP) is regarded as a daunting task—simply not feasible in practice [2], [3]. Ultimately, this situation has inflated the capital and operational expenses of running an IP network.

Software-defined networking (SDN) [4], [5] is an emerging networking paradigm that gives hope to change the limitations of current network infrastructures. First, it breaks the vertical integration by separating the network's control logic (the control plane) from the underlying routers and switches that forward the traffic (the data plane). Second, with the separation of the control and data planes, network switches become simple forwarding devices and the control logic is implemented in a logically centralized controller (or network operating system[1]), simplifying policy enforcement and network (re)configuration and evolution [6]. A simplified view of this architecture is shown in Fig. 1. It is important to emphasize that a logically centralized programmatic model does not postulate a physically centralized system [7]. In fact, the need to guarantee adequate levels of performance, scalability, and reliability would preclude such a solution. Instead, production-level SDN network designs resort to physically distributed control planes [7], [8].

The separation of the control plane and the data plane can be realized by means of a well-defined programming interface between the switches and the SDN controller. The controller exercises direct control over the state in the data plane elements via this well-defined application programming interface (API), as depicted in Fig. 1. The most notable example of such an API is OpenFlow [9], [10]. An OpenFlow switch has one or more tables of packet-

[1]We will use these two terms interchangeably.

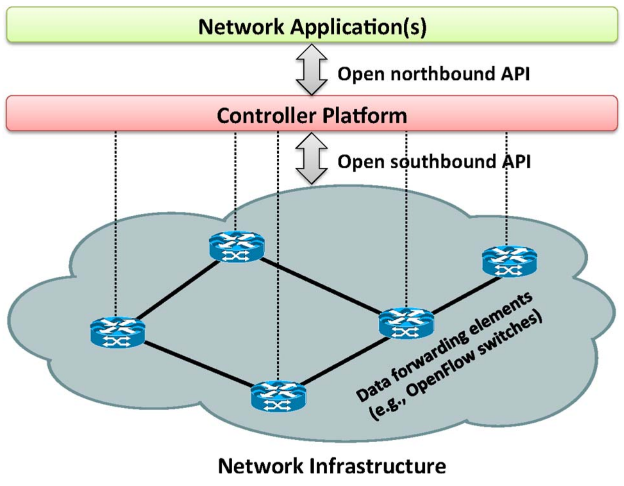

Fig. 1. ***Simplified view of an SDN architecture.***

handling rules (flow table). Each rule matches a subset of the traffic and performs certain actions (dropping, forwarding, modifying, etc.) on the traffic. Depending on the rules installed by a controller application, an OpenFlow switch can—instructed by the controller—behave like a router, switch, firewall, or perform other roles (e.g., load balancer, traffic shaper, and in general those of a middlebox).

An important consequence of the SDN principles is the separation of concerns introduced between the definition of network policies, their implementation in switching hardware, and the forwarding of traffic. This separation is key to the desired flexibility, breaking the network control problem into tractable pieces, and making it easier to create and introduce new abstractions in networking, simplifying network management and facilitating network evolution and innovation.

Although SDN and OpenFlow started as academic experiments [9], they gained significant traction in the industry over the past few years. Most vendors of commercial switches now include support of the OpenFlow API in their equipment. The SDN momentum was strong enough to make Google, Facebook, Yahoo, Microsoft, Verizon, and Deutsche Telekom fund Open Networking Foundation (ONF) [10] with the main goal of promotion and adoption of SDN through open standards development. As the initial concerns with SDN scalability were addressed [11]—in particular the myth that logical centralization implied a physically centralized controller, an issue we will return to later on—SDN ideas have matured and evolved from an academic exercise to a commercial success. Google, for example, has deployed an SDN to interconnect its data centers across the globe. This production network has been in deployment for three years, helping the company to improve operational efficiency and significantly reduce costs [8]. VMware's network virtualization platform, NSX [12], is another example. NSX is a commercial solution that delivers a fully functional network in software, provisioned independent of the underlying networking devices, entirely based around SDN principles. As a final example, the world's largest IT companies (from carriers and equipment manufacturers to cloud providers and financial services companies) have recently joined SDN consortia such as the ONF and the OpenDaylight initiative [13], another indication of the importance of SDN from an industrial perspective.

A few recent papers have surveyed specific architectural aspects of SDN [14]–[16]. An overview of OpenFlow and a short literature review can be found in [14] and [15]. These OpenFlow-oriented surveys present a relatively simplified three-layer stack composed of high-level network services, controllers, and the controller/switch interface. In [16], Jarraya *et al.* go a step further by proposing a taxonomy for SDN. However, similarly to the previous works, the survey is limited in terms of scope, and it does not provide an in-depth treatment of fundamental aspects of SDN. In essence, existing surveys lack a thorough discussion of the essential building blocks of an SDN such as the network operating systems (NOSs), programming languages, and interfaces. They also fall short on the analysis of cross-layer issues such as scalability, security, and dependability. A more complete overview of ongoing research efforts, challenges, and related standardization activities is also missing.

In this paper, we present, to the best of our knowledge, the most comprehensive literature survey on SDN to date. We organize this survey as depicted in Fig. 2. We start, in the next two sections, by explaining the context, introducing the motivation for SDN and explaining the main concepts of this new paradigm and how it differs from traditional networking. Our aim in the early part of the survey is also to explain that SDN is not as novel as a technological advance. Indeed, its existence is rooted at the intersection of a series of "old" ideas, technology drivers, and current and future needs. The concepts underlying SDN—the separation of the control and data planes, the flow abstraction upon which forwarding decisions are made, the (logical) centralization of network control, and the ability to program the network—are not novel by themselves [17]. However, the integration of already tested concepts with recent trends in networking—namely the availability of merchant switch silicon and the huge

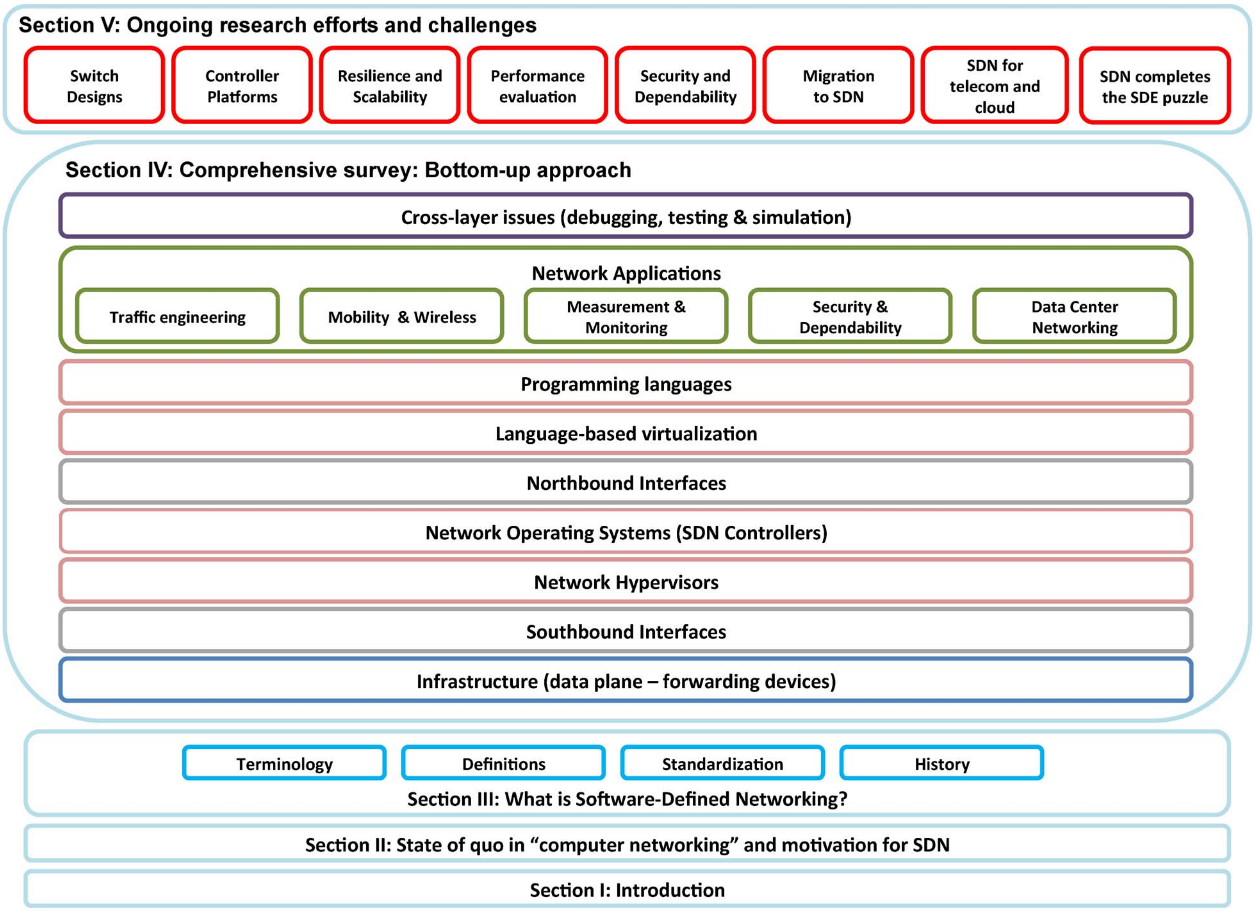
Section V: Ongoing research efforts and challenges
Switch Designs
Controller Platforms
Resilience and Scalability
Performance evaluation
Security and Dependability
Migration to SDN
SDN for telecom and cloud
SDN completes the SDE puzzle
Section IV: Comprehensive survey: Bottom-up approach
Cross-layer issues (debugging, testing & simulation)
Network Applications
Traffic engineering
Mobility & Wireless
Measurement & Monitoring
Security & Dependability
Data Center Networking
Programming languages
Language-based virtualization
Northbound Interfaces
Network Operating Systems (SDN Controllers)
Network Hypervisors
Southbound Interfaces
Infrastructure (data plane – forwarding devices)
Terminology
Definitions
Standardization
History
Section III: What is Software-Defined Networking?
Section II: State of quo in "computer networking" and motivation for SDN
Section I: Introduction

Fig. 2. ***Condensed overview of this survey on SDN.***

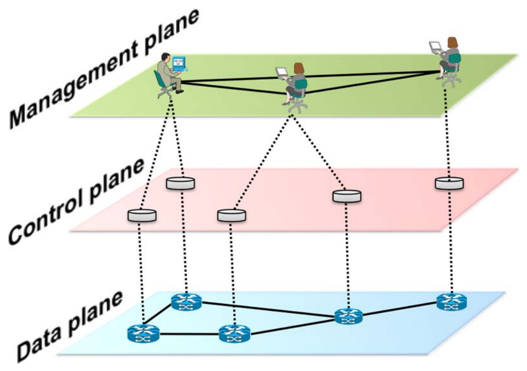

Fig. 3. *Layered view of networking functionality.*

interest in feasible forms of network virtualization—are leading to this paradigm shift in networking. As a result of the high industry interest and the potential to change the status quo of networking from multiple perspectives, a number of standardization efforts around SDN are ongoing, as we also discuss in Section III.

Section IV is the core of this survey, presenting an extensive and comprehensive analysis of the building blocks of an SDN infrastructure using a bottom-up, layered approach. The option for a layered approach is grounded on the fact that SDN allows thinking of networking along two fundamental concepts, which are common in other disciplines of computer science: separation of concerns (leveraging the concept of abstraction) and recursion. Our layered, bottom-up approach divides the networking problem into eight parts: 1) hardware infrastructure; 2) southbound interfaces; 3) network virtualization (hypervisor layer between the forwarding devices and the NOSs); 4) NOSs (SDN controllers and control platforms); 5) northbound interfaces (to offer a common programming abstraction to the upper layers, mainly the network applications); 6) virtualization using slicing techniques provided by special purpose libraries or programming languages and compilers; 7) network programming languages; and finally 8) network applications. In addition, we also look at cross-layer problems such as debugging and troubleshooting mechanisms. The discussion in Section V on ongoing research efforts, challenges, future work, and opportunities concludes this paper.

II. STATUS QUO IN NETWORKING

Computer networks can be divided in three planes of functionality: the data, control, and management planes (see Fig. 3). The data plane corresponds to the networking devices, which are responsible for (efficiently) forwarding data. The control plane represents the protocols used to populate the forwarding tables of the data plane elements. The management plane includes the software services, such as simple network management protocol (SNMP)-based tools [18], used to remotely monitor and configure the control functionality. Network policy is defined in the management plane, the control plane enforces the policy, and the data plane executes it by forwarding data accordingly.

In traditional IP networks, the control and data planes are tightly coupled, embedded in the same networking devices, and the whole structure is highly decentralized. This was considered important for the design of the Internet in the early days: it seemed the best way to guarantee network resilience, which was a crucial design goal. In fact, this approach has been quite effective in terms of network performance, with a rapid increase of line rate and port densities.

However, the outcome is a very complex and relatively static architecture, as has been often reported in the networking literature (e.g., [1]–[3], [6], and [19]). It is also the fundamental reason why traditional networks are rigid, and complex to manage and control. These two characteristics are largely responsible for a vertically integrated industry where innovation is difficult.

Network misconfigurations and related errors are extremely common in today's networks. For instance, more than 1000 configuration errors have been observed in border gateway protocol (BGP) routers [20]. From a single misconfigured device, very undesired network behavior may result (including, among others, packet losses, forwarding loops, setting up of unintended paths, or service contract violations). Indeed, while rare, a single misconfigured router is able to compromise the correct operation of the whole Internet for hours [21], [22].

To support network management, a small number of vendors offer proprietary solutions of specialized hardware, operating systems, and control programs (network applications). Network operators have to acquire and maintain different management solutions and the corresponding specialized teams. The capital and operational cost of building and maintaining a networking infrastructure is significant, with long return on investment cycles, which hamper innovation and addition of new features and services (for instance, access control, load balancing, energy efficiency, traffic engineering). To alleviate the lack of in-path functionalities within the network, a myriad of specialized components and middleboxes, such as firewalls, intrusion detection systems, and deep packet inspection engines, proliferate in current networks. A recent survey of 57 enterprise networks shows that the number of middleboxes is already on par with the number of routers in current networks [23]. Despite helping in-path functionalities, the net effect of middleboxes has increased complexity of network design and its operation.

III. WHAT IS SOFTWARE-DEFINED NETWORKING?

The term SDN was originally coined to represent the ideas and work around OpenFlow at Stanford University, Stanford, CA, USA [24]. As originally defined, SDN refers to a network architecture where the forwarding state in the data plane is managed by a remotely controlled plane decoupled from the former. The networking industry has on many occasions shifted from this original view of SDN by referring to anything that involves software as being SDN. We therefore attempt, in this section, to provide a much less ambiguous definition of SDN.

We define an SDN as a network architecture with four pillars.

1) The control and data planes are decoupled. Control functionality is removed from network devices that will become simple (packet) forwarding elements.
2) Forwarding decisions are flow based, instead of destination based. A flow is broadly defined by a set of packet field values acting as a match (filter) criterion and a set of actions (instructions). In the SDN/OpenFlow context, a flow is a sequence of packets between a source and a destination. All packets of a flow receive identical service policies at the forwarding devices [25], [26]. The flow abstraction allows unifying the behavior of different types of network devices, including routers, switches, firewalls, and middleboxes [27]. Flow programming enables unprecedented flexibility, limited only to the capabilities of the implemented flow tables [9].
3) Control logic is moved to an external entity, the so-called SDN controller or NOS. The NOS is a software platform that runs on commodity server technology and provides the essential resources and abstractions to facilitate the programming of forwarding devices based on a logically centralized, abstract network view. Its purpose is therefore similar to that of a traditional operating system.
4) The network is programmable through software applications running on top of the NOS that interacts with the underlying data plane devices. This is a fundamental characteristic of SDN, considered as its main value proposition.

Note that the logical centralization of the control logic, in particular, offers several additional benefits. First, it is simpler and less error prone to modify network policies through high-level languages and software components, compared with low-level device specific configurations. Second, a control program can automatically react to spurious changes of the network state and thus maintain the high-level policies intact. Third, the centralization of the control logic in a controller with global knowledge of the network state simplifies the development of more sophisticated networking functions, services, and applications.

Following the SDN concept introduced in [5], an SDN can be defined by three fundamental abstractions: forwarding, distribution, and specification. In fact, abstractions are essential tools of research in computer science and information technology, being already an ubiquitous feature of many computer architectures and systems [28].

Ideally, the forwarding abstraction should allow any forwarding behavior desired by the network application (the control program) while hiding details of the underlying hardware. OpenFlow is one realization of such abstraction, which can be seen as the equivalent to a "device driver" in an operating system.

The distribution abstraction should shield SDN applications from the vagaries of distributed state, making the distributed control problem a logically centralized one. Its realization requires a common distribution layer, which in SDN resides in the NOS. This layer has two essential functions. First, it is responsible for installing the control commands on the forwarding devices. Second, it collects status information about the forwarding layer (network devices and links), to offer a global network view to network applications.

The last abstraction is specification, which should allow a network application to express the desired network behavior without being responsible for implementing that behavior itself. This can be achieved through virtualization solutions, as well as network programming languages. These approaches map the abstract configurations that the applications express based on a simplified, abstract model of the network, into a physical configuration for the global network view exposed by the SDN controller. Fig. 4 depicts the SDN architecture, concepts, and building blocks.

As previously mentioned, the strong coupling between control and data planes has made it difficult to add new functionality to traditional networks, a fact illustrated in Fig. 5. The coupling of the control and data planes (and its physical embedding in the network elements) makes the development and deployment of new networking features

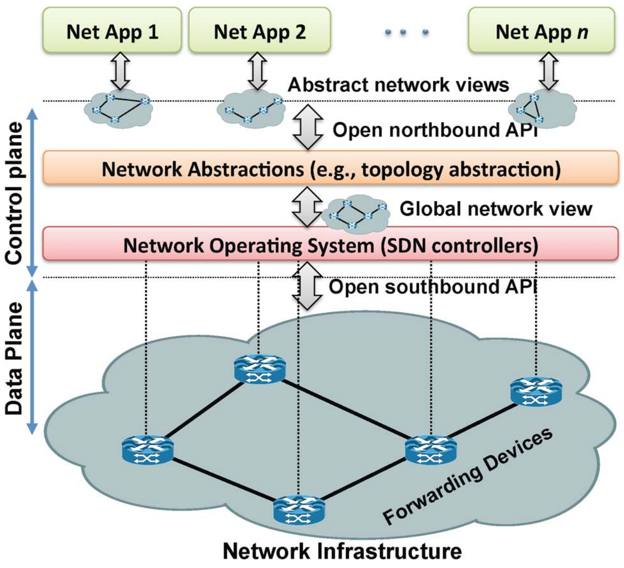

Fig. 4. *SDN architecture and its fundamental abstractions.*

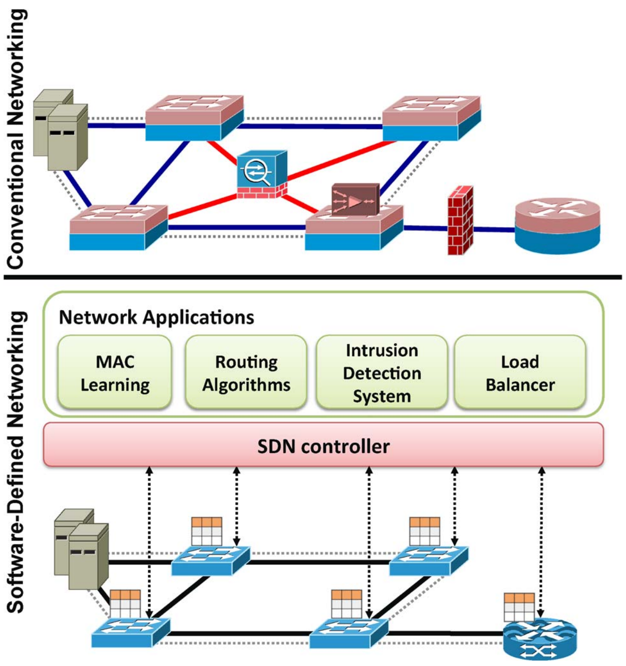

Fig. 5. *Traditional networking versus SDN. With SDN, management becomes simpler and middleboxes services can be delivered as SDN controller applications.*

(e.g., routing algorithms) very difficult, since it would imply a modification of the control plane of all network devices—through the installation of new firmware and, in some cases, hardware upgrades. Hence, the new networking features are commonly introduced via expensive, specialized, and hard-to-configure equipment (also known as middleboxes) such as load balancers, intrusion detection systems (IDSs), and firewalls, among others. These middleboxes need to be placed strategically in the network, making it even harder to later change the network topology, configuration, and functionality.

In contrast, SDN decouples the control plane from the network devices and becomes an external entity: the NOS or SDN controller. This approach has several advantages.

- It becomes easier to program these applications since the abstractions provided by the control platform and/or the network programming languages can be shared.
- All applications can take advantage of the same network information (the global network view), leading (arguably) to more consistent and effective policy decisions, while reusing control plane software modules.
- These applications can take actions (i.e., reconfigure forwarding devices) from any part of the network. There is therefore no need to devise a precise strategy about the location of the new functionality.
- The integration of different applications becomes more straightforward [29]. For instance, load balancing and routing applications can be combined sequentially, with load balancing decisions having precedence over routing policies.

A. Terminology

To identify the different elements of an SDN as unequivocally as possible, we now present the essential terminology used throughout this work.

1) Forwarding Devices (FD): These are hardware- or software-based data plane devices that perform a set of elementary operations. The forwarding devices have well-defined instruction sets (e.g., flow rules) used to take actions on the incoming packets (e.g., forward to specific ports, drop, forward to the controller, rewrite some header). These instructions are defined by southbound interfaces (e.g., OpenFlow [9], ForCES [30], protocol-oblivious forwarding (POF) [31]) and are installed in the forwarding devices by the SDN controllers implementing the southbound protocols.

2) Data Plane (DP): Forwarding devices are interconnected through wireless radio channels or wired cables. The network infrastructure comprises the interconnected forwarding devices, which represent the data plane.

3) Southbound Interface (SI): The instruction set of the forwarding devices is defined by the southbound API, which is part of the southbound interface. Furthermore, the SI also defines the communication protocol between forwarding devices and control plane elements. This protocol formalizes the way the control and data plane elements interact.

4) Control Plane (CP): Forwarding devices are programmed by control plane elements through well-defined SI embodiments. The control plane can therefore be seen as the "network brain." All control logic rests in the applications and controllers, which form the control plane.

5) Northbound Interface (NI): The NOS can offer an API to application developers. This API represents a northbound interface, i.e., a common interface for developing applications. Typically, a northbound interface abstracts the low-level instruction sets used by southbound interfaces to program forwarding devices.

6) Management Plane (MP): The management plane is the set of applications that leverage the functions offered by the NI to implement network control and operation logic. This includes applications such as routing, firewalls, load balancers, monitoring, and so forth. Essentially, a management application defines the policies, which are ultimately translated to southbound-specific instructions that program the behavior of the forwarding devices.

B. Alternative and Broadening Definitions

Since its inception in 2010 [24], the original OpenFlow-centered SDN term has seen its scope broadened beyond architectures with a cleanly decoupled control plane interface. The definition of SDN will likely continue to broaden, driven by the industry business-oriented views on SDN—irrespective of the decoupling of the control plane. In this survey, we focus on the original, "canonical" SDN definition based on the aforementioned key pillars and the concept of layered abstractions. However, for the sake of completeness and clarity, we acknowledge alternative SDN definitions [32], as follows.

1) Control Plane/Broker SDN: A networking approach that retains existing distributed control planes but offers new APIs that allow applications to interact (bidirectionally) with the network. An SDN controller—often called orchestration platform—acts as a broker between the applications and the network elements. This approach effectively presents control plane data to the application and allows a certain degree of network programmability by means of "plug-ins" between the orchestrator function and network protocols. This API-driven approach corresponds to a hybrid model of SDN, since it enables the broker to manipulate and directly interact with the control planes of devices such as routers and switches. Examples of this view on SDN include recent standardization efforts at the Internet Engineering Task Force (IETF) (see Section III-C) and the design philosophy behind the OpenDaylight project [13] that goes beyond the OpenFlow split control mode.

2) Overlay SDN: This is a networking approach where the (software- or hardware-based) network edge is dynamically programmed to manage tunnels between hypervisors and/or network switches, introducing an overlay network. In this hybrid networking approach, the distributed control plane providing the underlay remains untouched. The centralized control plane provides a logical overlay that utilizes the underlay as a transport network. This flavor of SDN follows a proactive model to install the overlay tunnels. The overlay tunnels usually terminate inside virtual switches within hypervisors or in physical devices acting as gateways to the existing network. This approach is very popular in recent data center network virtualization [33], and are based on a variety of tunneling technologies (e.g., stateless transport tunneling [34], virtualized layer 2 networks (VXLAN) [35], network virtualization using generic routing encapsulation (NVGRE) [36], locator/ID separation protocol (LISP) [37], [38], and generic network virtualization encapsulation (GENEVE) [39]) [40].

Recently, other attempts to define SDN in a layered approach have appeared [16], [41]. From a practical perspective and trying to keep backward compatibility with existing network management approaches, one initiative in the IRTF Software-Defined Networking Research Group (SDNRG) [41] proposes a management plane at the same level of the control plane, i.e., it classifies solutions in two categories: control logic (with control plane southbound interfaces) and management logic (with management plane southbound interfaces). In other words, the management plane can be seen as a control platform that accommodates traditional network management services and protocols, such as SNMP [18], BGP [42], path computation element communication protocol (PCEP) [43], and network configuration protocol (NETCONF) [44].

In addition to the broadening definitions above, the term SDN is often used to define extensible network management planes (e.g., OpenStack [45]), whitebox/bare-metal switches with open operating systems (e.g., Cumulus Linux), open-source data planes (e.g., Pica8 Xorplus [46], Quagga [47]), specialized programmable hardware devices (e.g., NetFPGA [48]), virtualized software-based appliances (e.g., open platform for network functions virtualization (OPNFV) [49]), in spite of lacking a decoupled control and data plane or common interface along its API. Hybrid SDN models are further discussed in Section V-G.

C. Standardization Activities

The standardization landscape in SDN (and SDN-related issues) is already wide and is expected to keep evolving over time. While some of the activities are being carried out in standard development organizations (SDOs), other related efforts are ongoing at industrial or community consortia (e.g., OpenDaylight, OpenStack, OPNFV), delivering results often considered candidates for *de facto* standards. These results often come in the form of open source implementations that have become the common strategy toward accelerating SDN and related cloud and networking technologies [50]. The reason for this fragmentation is due to SDN concepts spanning different areas of IT and networking, both from a network segmentation point of view (from access to core) and from a technology perspective (from optical to wireless).

Table 1 presents a summary of the main SDOs and organizations contributing to the standardization of SDN, as well as the main outcomes produced to date.

The ONF was conceived as a member-driven organization to promote the adoption of SDN through the development of the OpenFlow protocol as an open standard to communicate control decisions to data plane devices. The ONF is structured in several working groups (WGs). Some WGs are focused on either defining extensions to the OpenFlow protocol in general, such as the extensibility WG, or tailored to specific technological areas. Examples of the latter include the optical transport (OT) WG, the wireless and mobile (W&M) WG, and the northbound interfaces (NBI) WG. Other WGs center their activity in providing new protocol capabilities to enhance the protocol itself, such as the architecture WG or the forwarding abstractions (FA) WG.

Table 1 Openflow Standardization Activities

SDO	Working Group	Focus	Outcomes
ONF	Architecture & Framework	SDN architecture, defining architectural components and interfaces	SDN Architecture [51]
	Northbound Interfaces	Definition of standard NBIs for SDN controllers	
	Testing and Interoperability	Specification of OpenFlow conformance test suites	Conformance tests [52]
	Extensibility	Development of extensions to OpenFlow protocol, producing specifications of the OpenFlow switch (OF-WIRE) protocol	OF-WIRE 1.4.0 [53]
	Configuration & Management	OAM (operation, administration, and management) capabilities for OF protocol, producing specifications of the OF Configuration and Management (OF-CONFIG) protocol	OF-CONFIG 1.2 [54] OpenFlow Notifications Framework [55]
	Forwarding Abstractions	Development of hardware abstractions and simplification of behavioral descriptions mapping	OpenFlow Table Type Patterns [56]
	Optical Transport	Specification of SDN and control capabilities for optical transport networks by means of OpenFlow	Use cases [57] Requirements [58]
	Wireless & Mobile	Specification of SDN and control capabilities for wireless and mobile networks by means of OpenFlow	
	Migration	Methods to migrate from conventional networks to SDN-based networks based on OpenFlow	Use cases [59]
	Market Education	Dissemination of ONF initiatives in SDN and OpenFlow by releasing White Papers and Solution Briefs	SDN White Paper [60]
IETF	Application-Layer Traffic Optimization (ALTO)	Provides applications with network state information	Architectures for the coexistence of SDN and ALTO [61]
	Forwarding and Control Element Separation (ForCES)	Protocol specifications for the communication between control and forwarding elements.	Protocol specification [30]
	Interface to the Routing System (I2RS)	Real-time or event driven interaction with the routing system in an IP routed network	Architecture [62]
	Network Configuration (NETCONF)	Protocol specification for transferring configuration data to and from a device	NETCONF protocol [63]
	Network Virtualization Overlays (NVO3)	Overlay networks for supporting multi-tenancy in the context of data center communications (i.e., VM communication)	Control plane requirements [64]
	Path Computation Element (PCE)	Path computation for traffic engineering and path selection based on constrains	ABNO framework [65] Cross stratum path computation [66]
	Source Packet Routing in Networking (SPRING)	Specification of a forwarding path at the source of traffic	OpenFlow interworking [67] SDN controlled use cases [68]
	Abstraction and Control of Transport Networks (ACTN) BoF	Facilitate a centralized virtual network operation	Virtual network controller framework [69]
IRTF	Software-Defined Networking Research Group (SDNRG)	Prospection of SDN for the evolution of Internet	SDN operator perspective [70] SDN Architecture [71] Service / Transport separation [72]
ITU-T	SG 11	Signalling requirements using SDN technologies in Broadband Access Networks	Q.Supplement-SDN [73] Q.SBAN [74]
	SG 13	Functional requirements and architecture for SDN and networks of the future	Recommendation Y.3300 [75]
	SG 15	Specification of a transport network control plane architecture to support SDN control of transport networks	
	SG 17	Architectural aspects of security in SDN and security services using SDN	
BBF	Service Innovation and Market Requirements	Requirements and impacts of deploying SDN in broadband networks	SD-313 [76]
MEF	The Third Network	Service orchestration in Network as a Service and NFV environments	
IEEE	802	Applicability of SDN to IEEE 802 infrastructure	
OIF	Carrier WG	Transport SDN networks	Requirements for SDN enabled transport networks [77]
ODCA	SDN/Infrastructure	Requirements for SDN in cloud environments	Usage model [78]
ETSI	NFV ISG	Orchestration of network functions, including the combined control of computing, storage and networking resources	NFV Architecture [79]
ATIS	SDN Focus Group	Operational aspects of SDN and NFV	Operation of SDN [80]

Similar to how network programmability ideas have been considered by several IETF working groups (WGs) in the past, the present SDN trend is also influencing a number of activities. A related body that focuses on research aspects for the evolution of the Internet, IRTF, has created the SDNRG. This group investigates SDN from various perspectives with the goal of identifying the approaches that can be defined, deployed, and used in the near term, as well as identifying future research challenges.

In the International Telecommunications Union's Telecommunication sector (ITU–T), some study groups (SGs) have already started to develop recommendations for SDN, and a Joint Coordination Activity on SDN (JCA-SDN) has been established to coordinate the SDN standardization work.

The Broadband Forum (BBF) is working on SDN topics through the Service Innovation & Market Requirements (SIMR) WG. The objective of the BBF is to release recommendations for supporting SDN in multiservice broadband networks, including hybrid environments where only some of the network equipment is SDN enabled.

The Metro Ethernet Forum (MEF) is approaching SDN with the aim of defining service orchestration with APIs for existing networks.

At the IEEE, the 802 LAN/MAN Standards Committee has recently started some activities to standardize SDN capabilities on access networks based on IEEE 802 infrastructure through the P802.1CF project, for both wired and wireless technologies to embrace new control interfaces.

The Optical Internetworking Forum (OIF) Carrier WG released a set of requirements for transport SDN. The initial activities have as main goal to describe the features and functionalities needed to support the deployment of SDN capabilities in carrier transport networks. The Open Data Center Alliance (ODCA) is an organization working on unifying data center in the migration to cloud computing environments through interoperable solutions. Through the documentation of usage models, specifically one for SDN, the ODCA is defining new requirements for cloud deployment. The Alliance for Telecommunication Industry Solutions (ATIS) created a focus group for analyzing operational issues and opportunities associated with the programmable capabilities of network infrastructure.

At the European Telecommunication Standards Institute (ETSI), efforts are being devoted to network function virtualization (NFV) through a newly defined Industry Specification Group (ISG). NFV and SDN concepts are considered complementary, sharing the goal of accelerating innovation inside the network by allowing programmability, and altogether changing the network operational model through automation and a real shift to software-based platforms.

Finally, the mobile networking industry 3rd Generation Partnership Project consortium is studying the management of virtualized networks, an effort aligned with the ETSI NFV architecture and, as such, likely to leverage from SDN.

D. History of SDN

Albeit a fairly recent concept, SDN leverages on networking ideas with a longer history [17]. In particular, it builds on work made on programmable networks, such as active networks [81], programmable ATM networks [82], [83], and on proposals for control and data plane separation, such as the network control point (NCP) [84] and routing control platform (RCP) [85].

In order to present a historical perspective, we summarize in Table 2 different instances of SDN-related work prior to SDN, splitting it into five categories. Along with the categories we defined, the second and third columns of the table mention past initiatives (pre-SDN, i.e., before the OpenFlow-based initiatives that sprung into the SDN concept) and recent developments that led to the definition of SDN.

Data plane programmability has a long history. Active networks [81] represent one of the early attempts on building new network architectures based on this concept. The main idea behind active networks is for each node to have the capability to perform computations on, or modify the content of, packets. To this end, active networks propose two distinct approaches: programmable switches and capsules. The former does not imply changes in the existing packet or cell format. It assumes that switching devices support the downloading of programs with specific instructions on how to process packets. The second approach, on the other hand, suggests that packets should be replaced by tiny programs, which are encapsulated in transmission frames and executed at each node along their path.

ForCES [30], OpenFlow [9], and POF [31] represent recent approaches for designing and deploying programmable data plane devices. In a manner different from

Table 2 Summarized Overview of the History of Programmable Networks

Category	Pre-SDN initiatives	More recent SDN developments
Data plane programmability	xbind [82], IEEE P1520 [86], smart packets [87], ANTS [88], SwitchWare [89], Calvert [90], high performance router [91], NetScript [92], Tennenhouse [93]	ForCES [30], OpenFlow [9], POF [31]
Control and data plane decoupling	NCP [84], GSMP [94], [95], Tempest [96], ForCES [30], RCP [85], SoftRouter [97], PCE [43], 4D [98], IRSCP [99]	SANE [100], Ethane [101], OpenFlow [9], NOX [26], POF [31]
Network virtualization	Tempest [96], MBone [102], 6Bone [103], RON [104], Planet Lab [105], Impasse [106], GENI [107], VINI [108]	Open vSwitch [109], Mininet [110], FlowVisor [111], NVP [112]
Network operating systems	Cisco IOS [113], JUNOS [114]. ExtremeXOS [115], SR OS [116]	NOX [26], Onix [7], ONOS [117]
Technology pull initiatives	Open Signaling [118]	ONF [10]

active networks, these new proposals rely essentially on modifying forwarding devices to support flow tables, which can be dynamically configured by remote entities through simple operations such as adding, removing, or updating flow rules, i.e., entries on the flow tables.

The earliest initiatives on separating data and control signaling date back to the 1980s and 1990s. The NCP [84] is probably the first attempt to separate control and data plane signaling. NCPs were introduced by AT&T to improve the management and control of its telephone network. This change promoted a faster pace of innovation of the network and provided new means for improving its efficiency, by taking advantage of the global view of the network provided by NCPs. Similarly, other initiatives such as Tempest [96], ForCES [30], RCP [85], and PCE [43] proposed the separation of the control and data planes for improved management in ATM, Ethernet, BGP, and multiprotocol label switching (MPLS) networks, respectively.

More recently, initiatives such as SANE [100], Ethane [101], OpenFlow [9], NOX [26], and POF [31] proposed the decoupling of the control and data planes for Ethernet networks. Interestingly, these recent solutions do not require significant modifications on the forwarding devices, making them attractive not only for the networking research community, but even more to the networking industry. OpenFlow-based devices [9], for instance, can easily coexist with traditional Ethernet devices, enabling a progressive adoption (i.e., not requiring a disruptive change to existing networks).

Network virtualization has gained a new traction with the advent of SDN. Nevertheless, network virtualization also has its roots back in the 1990s. The Tempest project [96] is one of the first initiatives to introduce network virtualization, by introducing the concept of switchlets in ATM networks. The core idea was to allow multiple switchlets on top of a single ATM switch, enabling multiple independent ATM networks to share the same physical resources. Similarly, MBone [102] was one of the early initiatives that targeted the creation of virtual network topologies on top of legacy networks, or overlay networks. This work was followed by several other projects such as Planet Lab [105], GENI [107], and VINI [108]. FlowVisor [119] is also worth mentioning as one of the first recent initiatives to promote a hypervisor-like virtualization architecture for network infrastructures, resembling the hypervisor model common for compute and storage. More recently, Koponen *et al.* proposed a network virtualization platform (NVP) [112] for multitenant data centers using SDN as a base technology.

The concept of a NOS was reborn with the introduction of OpenFlow-based NOSs, such as NOX [26], Onix [7], and ONOS [117]. Indeed, NOSs have been in existence for decades. One of the most widely known and deployed is the Cisco IOS [113], which was originally conceived back in the early 1990s. Other NOSs worth mentioning are JUNOS [114], ExtremeXOS [115], and SR OS [116]. Despite being more specialized NOSs, targeting network devices such as high-performance core routers, these NOSs abstract the underlying hardware to the network operator, making it easier to control the network infrastructure as well as simplifying the development and deployment of new protocols and management applications.

Finally, initiatives that can be seen as "technology pull" drivers are also worth recalling. Back in the 1990s, a movement toward open signaling [118] began to happen. The main motivation was to promote the wider adoption of the ideas proposed by projects such as NCP [84] and Tempest [96]. The open signaling movement worked toward separating the control and data signaling by proposing open and programmable interfaces. Curiously, a rather similar movement can be observed with the recent advent of OpenFlow and SDN, with the lead of the ONF [10]. This type of movement is crucial to promote open technologies into the market, hopefully leading equipment manufacturers to support open standards and thus fostering interoperability, competition, and innovation.

For a more extensive intellectual history of programmable networks and SDN, we direct the reader to the recent paper by Feamster *et al.* [17].

IV. SOFTWARE-DEFINED NETWORKS: BOTTOM-UP

An SDN architecture can be depicted as a composition of different layers, as shown in Fig. 6(b). Each layer has its own specific functions. While some of them are always present in an SDN deployment, such as the southbound API, NOSs, northbound API, and network applications, others may be present only in particular deployments, such as hypervisor- or language-based virtualization.

Fig. 6 presents a trifold perspective of SDNs. The SDN layers are represented in Fig. 6(b), as explained above. Fig. 6(a) and (c) depicts a plane-oriented view and a system design perspective, respectively.

The following sections introduce each layer, following a bottom-up approach. For each layer, the core properties and concepts are explained based on the different technologies and solutions. Additionally, debugging and troubleshooting techniques and tools are discussed.

A. Layer I: Infrastructure

An SDN infrastructure, similarly to a traditional network, is composed of a set of networking equipment (switches, routers, and middlebox appliances). The main difference resides in the fact that those traditional physical devices are now simple forwarding elements without embedded control or software to take autonomous decisions. The network intelligence is removed from the data plane devices to a logically centralized control system, i.e., the NOS and applications, as shown in Fig. 6(c). More importantly, these new networks are built (conceptually) on top

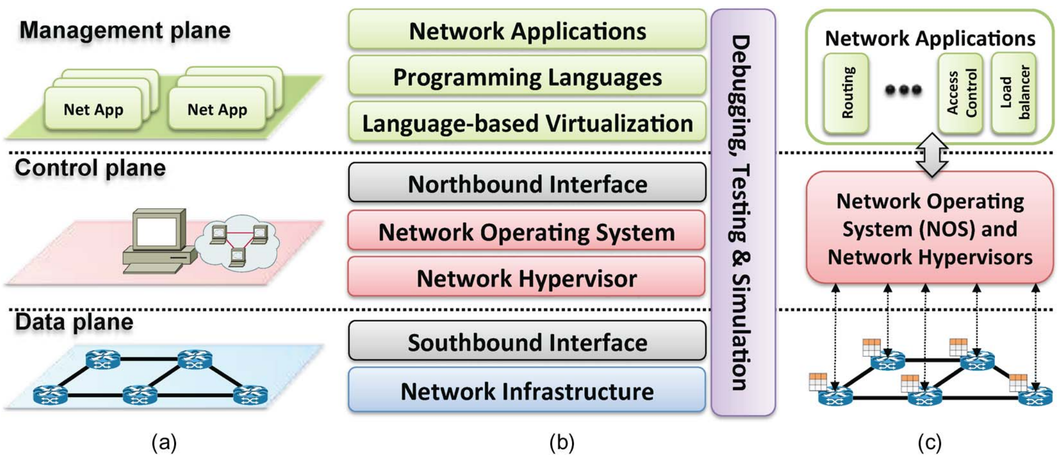

Fig. 6. *Software-Defined Networks in (a) planes, (b) layers, and (c) system design architecture.*

of open and standard interfaces (e.g., OpenFlow), a crucial approach for ensuring configuration and communication compatibility and interoperability among different data and control plane devices. In other words, these open interfaces enable controller entities to dynamically program heterogeneous forwarding devices, something difficult in traditional networks, due to the large variety of proprietary and closed interfaces and the distributed nature of the control plane.

In an SDN/OpenFlow architecture, there are two main elements, the controllers and the forwarding devices, as shown in Fig. 7. A data plane device is a hardware or software element specialized in packet forwarding, while a controller is a software stack (the "network brain") running on a commodity hardware platform. An OpenFlow-enabled forwarding device is based on a pipeline of flow tables where each entry of a flow table has three parts: 1) a matching rule; 2) actions to be executed on matching packets; and 3) counters that keep statistics of matching packets. This high-level and simplified model derived from OpenFlow is currently the most widespread design of SDN data plane devices. Nevertheless, other specifications of SDN-enabled forwarding devices are being pursued, including POF [31], [120] and the negotiable datapath models (NDMs) from the ONF Forwarding Abstractions Working Group (FAWG) [121].

Inside an OpenFlow device, a path through a sequence of flow tables defines how packets should be handled. When a new packet arrives, the lookup process starts in the first table and ends either with a match in one of the tables of the pipeline or with a miss (when no rule is found for that packet). A flow rule can be defined by combining different matching fields, as illustrated in Fig. 7. If there is no default rule, the packet will be discarded. However, the common case is to install a default rule which tells the switch to send the packet to the controller (or to the normal non-OpenFlow pipeline of the switch). The priority of the rules follows the natural sequence number of the tables and the row order in a flow table. Possible actions include: 1) forward the packet to outgoing port(s);

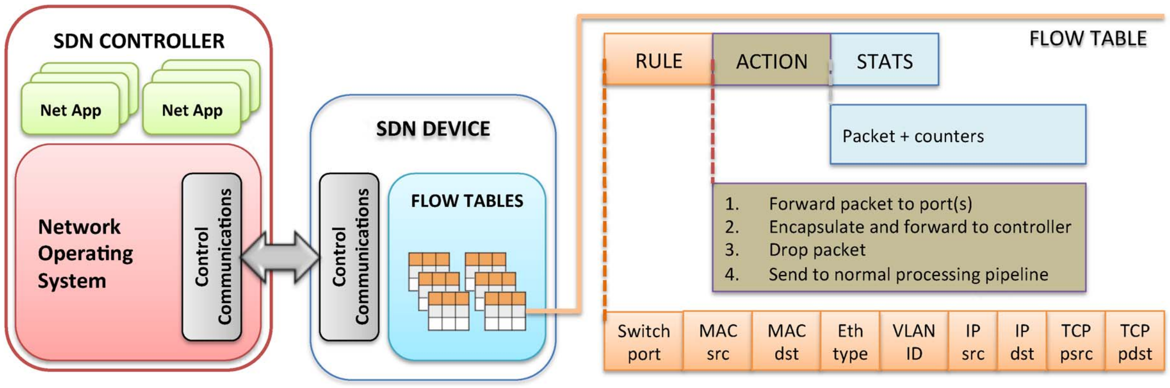

Fig. 7. *OpenFlow-enabled SDN devices.*

Table 3 Different Match Fields, Statistics, and Capabilities Have Been Added on Each Openflow Protocol Revision. The Number of Required (REQ) and Optional (OPT) Capabilities Has Grown Considerably

OpenFlow Version	Match fields	Statistics	# Matches		# Instructions		# Actions		# Ports	
			Req	Opt	Req	Opt	Req	Opt	Req	Opt
v 1.0	Ingress Port	Per table statistics	18	2	1	0	2	11	6	2
	Ethernet: src, dst, type, VLAN	Per flow statistics								
	IPv4: src, dst, proto, ToS	Per port statistics								
	TCP/UDP: src port, dst port	Per queue statistics								
v 1.1	Metadata, SCTP, VLAN tagging	Group statistics	23	2	0	0	3	28	5	3
	MPLS: label, traffic class	Action bucket statistics								
v 1.2	OpenFlow Extensible Match (OXM)		14	18	2	3	2	49	5	3
	IPv6: src, dst, flow label, ICMPv6									
v 1.3	PBB, IPv6 Extension Headers	Per-flow meter	14	26	2	4	2	56	5	3
		Per-flow meter band								
v 1.4	—	—	**14**	**27**	**2**	**4**	**2**	**57**	**5**	**3**
		Optical port properties								

2) encapsulate it and forward it to the controller; 3) drop it; 4) send it to the normal processing pipeline; and 5) send it to the next flow table or to special tables, such as group or metering tables introduced in the latest OpenFlow protocol.

As detailed in Table 3, each version of the OpenFlow specification introduced new match fields including Ethernet, IPv4/v6, MPLS, TCP/UDP, etc. However, only a subset of those matching fields are mandatory to be compliant to a given protocol version. Similarly, many actions and port types are optional features. Flow match rules can be based on almost arbitrary combinations of bits of the different packet headers using bit masks for each field. Adding new matching fields has been eased with the extensibility capabilities introduced in OpenFlow version 1.2 through an OpenFlow Extensible Match (OXM) based on type-length-value (TLV) structures. To improve the overall protocol extensibility, with OpenFlow version 1.4, TLV structures have been also added to ports, tables, and queues in replacement of the hard-coded counterparts of earlier protocol versions.

1) Overview of Available OpenFlow Devices: Several OpenFlow-enabled forwarding devices are available on the market, both as commercial and open source products (see Table 4). There are many off-the-shelf, ready to deploy, OpenFlow switches and routers, among other appliances. Most of the switches available on the market have relatively small ternary content-addressable memory (TCAMs), with up to 8000 entries. Nonetheless, this is changing at a fast pace. Some of the latest devices released in the market go far beyond that figure. Gigabit Ethernet (GbE) switches for common business purposes are already supporting up to 32 000 Layer 2 (L2) + Layer 3 (L3) or 64 000 L2/L3 exact match flows [122]. Enterprise class 10GbE switches are being delivered with more than 80 000 layer 2 flow entries [123]. Other switching devices using high-performance chips (e.g., EZchip NP-4) provide optimized TCAM memory that supports from 125 000 up to 1 000 000 flow table entries [124]. This is a clear sign that the size of the flow tables is growing at a pace aiming to meet the needs of future SDN deployments. Networking hardware manufacturers have produced various kinds of OpenFlow-enabled devices, as is shown in Table 4. These devices range from equipment for small businesses (e.g., GbE switches) to high-class data center equipment (e.g., high-density switch chassis with up to 100GbE connectivity for edge-to-core applications, with tens of terabits per second of switching capacity).

Software switches are emerging as one of the most promising solutions for data centers and virtualized network infrastructures [147]–[149]. Examples of software-based OpenFlow switch implementations include Switch Light [145], ofsoftswitch13 [141], Open vSwitch [142], OpenFlow Reference [143], Pica8 [150], Pantou [146], and XorPlus [46]. Recent reports show that the number of virtual access ports is already larger than physical access ports on data centers [149]. Network virtualization has been one of the drivers behind this trend. Software switches such as Open vSwitch have been used for moving network functions to the edge (with the core performing traditional IP forwarding), thus enabling network virtualization [112].

An interesting observation is the number of small, startup enterprises devoted to SDN, such as Big Switch, Pica8, Cyan, Plexxi, and NoviFlow. This seems to imply that SDN is springing a more competitive and open networking market, one of its original goals. Other effects of this openness triggered by SDN include the emergence of so-called "bare metal switches" or "whitebox switches," where software and hardware are sold separately and the end user is free to load an operating system of its choice [151].

B. Layer II: Southbound Interfaces

Southbound interfaces (or southbound APIs) are the connecting bridges between control and forwarding

Table 4 OpenFlow Enabled Hardware and Software Devices

Group	Product	Type	Maker/Developer	Version	Short description
Hardware	8200zl and 5400zl [125]	chassis	Hewlett-Packard	v1.0	Data center class chassis (switch modules).
	Arista 7150 Series [126]	switch	Arista Networks	v1.0	Data centers hybrid Ethernet/OpenFlow switches.
	BlackDiamond X8 [127]	switch	Extreme Networks	v1.0	Cloud-scale hybrid Ethernet/OpenFlow switches.
	CX600 Series [128]	router	Huawei	v1.0	Carrier class MAN routers.
	EX9200 Ethernet [129]	chassis	Juniper	v1.0	Chassis based switches for cloud data centers.
	EZchip NP-4 [130]	chip	EZchip Technologies	v1.1	High performance 100-Gigabit network processors.
	MLX Series [131]	router	Brocade	v1.0	Service providers and enterprise class routers.
	NoviSwitch 1248 [124]	switch	NoviFlow	v1.3	High performance OpenFlow switch.
	NetFPGA [48]	card	NetFPGA	v1.0	1G and 10G OpenFlow implementations.
	RackSwitch G8264 [132]	switch	IBM	v1.0	Data center switches supporting Virtual Fabric and OpenFlow.
	PF5240 and PF5820 [133]	switch	NEC	v1.0	Enterprise class hybrid Ethernet/OpenFlow switches.
	Pica8 3920 [134]	switch	Pica8	v1.0	Hybrid Ethernet/OpenFlow switches.
	Plexxi Switch 1 [135]	switch	Plexxi	v1.0	Optical multiplexing interconnect for data centers.
	V330 Series [136]	switch	Centec Networks	v1.0	Hybrid Ethernet/OpenFlow switches.
	Z-Series [137]	switch	Cyan	v1.0	Family of packet-optical transport platforms.
Software	contrail-vrouter [138]	vrouter	Juniper Networks	v1.0	Data-plane function to interface with a VRF.
	LINC [139], [140]	switch	FlowForwarding	v1.4	Erlang-based soft switch with OF-Config 1.1 support.
	ofsoftswitch13 [141]	switch	Ericsson, CPqD	v1.3	OF 1.3 compatible user-space software switch implementation.
	Open vSwitch [142], [109]	switch	Open Community	v1.0-1.3	Switch platform designed for virtualized server environments.
	OpenFlow Reference [143]	switch	Stanford	v1.0	OF Switching capability to a Linux PC with multiple NICs.
	OpenFlowClick [144]	vrouter	Yogesh Mundada	v1.0	OpenFlow switching element for Click software routers.
	Switch Light [145]	switch	Big Switch	v1.0	Thin switching software platform for physical/virtual switches.
	Pantou/OpenWRT [146]	switch	Stanford	v1.0	Turns a wireless router into an OF-enabled switch.
	XorPlus [46]	switch	Pica8	v1.0	Switching software for high performance ASICs.

elements, thus being the crucial instrument for clearly separating control and data plane functionality. However, these APIs are still tightly tied to the forwarding elements of the underlying physical or virtual infrastructure.

Typically, a new switch can take two years to be ready for commercialization if built from scratch, with upgrade cycles that can take up to nine months. The software development for a new product can take from six months to one year [152]. The initial investment is high and risky. As a central component of its design, the southbound APIs represent one of the major barriers for the introduction and acceptance of any new networking technology. In this light, the emergence of SDN southbound API proposals such as OpenFlow [9] is seen as welcome by many in the industry. These standards promote interoperability, allowing the deployment of vendor-agnostic network devices. This has already been demonstrated by the interoperability between OpenFlow-enabled equipments from different vendors.

As of this writing, OpenFlow is the most widely accepted and deployed open southbound standard for SDN. It provides a common specification to implement OpenFlow-enabled forwarding devices, and for the communication channel between data and control plane devices (e.g., switches and controllers). The OpenFlow protocol provides three information sources for NOSs. First, event-based messages are sent by forwarding devices to the controller when a link or port change is triggered. Second, flow statistics are generated by the forwarding devices and collected by the controller. Third, packet-in messages are sent by forwarding devices to the controller when they do not known what to do with a new incoming flow or because there is an explicit "send to controller" action in the matched entry of the flow table. These information channels are the essential means to provide flow-level information to the NOS.

Albeit the most visible, OpenFlow is not the only available southbound interface for SDN. There are other API proposals such as ForCES [30], Open vSwitch Database (OVSDB) [153], POF [31], [120], OpFlex [154], OpenState [155], revised open-flow library (ROFL) [156], hardware abstraction layer (HAL) [157], [158], and programmable abstraction of data path (PAD) [159]. ForCES proposes a more flexible approach to traditional network management without changing the current architecture of the network, i.e., without the need of a logically centralized external controller. The control and data planes are separated, but can potentially be kept in the same network element. However, the control part of the network element can be upgraded on-the-fly with third-party firmware.

OVSDB [153] is another type of southbound API, designed to provide advanced management capabilities for Open vSwitches. Beyond OpenFlow's capabilities to configure the behavior of flows in a forwarding device, an Open vSwitch offers other networking functions. For instance, it allows the control elements to create multiple virtual switch instances, set quality of service (QoS) policies on interfaces, attach interfaces to the switches, configure tunnel interfaces on OpenFlow data paths, manage queues, and collect statistics. Therefore, the OVSDB is a complementary protocol to OpenFlow for Open vSwitch.

One of the first direct competitors of OpenFlow is POF [31], [120]. One of the main goals of POF is to enhance the current SDN forwarding plane. With OpenFlow, switches have to understand the protocol headers to extract the required bits to be matched with the flow tables entries. This parsing represents a significant burden for data plane devices, in particular if we consider that OpenFlow version 1.3 already contains more than 40 header fields. Besides this inherent complexity, backward compatibility issues may arise every time new header fields are included in or removed from the protocol. To achieve its goal, POF proposes a generic flow instruction set (FIS) that makes the forwarding plane protocol oblivious. A forwarding element does not need to know, by itself, anything about the packet format in advance. Forwarding devices are seen as white boxes with only processing and forwarding capabilities. In POF, packet parsing is a controller task that results in a sequence of generic keys and table lookup instructions that are installed in the forwarding elements. The behavior of data plane devices is, therefore, completely under the control of the SDN controller. Similar to a central processing unit (CPU) in a computer system, a POF switch is application and protocol agnostic.

A recent southbound interface proposal is OpFlex [154]. Contrary to OpenFlow (and similar to ForCES), one of the ideas behind OpFlex is to distribute part of the complexity of managing the network back to the forwarding devices, with the aim of improving scalability. Similar to OpenFlow, policies are logically centralized and abstracted from the underlying implementation. The differences between OpenFlow and OpFlex are a clear illustration of one of the important questions to be answered when devising a southbound interface: where to place each piece of the overall functionality.

In contrast to OpFlex and POF, OpenState [155] and ROFL [156] do not propose a new set of instructions for programming data plane devices. OpenState proposes extended finite machines (stateful programming abstractions) as an extension (superset) of the OpenFlow match/action abstraction. Finite state machines allow the implementation of several stateful tasks inside forwarding devices, i.e., without augmenting the complexity or overhead of the control plane. For instance, all tasks involving only local state, such as media access control (MAC) learning operations, port knocking, or stateful edge firewalls, can be performed directly on the forwarding devices without any extra control plane communication and processing delay. ROFL, on the other hand, proposes an abstraction layer that hides the details of the different OpenFlow versions, thus providing a clean API for software developers, simplifying application development.

HAL [157], [158] is not exactly a southbound API, but is closely related. Differently from the aforementioned approaches, HAL is rather a translator that enables a southbound API such as OpenFlow to control heterogeneous hardware devices. It thus sits between the southbound API and the hardware device. Recent research experiments with HAL have demonstrated the viability of SDN control in access networks such as Gigabit Ethernet passive optical networks (GEPONs) [160] and cable networks (DOCSISs) [161]. A similar effort to HAL is PAD [159], a proposal that goes a bit further by also working as a southbound API by itself. More importantly, PAD allows a more generic programming of forwarding devices by enabling the control of data path behavior using generic byte operations, defining protocol headers and providing function definitions.

C. Layer III: Network Hypervisors

Virtualization is already a consolidated technology in modern computers. The fast developments of the past decade have made virtualization of computing platforms mainstream. Based on recent reports, the number of virtual servers has already exceeded the number of physical servers [162], [112].

Hypervisors enable distinct virtual machines to share the same hardware resources. In a cloud infrastructure-as-a-service (IaaS), each user can have its own virtual resources, from computing to storage. This enabled new revenue and business models where users allocate resources on demand, from a shared physical infrastructure, at a relatively low cost. At the same time, providers make better use of the capacity of their installed physical infrastructures, creating new revenue streams without significantly increasing their capital expenditure and operational expenditure (OPEX) costs. One of the interesting features of virtualization technologies today is the fact that virtual machines can be easily migrated from one physical server to another and can be created and/or destroyed on demand, enabling the provisioning of elastic services with flexible and easy management. Unfortunately, virtualization has been only partially realized in practice. Despite the great advances in virtualizing computing and storage elements, the network is still mostly statically configured in a box-by-box manner [33].

The main network requirements can be captured along two dimensions: network topology and address space. Different workloads require different network topologies and services, such as flat L2 or L3 services, or even more complex L4–L7 services for advanced functionality. Currently, it is very difficult for a single physical topology to support the diverse demands of applications and services. Similarly, address space is hard to change in current networks. Today, virtualized workloads have to operate in the same address of the physical infrastructure. Therefore, it is hard to keep the original network configuration for a tenant, virtual machines cannot migrate to arbitrary locations, and the addressing scheme is fixed and hard to change. For example, IPv6 cannot be used by the virtual machines (VMs) of a tenant if the underlying physical forwarding devices support only IPv4.

To provide complete virtualization, the network should provide similar properties to the computing layer [33]. The network infrastructure should be able to support arbitrary network topologies and addressing schemes. Each tenant should have the ability to configure both the computing nodes and the network simultaneously. Host migration should automatically trigger the migration of the corresponding virtual network ports. One might think that long standing virtualization primitives such as VLANs (virtualized L2 domain), NAT (virtualized IP address space), and MPLS (virtualized path) are enough to provide full and automated network virtualization. However, these technologies are anchored on a box-by-box basis configuration, i.e., there is no single unifying abstraction that can be leveraged to configure (or reconfigure) the network in a global manner. As a consequence, current network provisioning can take months, while computing provisioning takes only minutes [112], [163]–[165].

There is hope that this situation will change with SDN and the availability of new tunneling techniques (e.g., VXLAN [35] and NVGRE [36]). For instance, solutions such as FlowVisor [111], [166], [167], FlowN [168], NVP [112], OpenVirteX [169], [170], IBM SDN VE [171], [172], RadioVisor [173], AutoVFlow [174], eXtensible Datapath Daemon (xDPd) [175], [176], optical transport network virtualization [177], and version-agnostic OpenFlow slicing mechanisms [178], have been recently proposed, evaluated, and deployed in real scenarios for on-demand provisioning of virtual networks.

1) Slicing the Network: FlowVisor is one of the early technologies to virtualize an SDN. Its basic idea is to allow multiple logical networks share the same OpenFlow networking infrastructure. For this purpose, it provides an abstraction layer that makes it easier to slice a data plane based on off-the-shelf OpenFlow-enabled switches, allowing multiple and diverse networks to coexist. Five slicing dimensions are considered in FlowVisor: bandwidth, topology, traffic, device CPU, and forwarding tables. Moreover, each network slice supports a controller, i.e., multiple controllers can coexist on top of the same physical network infrastructure. Each controller is allowed to act only on its own network slice. In general terms, a slice is defined as a particular set of flows on the data plane. From a system design perspective, FlowVisor is a transparent proxy that intercepts OpenFlow messages between switches and controllers. It partitions the link bandwidth and flow tables of each switch. Each slice receives a minimum data rate, and each guest controller gets its own virtual flow table in the switches.

Similarly to FlowVisor, OpenVirteX [169], [170] acts as a proxy between the NOS and the forwarding devices. However, its main goal is to provide virtual SDNs through topology, address, and control function virtualization. All these properties are necessary in multitenant environments where virtual networks need to be managed and migrated according to the computing and storage virtual resources. Virtual network topologies have to be mapped onto the underlying forwarding devices, with virtual addresses allowing tenants to completely manage their address space without depending on the underlying network elements addressing schemes.

AutoSlice [179] is another SDN-based virtualization proposal. Differently from FlowVisor, it focuses on the automation of the deployment and operation of virtual SDN (vSDN) topologies with minimal mediation or arbitration by the substrate network operator. Additionally, AutoSlice targets also scalability aspects of network hypervisors by optimizing resource utilization and by mitigating the flow-table limitations through a precise monitoring of the flow traffic statistics. Similarly to AutoSlice, AutoVFlow [174] also enables multidomain network virtualization. However, instead of having a single third party to control the mapping of vSDN topologies, as is the case of AutoSlice, AutoVFlow uses a multiproxy architecture that allows network owners to implement flow space virtualization in an autonomous way by exchanging information among the different domains.

FlowN [168], [180] is based on a slightly different concept. Whereas FlowVisor can be compared to a full virtualization technology, FlowN is analogous to a container-based virtualization, i.e., a lightweight virtualization approach. FlowN was also primarily conceived to address multitenancy in the context of cloud platforms. It is designed to be scalable and allows a unique shared controller platform to be used for managing multiple domains in a cloud environment. Each tenant has full control over its virtual networks and is free to deploy any network abstraction and application on top of the controller platform.

The compositional SDN hypervisor [181] was designed with a different set of goals. Its main objective is to allow the cooperative (sequential or parallel) execution of applications developed with different programming languages or conceived for diverse control platforms. It thus offers interoperability and portability in addition to the typical functions of network hypervisors.

2) Commercial Multitenant Network Hypervisors: None of the aforementioned approaches is designed to address all challenges of multitenant data centers. For instance, tenants want to be able to migrate their enterprise solutions to cloud providers without the need to modify the network configuration of their home network. Existing networking technologies and migration strategies have mostly failed to meet both tenant and service provider requirements. A multitenant environment should be anchored in a network hypervisor capable of abstracting the underlaying forwarding devices and physical network topology from the tenants. Moreover, each tenant should have access to control abstractions and manage its own virtual networks independently and isolated from other tenants.

With the market demand for network virtualization and the recent research on SDN showing promise as an enabling technology, different commercial virtualization platforms based on SDN concepts have started to appear. VMWare has proposed a network virtualization platform (NVP) [112] that provides the necessary abstractions to allow the creation of independent virtual networks for large-scale multitenant environments. NVP is a complete network virtualization solution that allows the creation of virtual networks, each with independent service model, topologies, and addressing architectures over the same physical network. With NVP, tenants do not need to know anything about the underlying network topology, configuration, or other specific aspects of the forwarding devices. NVP's network hypervisor translates the tenants configurations and requirements into low-level instruction sets to be installed on the forwarding devices. For this purpose, the platform uses a cluster of SDN controllers to manipulate the forwarding tables of the Open vSwitches in the host's hypervisor. Forwarding decisions are therefore made exclusively on the network edge. After the decision is made, the packet is tunneled over the physical network to the receiving host hypervisor (the physical network sees nothing but ordinary IP packets).

IBM has also recently proposed SDN VE [171], [172], another commercial and enterprise-class network virtualization platform. SDN VE uses OpenDaylight as one of the building blocks of the so-called software-defined environments (SDEs), a trend further discussed in Section V. This solution also offers a complete implementation framework for network virtualization. Like NVP, it uses a host-based overlay approach, achieving advanced network abstraction that enables application-level network services in large-scale multitenant environments. Interestingly, SDN VE 1.0 is capable of supporting in one single instantiation up to 16 000 virtual networks and 128 000 virtual machines [171], [172].

To summarize, currently there are already a few network hypervisor proposals leveraging the advances of SDN. There are, however, still several issues to be addressed. These include, among others, the improvement of virtual-to-physical mapping techniques [182], the definition of the level of detail that should be exposed at the logical level, and the support for nested virtualization [29]. We anticipate, however, this ecosystem to expand in the near future since network virtualization will most likely play a key role in future virtualized environments, similarly to the expansion we have been witnessing in virtualized computing.

D. Layer IV: Network Operating Systems/Controllers

Traditional operating systems provide abstractions (e.g., high-level programming APIs) for accessing lower level devices, manage the concurrent access to the underlying resources (e.g., hard drive, network adapter, CPU, memory), and provide security protection mechanisms. These functionalities and resources are key enablers for increased productivity, making the life of system and application developers easier. Their widespread use has significantly contributed to the evolution of various ecosystems (e.g., programming languages) and the development of a myriad of applications.

In contrast, networks have so far been managed and configured using lower level, device-specific instruction sets and mostly closed proprietary NOSs (e.g., Cisco IOS and Juniper JunOS). Moreover, the idea of operating systems abstracting device-specific characteristics and providing, in a transparent way, common functionalities is still mostly absent in networks. For instance, today designers of routing protocols need to deal with complicated distributed algorithms when solving networking problems. Network practitioners have therefore been solving the same problems over and over again.

SDN is promised to facilitate network management and ease the burden of solving networking problems by means of the logically centralized control offered by a NOS [26]. As with traditional operating systems, the crucial value of a NOS is to provide abstractions, essential services, and common APIs to developers. Generic functionality as network state and network topology information, device discovery, and distribution of network configuration can be provided as services of the NOS. With NOSs, to define network policies a developer no longer needs to care about the low-level details of data distribution among routing elements, for instance. Such systems can arguably create a new environment capable of fostering innovation at a faster pace by reducing the inherent complexity of creating new network protocols and network applications.

A NOS (or a controller) is a critical element in an SDN architecture as it is the key supporting piece for the control logic (applications) to generate the network configuration based on the policies defined by the network operator. Similar to a traditional operating system, the control platform abstracts the lower level details of connecting and interacting with forwarding devices (i.e., of materializing the network policies).

1) Architecture and Design Axes: There is a very diverse set of controllers and control platforms with different design and architectural choices [7], [13], [183]–[186]. Existing controllers can be categorized based on many aspects. From an architectural point of view, one of the most relevant is if they are centralized or distributed. This is one of the key design axes of SDN control platforms, so we start by discussing this aspect next.

2) Centralized Versus Distributed: A centralized controller is a single entity that manages all forwarding devices of the network. Naturally, it represents a single point of failure and may have scaling limitations. A single controller may not be enough to manage a network with a large number of data plane elements. Centralized controllers such as NOX–MT [187], Maestro [188], Beacon [186], and Floodlight [189] have been designed as highly concurrent systems, to achieve the throughput required by enterprise class networks and data centers. These controllers are based on multithreaded designs to explore the parallelism of multicore computer architectures. As an example, Beacon can deal with more than 12 million flows per second by using large size computing nodes of cloud providers such as Amazon [186]. Other centralized controllers such as Trema [190], Ryu NOS [191], Meridian [192], and ProgrammableFlow [133], [193] target specific environments such as data centers, cloud infrastructures, and carrier grade networks. Furthermore, controllers such as Rosemary [194] offer specific functionality and guarantees, namely security and isolation of applications. By using a container-based architecture called micro-NOS, it achieves its primary goal of isolating applications and preventing the propagation of failures throughout the SDN stack.

Contrary to a centralized design, a distributed NOS can be scaled up to meet the requirements of potentially any environment, from small- to large-scale networks. A distributed controller can be a centralized cluster of nodes or a physically distributed set of elements. While the first alternative can offer high throughput for very dense data centers, the latter can be more resilient to different kinds of logical and physical failures. A cloud provider that spans multiple data centers interconnected by a wide area network may require a hybrid approach, with clusters of controllers inside each data center and distributed controller nodes in the different sites [8].

Onix [7], HyperFlow [195], HP VAN SDN [184], ONOS [117], DISCO [185], *yanc* [196], PANE [197], SMaRt-Light [198], and Fleet [199] are examples of distributed controllers. Most distributed controllers offer weak consistency semantics, which means that data updates on distinct nodes will eventually be updated on all controller nodes. This implies that there is a period of time in which distinct nodes may read different values (old value or new value) for the same property. Strong consistency, on the other hand, ensures that all controller nodes will read the most updated property value after a write operation. Despite its impact on system performance, strong consistency offers a simpler interface to application developers. To date, only Onix, ONOS, and SMaRtLight provide this data consistency model.

Another common property of distributed controllers is fault tolerance. When one node fails, another neighbor node should take over the duties and devices of the failed node. So far, despite some controllers tolerating crash failures, they do not tolerate arbitrary failures, which means that any node with an abnormal behavior will not be replaced by a potentially well-behaved one.

A single controller may be enough to manage a small network, however it represents a single point of failure. Similarly, independent controllers can be spread across the network, each of them managing a network segment, reducing the impact of a single controller failure. Yet, if the control plane availability is critical, a cluster of controllers can be used to achieve a higher degree of availability and/or for supporting more devices. Ultimately, a distributed controller can improve the control plane resilience and scalability and reduce the impact of problems caused by network partition, for instance. SDN resiliency as a whole is an open challenge that will be further discussed in Section V-C.

3) Dissecting SDN Controller Platforms: To provide a better architectural overview and understanding the design of a NOS, Table 5 summarizes some of the most relevant architectural and design properties of SDN controllers and

Table 5 Architecture and Design Elements of Control Platforms

Component	**OpenDaylight**	**OpenContrail**	**HP VAN SDN**	**Onix**	**Beacon**
Base network services	Topology/Stats/Switch Manager, Host Tracker, Shortest Path Forwarding	Routing, Tenant Isolation	Audit Log, Alerts, Topology, Discovery	Discovery, Multi-consistency Storage, Read State, Register for updates	Topology, device manager, and routing
East/Westbound APIs	—	Control Node (XMPP-like control channel)	Sync API	Distribution I/O module	*Not present*
Integration Plug-ins	OpenStack Neutron	CloudStack, OpenStack	OpenStack	—	—
Management Interfaces	GUI/CLI, REST API	GUI/CLI	REST API Shell / GUI Shell	—	Web
Northbound APIs	REST, REST-CONF [200], Java APIs	REST APIs (configuration, operational, and analytic)	REST API, GUI Shell	Onix API (general purpose)	API (based on OpenFlow events)
Service abstraction layers	Service Abstraction Layer (SAL)	—	Device Abstraction API	Network Information Base (NIB) Graph with Import/Export Functions	—
Southbound APIs or connectors	OpenFlow, OVSDB, SNMP, PCEP, BGP, NETCONF	—	OpenFlow, L3 Agent, L2 Agent	OpenFlow, OVSDB	OpenFlow

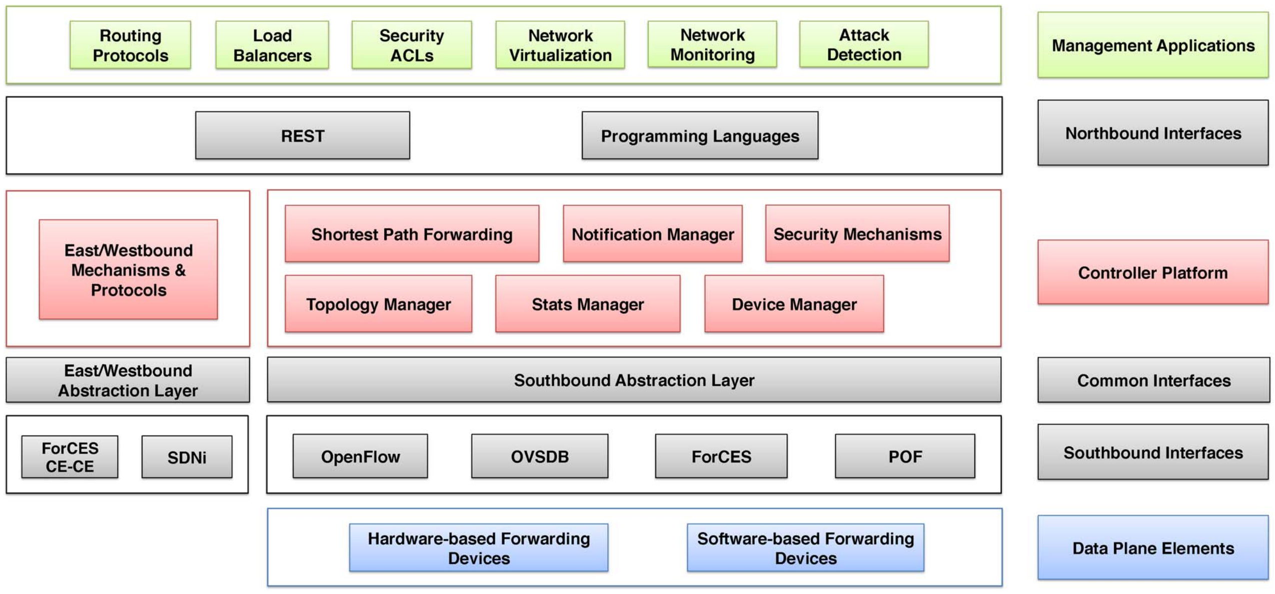

Fig. 8. ***SDN control platforms: elements, services, and interfaces.***

control platforms. We have focused on the elements, services, and interfaces of a selection of production-level, well-documented controllers and control platforms. Each line in the table represents a component we consider important in a modular and scalable control platform. We observe a highly diversified environment, with different properties and components being used by distinct control platforms. This is not surprising, given an environment with many competitors willing to be at the forefront of SDN development. Note also that not all components are available on all platforms. For instance, east/westbound APIs are not required in centralized controllers such as Beacon. In fact, some platforms have very specific niche markets, such as telecom companies and cloud providers, so the requirements will be different.

Based on the analysis of the different SDN controllers proposed to date (both those presented in Table 5 and others, such as NOX [26], Meridian [192], ForCES [30], and FortNOX [201]), we extract several common elements and provide a first attempt to clearly and systematically dissect an SDN control platform in Fig. 8.

There are at least three relatively well-defined layers in most of the existing control platforms: 1) the application, orchestration, and services; 2) the core controller functions; and 3) the elements for southbound communications. The connection at the upper level layers is based on northbound interfaces such as REST APIs [202] and programming languages such as FML [203], Frenetic [204], and NetCore [205]. On the lower level part of a control platform, southbound APIs and protocol plug-ins interface the forwarding elements. The core of a controller platform can be characterized as a combination of its base network service functions and the various interfaces.

4) Core Controller Functions: The base network service functions are what we consider the essential functionality all controllers should provide. As an analogy, these functions are like base services of operating systems, such as program execution, input/output (I/O) operations control, communications, protection, and so on. These services are used by other operating system level services and user applications. In a similar way, functions such as topology, statistics, notifications, and device management, together with shortest path forwarding and security mechanisms, are essential network control functionalities that network applications may use in building its logic. For instance, the notification manager should be able to receive, process, and forward events (e.g., alarm notifications, security alarms, state changes) [55]. Security mechanisms are another example, as they are critical components to provide basic isolation and security enforcement between services and applications. For instance, rules generated by high priority services should not be overwritten with rules created by applications with a lower priority.

5) Southbound: On the lower level of control platforms, the southbound APIs can be seen as a layer of device drivers. They provide a common interface for the upper layers, while allowing a control platform to use different southbound APIs (e.g., OpenFlow, OVSDB, and ForCES) and protocol plug-ins to manage existing or new physical or virtual devices (e.g., SNMP, BGP, and NetConf). This is essential both for backward compatibility and heterogeneity, i.e., to allow multiple protocols and device management connectors. Therefore, on the data plane, a mix of physical devices, virtual devices (e.g., Open vSwitch [109], [142], vRouter [206]) and a variety of device

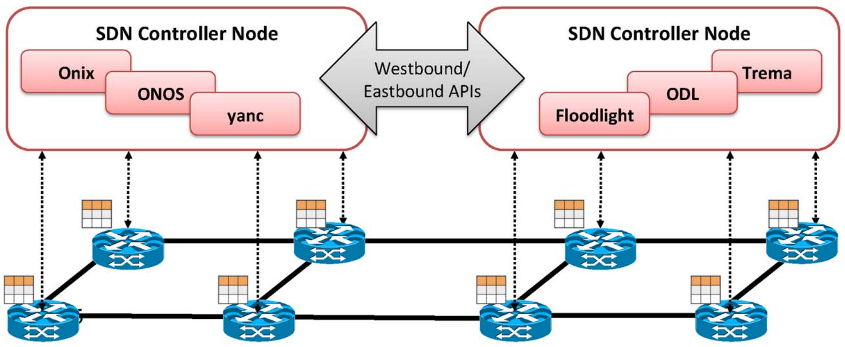

Fig. 9. *Distributed controllers: east/westbound APIs.*

interfaces (e.g., OpenFlow, OVSDB, of-config [207], NetConf, and SNMP) can coexist.

Most controllers support only OpenFlow as a southbound API. Still, a few of them, such as OpenDaylight, Onix, and HP VAN SDN Controller, offer a wider range of southbound APIs and/or protocol plug-ins. Onix supports both the OpenFlow and OVSDB protocols. The HP VAN SDN Controller has other southbound connectors such as L2 and L3 agents.

OpenDaylight goes a step beyond by providing a service layer abstraction (SLA) that allows several southbound APIs and protocols to coexist in the control platform. For instance, its original architecture was designed to support at least seven different protocols and plug-ins: OpenFlow, OVSDB [153], NETCONF [44], PCEP [43], SNMP [208], BGP [42], and LISP Flow Mapping [13]. Hence, OpenDaylight is one of the few control platforms being conceived to support a broader integration of technologies in a single control platform.

6) Eastbound and Westbound: East/westbound APIs, as illustrated in Fig. 9, are a special case of interfaces required by distributed controllers. Currently, each controller implements its own east/westbound API. The functions of these interfaces include import/export data between controllers, algorithms for data consistency models, and monitoring/notification capabilities (e.g., check if a controller is up or notify a take over on a set of forwarding devices).

Similarly to southbound and northbound interfaces, east/westbound APIs are essential components of distributed controllers. To identify and provide common compatibility and interoperability between different controllers, it is necessary to have standard east/westbound interfaces. For instance, SDNi [209] defines common requirements to coordinate flow setup and exchange reachability information across multiple domains. In essence, such protocols can be used in an orchestrated and interoperable way to create more scalable and dependable distributed control platforms. Interoperability can be leveraged to increase the diversity of the control platform element. Indeed, diversity increases the system robustness by reducing the probability of common faults, such as software faults [210].

Other proposals that try to define interfaces between controllers include Onix data import/export functions [7], ForCES CE–CE interface [30], [211], ForCES Intra-NE cold-standby mechanisms for high availability [212], and distributed data stores [213]. An east/westbound API requires advanced data distribution mechanisms such as the advanced message queuing protocol (AMQP) [214] used by DISCO [185], techniques for distributed concurrent and consistent policy composition [215], transactional databases and DHTs [216] (as used in Onix [7]), or advanced algorithms for strong consistency and fault tolerance [198], [213].

In a multidomain setup, east/westbound APIs may require also more specific communication protocols between SDN domain controllers [217]. Some of the essential functions of such protocols are to coordinate flow setup originated by applications, exchange reachability information to facilitate inter-SDN routing, reachability update to keep the network state consistent, among others.

Another important issue regarding east/westbound interfaces is heterogeneity. For instance, besides communicating with peer SDN controllers, controllers may also need to communicate with subordinate controllers (in a hierarchy of controllers) and non-SDN controllers [218], as is the case of Closed-Flow [219]. To be interoperable, east/westbound interfaces thus need to accommodate different controller interfaces, with their specific set of services, and the diverse characteristics of the underlying infrastructure, including the diversity of technology, the geographic span and scale of the network, and the distinction between WAN and LAN—potentially across administrative boundaries. In those cases, different information has to be exchanged between controllers, including adjacency and capability discovery, topology information (to the extent of the agreed contracts between administrative domains), billing information, among many others [218].

Last, an "SDN compass" methodology [220] suggests a finer distinction between eastbound and westbound horizontal interfaces, referring to westbound interfaces

Table 6 Controllers Classification

Name	Architecture	Northbound API	Consistency	Faults	License	Prog. language	Version
Beacon [186]	centralized multi-threaded	ad-hoc API	no	no	GPLv2	Java	v1.0
DISCO [185]	distributed	REST	—	yes	—	Java	v1.1
ElastiCon [228]	distributed	RESTful API	yes	no	—	Java	v1.0
Fleet [199]	distributed	ad-hoc	no	no	—	—	v1.0
Floodlight [189]	centralized multi-threaded	RESTful API	no	no	Apache	Java	v1.1
HP VAN SDN [184]	distributed	RESTful API	weak	yes	—	Java	v1.0
HyperFlow [195]	distributed	—	weak	yes	—	C++	v1.0
Kandoo [229]	hierarchically distributed	—	no	no	—	C, C++, Python	v1.0
Onix [7]	distributed	NVP NBAPI	weak, strong	yes	commercial	Python, C	v1.0
Maestro [188]	centralized multi-threaded	ad-hoc API	no	no	LGPLv2.1	Java	v1.0
Meridian [192]	centralized multi-threaded	extensible API layer	no	no	—	Java	v1.0
MobileFlow [222]	—	SDMN API	—	—	—	—	v1.2
MuL [230]	centralized multi-threaded	multi-level interface	no	no	GPLv2	C	v1.0
NOX [26]	centralized	ad-hoc API	no	no	GPLv3	C++	v1.0
NOX-MT [187]	centralized multi-threaded	ad-hoc API	no	no	GPLv3	C++	v1.0
NVP Controller [112]	distributed	—	—	—	commercial	—	—
OpenContrail [183]	—	REST API	no	no	Apache 2.0	Python, C++, Java	v1.0
OpenDaylight [13]	distributed	REST, RESTCONF	weak	no	EPL v1.0	Java	v1.{0,3}
ONOS [117]	distributed	RESTful API	weak, strong	yes	—	Java	v1.0
PANE [197]	distributed	PANE API	yes	—	—	—	—
POX [231]	centralized	ad-hoc API	no	no	GPLv3	Python	v1.0
ProgrammableFlow [232]	centralized	—	—	—	—	C	v1.3
Rosemary [194]	centralized	ad-hoc	—	—	—	—	v1.0
Ryu NOS [191]	centralized multi-threaded	ad-hoc API	no	no	Apache 2.0	Python	v1.{0,2,3}
SMaRtLight [198]	distributed	RESTful API	yes	yes	—	Java	v1.0
SNAC [233]	centralized	ad-hoc API	no	no	GPL	C++	v1.0
Trema [190]	centralized multi-threaded	ad-hoc API	no	no	GPLv2	C, Ruby	v1.0
Unified Controller [171]	—	REST API	—	—	commercial	—	v1.0
yanc [196]	distributed	file system	—	—	—	—	—

as SDN-to-SDN protocols and controller APIs while eastbound interfaces would be used to refer to standard protocols used to communicate with legacy network control planes (e.g., PCEP [43] and GMPLS [221]).

7) Northbound: Current controllers offer a quite broad variety of northbound APIs, such as *ad hoc* APIs, RESTful APIs [202], multilevel programming interfaces, file systems, among other more specialized APIs such as NVP NBAPI [7], [112] and SDMN API [222]. Section IV-E is devoted to a more detailed discussion on the evolving layer of northbound APIs. A second kind of northbound interfaces are those stemming out of SDN programming languages such as Frenetic [204], Nettle [223], NetCore [205], Procera [224], Pyretic [225], NetKAT [226], and other query-based languages [227]. Section IV-G gives a more detailed overview of the several existing programming languages for SDN.

8) Wrapping Up Remarks and Platforms Comparison: Table 6 shows a summary of some of the existing controllers with their respective architectures and characteristics. As can be observed, most controllers are centralized and multithreaded. Curiously, the northbound API is very diverse. In particular, five controllers (Onix, Floodlight, MuL, Meridian, and SDN Unified Controller) pay a bit more attention to this interface, as a statement of its importance. Consistency models and fault tolerance are only present in Onix, HyperFlow, HP VAN SDN, ONOS, and SMaRtLight. Last, when it comes to the OpenFlow standard as southbound API, only Ryu supports its three major versions (v1.0, v1.2, and v1.3).

To conclude, it is important to emphasize that the control platform is one of the critical points for the success of SDN [234]. One of the main issues that needs to be addressed in this respect is interoperability. This is rather interesting, as it was the very first problem that southbound APIs (such as OpenFlow) tried to solve. For instance, while WiFi and long-term evolution (LTE) networks [235] need specialized control platforms such as MobileFlow [222] or SoftRAN [236], data center networks have different requirements that can be met with platforms such as Onix [7] or OpenDaylight [13]. For this reason, in environments where diversity of networking infrastructures is a reality, coordination and cooperation between different controllers is crucial. Standardized APIs for multicontroller and multidomain deployments are therefore seen as an important step to achieve this goal.

E. Layer V: Northbound Interfaces

The northbound and southbound interfaces are two key abstractions of the SDN ecosystem. The southbound interface has already a widely accepted proposal (OpenFlow), but a common northbound interface is still an open issue. At this moment it may still be a bit too early to define a standard northbound interface, as use cases are still being worked out [237]. Anyway, it is to be expected a common (or a *de facto*) northbound interface to arise as SDN evolves. An abstraction that would allow network applications not to depend on specific implementations is important to explore the full potential of SDN.

The northbound interface is mostly a software ecosystem, not a hardware one, as is the case of the southbound APIs.

In these ecosystems, the implementation is commonly the forefront driver, while standards emerge later and are essentially driven by wide adoption [238]. Nevertheless, an initial and minimal standard for northbound interfaces can still play an important role for the future of SDN. Discussions about this issue have already begun [237]–[244], and a common consensus is that northbound APIs are indeed important but that it is indeed too early to define a single standard right now. The experience from the development of different controllers will certainly be the basis for coming up with a common application level interface.

Open and standard northbound interfaces are crucial to promote application portability and interoperability among the different control platforms. A northbound API can be compared to the POSIX standard [245] in operating systems, representing an abstraction that guarantees programming language and controller independence. NOSIX [246] is one of the first examples of an effort in this direction. It tries to define portable low-level (e.g., flow model) application interfaces, making southbound APIs such as OpenFlow look like "device drivers." However, NOSIX is not exactly a general purpose northbound interface, but rather a higher level abstraction for southbound interfaces. Indeed, it could be part of the common abstraction layer in a control platform as the one described in Section IV-D.

Existing controllers such as Floodlight, Trema, NOX, Onix, and OpenDaylight propose and define their own northbound APIs [239], [247]. However, each of them has its own specific definitions. Programming languages such as Frenetic [204], Nettle [223], NetCore [205], Procera [224], Pyretic [248], and NetKAT [226] also abstract the inner details of the controller functions and data plane behavior from the application developers. Moreover, as we explain in Section IV-G, programming languages can provide a wide range of powerful abstractions and mechanisms such as application composition, transparent data plane fault tolerance, and a variety of basic building blocks to ease software module and application development.

SFNet [249] is another example of a northbound interface. It is a high-level API that translates application requirements into lower level service requests. However, SFNet has a limited scope, targeting queries to request the congestion state of the network and services such as bandwidth reservation and multicast.

Other proposals use different approaches to allow applications to interact with controllers. The *yanc* control platform [196] explores this idea by proposing a general control platform based on Linux and abstractions such as the virtual file system (VFS). This approach simplifies the development of SDN applications as programmers are able to use a traditional concept (files) to communicate with lower level devices and subsystems.

Eventually, it is unlikely that a single northbound interface emerges as the winner, as the requirements for different network applications are quite different. APIs for security applications are likely to be different from those for routing or financial applications. One possible path of evolution for northbound APIs are vertically oriented proposals, before any type of standardization occurs, a challenge the ONF has started to undertake in the NBI WG in parallel to open-source SDN developments [50]. The ONF architectural work [218] includes the possibility of northbound APIs providing resources to enable dynamic and granular control of the network resources from customer applications, eventually across different business and organizational boundaries.

There are also other kinds of APIs, such as those provided by the PANE controller [197]. Designed to be suitable for the concept of participatory networking, PANE allows network administrators to define module-specific quotas and access control policies on network resources. The controller provides an API that allows end-host applications to dynamically and autonomously request network resources. For example, audio (e.g., VoIP) and video applications can easily be modified to use the PANE API to reserve bandwidth for certain quality guarantees during the communication session. PANE includes a compiler and verification engine to ensure that bandwidth requests do not exceed the limits set by the administrator and to avoid starvation, i.e., other applications will not be impaired by new resource requests.

F. Layer VI: Language-Based Virtualization

Two essential characteristics of virtualization solutions are the capability of expressing modularity and of allowing different levels of abstractions while still guaranteeing desired properties such as protection. For instance, virtualization techniques can allow different views of a single physical infrastructure. As an example, one virtual "big switch" could represent a combination of several underlying forwarding devices. This intrinsically simplifies the task of application developers as they do not need to think about the sequence of switches where forwarding rules have to be installed, but rather see the network as a simple "big switch." Such kind of abstraction significantly simplifies the development and deployment of complex network applications, such as advanced security related services.

Pyretic [248] is an interesting example of a programming language that offers this type of high-level abstraction of network topology. It incorporates this concept of abstraction by introducing network objects. These objects consist of an abstract network topology and the sets of policies applied to it. Network objects simultaneously hide information and offer the required services.

Another form of language-based virtualization is static slicing. This a scheme where the network is sliced by a compiler, based on application layer definitions. The output of the compiler is a monolithic control program that has already slicing definitions and configuration commands for the network. In such a case, there is no need for a hypervisor to dynamically manage the network slices. Static slicing can be valuable for deployments with specific requirements, in particular those where higher performance and simple isolation guarantees are preferable to dynamic slicing.

One example of static slicing approach is the Splendid isolation [250]. In this solution, the network slices are made of three components: 1) topology, consisting of switches, ports, and links; 2) mapping of slice-level switches, ports, and links on the network infrastructure; and 3) predicates on packets, where each port of the slice's edge switches has an associated predicate. The topology is a simple graph of the sliced nodes, ports, and links. Mapping will translate the abstract topology elements into the corresponding physical ones. The predicates are used to indicate whether a packet is permitted to enter a specific slice. Different applications can be associated to each slice. The compiler takes the combination of slices (topology, mapping, and predicates) and respective programs to generate a global configuration for the entire network. It also ensures that properties such as isolation are enforced among slices, i.e., no packets of slice A can traverse to slice B unless explicitly allowed.

Other solutions, such as libNetVirt [251], try to integrate heterogeneous technologies for creating static network slices. libNetVirt is a library designed to provide a flexible way to create and manage virtual networks in different computing environments. Its main idea is similar to the OpenStack Quantum project [252]. While Quantum is designed for OpenStack (cloud environments), libNetVirt is a more general purpose library which can be used in different environments. Additionally, it goes one step beyond OpenStack Quantum by enabling QoS capabilities in virtual networks [251]. The libNetVirt library has two layers: a generic network interface and technology specific device drivers (e.g., VPN, MPLS, OpenFlow). On top of the layers are the network applications and virtual network descriptions. The OpenFlow driver uses a NOX controller to manage the underlying infrastructure, using OpenFlow rule-based flow tables to create isolated virtual networks. By supporting different technologies, it can be used as a bridging component in heterogeneous networks.

Table 7 summarizes the hypervisor- and nonhypervisor-based virtualization technologies. As can be observed, only libNetVirt supports heterogeneous technologies, not restricting its application to OpenFlow-enabled networks. FlowVisor, AutoSlice, and OpenVirteX allow multiple controllers, one per network slice. FlowN provides a container-based approach where multiple applications from different users can coexist on a single controller. FlowVisor allows QoS provisioning guarantees by using VLAN PCP bits for priority queues. SDN VE and NVP also provide their own provisioning methods for guaranteeing QoS.

G. Layer VII: Programming Languages

Programming languages have been proliferating for decades. Both academia and industry have evolved from low-level hardware-specific machine languages, such as

Table 7 Virtualization Solutions

Solution	Multiple controllers	Slicing	QoS "guarantees"	Multi-technology
AutoVFlow [174]	yes, one per tenant	flow space virtualization	no	no, OF only
AutoSlice [179]	yes, one per slice	VLAN tags	no	no, OF only
Compositional Hypervisor [181]	—	—	no	no, OF only
FlowVisor [166], [167]	yes, one per slice	virtual flow tables per slice	yes (VLAN PCP bits)	no, OF only
FlowN [168], [180]	no (contained applications)	VLAN tags	no	no, OF only
IBM SDN VE [171]	yes, a cluster of controllers	logical datapaths	yes (priority-based)	yes (VXLAN, OVS, OF)
libNetVirt [251]	no, one single controller	VLAN tags	no	yes (e.g., VPN, MPLS, OF)
NVP's Hypervisor [112]	yes, a cluster of controller	logical datapaths	yes	no, OVS only
OpenVirteX [170]	yes, one per slice	virtual flow tables per slice	*unknown*	no, OF only
Pyretic [248]	no, one single controller	compiler time OF rules	no	no, OF only
Splendid Isolation [250]	no, one single controller	compiler time VLANs	no	no, OF only
RadioVisor [173]	yes, one per slice	3D resource grid slices	—	Femto API [253]
xDPd virtualization [176]	yes, one per slice	flow space virtualization	no	no, OF only

assembly for x86 architectures, to high-level and powerful programming languages such as Java and Python. The advancements toward more portable and reusable code have driven a significant shift on the computer industry [254], [255].

Similarly, programmability in networks is starting to move from low-level machine languages such as OpenFlow ("assembly") to high-level programming languages [112], [203]–[205], [223]–[225]. Assembly-like machine languages, such as OpenFlow [9] and POF [31], [120], essentially mimic the behavior of forwarding devices, forcing developers to spend too much time on low-level details rather than on the problem solve. Raw OpenFlow programs have to deal with hardware behavior details such as overlapping rules, the priority ordering of rules, and data plane inconsistencies that arise from in-flight packets whose flow rules are under installation [204], [205], [256]. The use of these low-level languages makes it difficult to reuse software, to create modular and extensive code, and leads to a more error-prone development process [225], [257], [258].

Abstractions provided by high-level programming languages can significantly help address many of the challenges of these lower level instruction sets [203]–[205], [223]–[225]. In SDNs, high-level programming languages can be designed and used to:

1) create higher level abstractions for simplifying the task of programming forwarding devices;
2) enable more productive and problem-focused environments for network software programmers, speeding up development and innovation;
3) promote software modularization and code reusability in the network control plane;
4) foster the development of network virtualization.

Several challenges can be better addressed by programming languages in SDNs. For instance, in pure OpenFlow-based SDNs, it is hard to ensure that multiple tasks of a single application (e.g., routing, monitoring, access control) do not interfere with each other. For example, rules generated for one task should not override the functionality of another task [204], [256]. Another example is when multiple applications run on a single controller [201], [225], [256], [259], [260]. Typically, each application generates rules based on its own needs and policies without further knowledge about the rules generated by other applications. As a consequence, conflicting rules can be generated and installed in forwarding devices, which can create problems for network operation. Programming languages and runtime systems can help to solve these problems that would be otherwise hard to prevent.

Important software design techniques such as code modularity and reusability are very hard to achieve using low-level programming models [225]. Applications thus built are monolithic and consist of building blocks that cannot be reused in other applications. The end result is a very time-consuming and error-prone development process.

Another interesting feature that programming language abstractions provide is the capability of creating and writing programs for virtual network topologies [248], [250]. This concept is similar to object-oriented programming, where objects abstract both data and specific functions for application developers, making it easier to focus on solving a particular problem without worrying about data structures and their management. For instance, in an SDN context, instead of generating and installing rules in each forwarding device, one can think of creating simplified virtual network topologies that represent the entire network, or a subset of it. For example, the application developer should be able to abstract the network as an atomic big switch, rather than a combination of several underlying physical devices. The programming languages or runtime systems should be responsible for generating and installing the lower level instructions required at each forwarding device to enforce the user policy across the network. With such kind of abstractions, developing a routing application becomes a straightforward process. Similarly, a single physical switch could be represented as a set of virtual switches, each of them belonging to a different virtual network. These two examples of abstract network topologies would be much harder to implement with low-level instruction sets. In contrast, a programming language or runtime system can more easily provide abstractions for virtual network topologies, as has already been demonstrated by languages such as Pyretic [248].

1) High-Level SDN Programming Languages: High-level programming languages can be powerful tools as a mean for implementing and providing abstractions for different important properties and functions of SDN such as network-wide structures, distributed updates, modular composition, virtualization, and formal verification [29].

Low-level instruction sets suffer from several problems. To address some of these challenges, higher level programming languages have been proposed, with diverse goals, such as:

- avoiding low-level and device-specific configurations and dependencies spread across the network, as happens in traditional network configuration approaches;
- providing abstractions that allow different management tasks to be accomplished through easy to understand and maintain network policies;
- decoupling of multiple tasks (e.g., routing, access control, traffic engineering);
- implementing higher level programming interfaces to avoid low-level instruction sets;
- solving forwarding rules problems, e.g., conflicting or incomplete rules that can prevent a switch event to be triggered, in an automated way;
- addressing different race condition issues which are inherent to distributed systems;

Table 8 Programming Languages

Name	Programming paradigm	Short description/purpose
FatTire [262]	declarative (functional)	Uses regular expressions to allow programmers to describe network paths and respective fault-tolerance requirements.
Flog [258]	declarative (logic), event-driven	Combines ideas of FML and Frenetic, providing an event-driven and forward-chaining logic programming language.
FlowLog [257]	declarative (functional)	Provides a finite-state language to allow different analysis, such as model-checking.
FML [203]	declarative (dataflow, reactive)	High level policy description language (e.g., access control).
Frenetic [204]	declarative (functional)	Language designed to avoid race conditions through well defined high level programming abstractions.
HFT [256]	declarative (logic, functional)	Enables hierarchical policies description with conflict-resolution operators, well suited for decentralized decision makers.
Maple [263]	declarative (functional)	Provides a highly-efficient multi-core scheduler that can scale efficiently to controllers with 40+ cores.
Merlin [264]	declarative (logic)	Provides mechanisms for delegating management of sub-policies to tenants without violating global constraints.
nlog [112]	declarative (functional)	Provides mechanisms for data log queries over a number of tables. Produces immutable tuples for reliable detection and propagation of updates.
NetCore [205]	declarative (functional)	High level programming language that provides means for expressing packet-forwarding policies in a high level.
NetKAT [226]	declarative (functional)	It is based on Kleene algebra for reasoning about network structure and supported by solid foundation on equational theory.
Nettle [223]	declarative (functional, reactive)	Based on functional reactive programming principles in order to allow programmers to deal with streams instead of events.
Procera [224]	declarative (functional, reactive)	Incorporates a set of high level abstractions to make it easier to describe reactive and temporal behaviors.
Pyretic [225]	imperative	Specifies network policies at a high level of abstraction, offering transparent composition and topology mapping.

- enhancing conflict-resolution techniques on environments with distributed decision makers;
- providing native fault-tolerance capabilities on data plane path setup;
- reducing the latency in the processing of new flows;
- easing the creation of stateful applications (e.g., stateful firewall).

Programming languages can also provide specialized abstractions to cope with other management requirements, such as monitoring [204], [224], [227], [261]. For instance, the runtime system of a programming language can do all the "laundry work" of installing rules, polling the counters, receiving the responses, combining the results as needed, and composing monitoring queries in conjunction with other policies. Consequently, application developers can take advantage of the simplicity and power of higher level query instructions to easily implement monitoring modules or applications.

Another aspect of paramount importance is the portability of the programming language, necessary so that developers do not need to re-implement applications for different control platforms. The portability of a programming language can be considered as a significant added value to the control plane ecosystem. Mechanisms such as decoupled back–ends could be key architectural ingredients to enable platform portability. Similarly to the Java virtual machine, a portable northbound interface will easily allow applications to run on different controllers without requiring any modification. As an example, the Pyretic language requires only a standard socket interface and a simple OpenFlow client on the target controller platform [225].

Several programming languages have been proposed for SDNs, as summarized in Table 8. The great majority propose abstractions for OpenFlow-enabled networks. The predominant programming paradigm is the declarative one, with a single exception, Pyretic, which is an imperative language. Most declarative languages are functional, but there are instances of the logic and reactive types. The purpose—i.e., the specific problems they intend to solve—and the expressiveness power vary from language to language, while the end goal is almost always the same: to provide higher level abstractions to facilitate the development of network control logic.

Programming languages such as FML [203], Nettle [223], and Procera [224] are functional and reactive. Policies and applications written in these languages are based on reactive actions triggered by events (e.g., a new host connected to the network, or the current network load). Such languages allow users to declaratively express different network configuration rules such as access control lists (ACLs), virtual LANs (VLANs), and many others. Rules are essentially expressed as allow-or-deny policies, which are applied to the forwarding elements to ensure the desired network behavior.

Other SDN programming languages, such as Frenetic [204], hierarchical flow tables (HFTs) [256], NetCore [205], and Pyretic [225], were designed with the simultaneous goal of efficiently expressing packet-forwarding policies and dealing with overlapping rules of different applications, offering advanced operators for parallel and sequential composition of software modules. To avoid overlapping conflicts, Frenetic disambiguates rules with overlapping patterns by assigning different integer priorities, while HFT uses hierarchical policies with enhanced conflict-resolution operators.

See-every-packet abstractions and race-free semantics also represent interesting features provided by programming languages (such as Frenetic [204]). The former ensures that all control packets will be available for analysis, sooner or later, while the latter provides the mechanisms for suppressing unimportant packets. As an example, packets that arise from a network race condition, such as due to a concurrent flow rule installation on switches, can be simply discarded by the runtime system.

Advanced operators for parallel and sequential composition help bind (through internal workflow operators) the key characteristics of programming languages such as Pyretic [225]. Parallel composition makes it possible to operate multiple policies on the same set of packets, while sequential composition facilitates the definition of a sequential workflow of policies to be processed on a set of packets. Sequential policy processing allows multiple modules (e.g., access control and routing) to operate in a cooperative way. By using sequential composition complex applications can be built out of a combination of different modules (in a similar way as pipes can be used to build sophisticated Unix applications).

Further advanced features are provided by other SDN programming languages. FatTire [262] is an example of a declarative language that heavily relies on regular expressions to allow programmers to describe network paths with fault-tolerance requirements. For instance, each flow can have its own alternative paths for dealing with failure of the primary paths. Interestingly, this feature is provided in a very programmer-friendly way, with the application programmer having only to use regular expressions with special characters, such as an asterisk. In the particular case of FatTire, an asterisk will produce the same behavior as a traditional regular expression, but translated into alternative traversing paths.

Programming languages such as FlowLog [257] and Flog [258] bring different features, such as model checking, dynamic verification, and stateful middleboxes. For instance, using a programming language such as Flog, it is possible to build a stateful firewall application with only five lines of code [258].

Merlin [264] is one of the first examples of unified framework for controlling different network components, such as forwarding devices, middleboxes, and end-hosts. An important advantage is backward compatibility with existing systems. To achieve this goal, Merlin generates specific code for each type of component. Taking a policy definition as input, Merlin's compiler determines forwarding paths, transformation placement, and bandwidth allocation. The compiled outputs are sets of component-specific low-level instructions to be installed in the devices. Merlin's policy language also allows operators to delegate the control of a subnetwork to tenants, while ensuring isolation. This delegated control is expressed by means of policies that can be further refined by each tenant owner, allowing them to customize policies for their particular needs.

Other recent initiatives (e.g., systems programming languages [265]) target problems such as detecting anomalies to improve the security of network protocols (e.g., Open-Flow), and optimizing horizontal scalability for achieving high throughput in applications running on multicore architectures [263]. Nevertheless, there is still scope for further investigation and development on programming languages. For instance, one recent research has revealed that current policy compilers generate unnecessary (redundant) rule updates, most of which modify only the priority field [266].

Most of the value of SDN will come from the network managements applications built on top of the infrastructure. Advances in high-level programming languages are a fundamental component to the success of a prolific SDN application development ecosystem. To this end, efforts are undergoing to shape forthcoming standard interfaces (cf. [267]) and toward the realization of integrated development environments (e.g., NetIDE [268]) with the goal of fostering the development of a myriad of SDN applications. We discuss these next.

H. Layer VIII: Network Applications

Network applications can be seen as the "network brains." They implement the control logic that will be translated into commands to be installed in the data plane, dictating the behavior of the forwarding devices. Take a simple application as routing as an example. The logic of this application is to define the path through which packets will flow from point A to point B. To achieve this goal, a routing application has to, based on the topology input, decide on the path to use and instruct the controller to install the respective forwarding rules in all forwarding devices on the chosen path, from A to B.

SDNs can be deployed on any traditional network environment, from home and enterprise networks to data centers and Internet exchange points. Such variety of environments has led to a wide array of network applications. Existing network applications perform traditional functionality such as routing, load balancing, and security policy enforcement, but also explore novel approaches, such as reducing power consumption. Other examples include fail-over and reliability functionalities to the data plane, end-to-end QoS enforcement, network virtualization, mobility management in wireless networks, among many others. The variety of network applications, combined with real use case deployments, is expected to be one of the major forces on fostering a broad adoption of SDN [269].

Despite the wide variety of use cases, most SDN applications can be grouped in one of five categories: traffic engineering, mobility and wireless, measurement and monitoring, security and dependability and data center networking. Tables 9 and 10 summarize several applications categorized as such, stating their main purpose, controller where it was implemented/evaluated, and southbound API used.

1) Traffic Engineering: Several traffic engineering applications have been proposed, including ElasticTree [274], Hedera [276], OpenFlow-based server load balancing [338], Plug-n-Serve [285] and Aster*x [273], In-packet Bloom filter [277], SIM–PLE [292], QNOX [287], QoS framework [289], QoS for SDN [288], ALTO [270], ViAggre SDN [293], ProCel [282], FlowQoS [275], and Middlepipes [27]. In addition to these, recent proposals include optimization of rules placement [339], the use of MAC as a universal label for efficient routing in data centers [340], among other techniques for flow management, fault tolerance, topology update, and traffic characterization [341]. The main goal of most applications is to engineer traffic with the aim of minimizing power consumption, maximizing aggregate network utilization, providing optimized load balancing, and other generic traffic optimization techniques.

Load balancing was one of the first applications envisioned for SDN/OpenFlow. Different algorithms and techniques have been proposed for this purpose [338], [273], [285]. One particular concern is the scalability of these solutions. A technique to allow this type of applications to scale is to use wildcard-based rules to perform proactive load balancing [338]. Wildcards can be utilized for aggregating client requests based on the ranges of IP prefixes, for instance, allowing the distribution and directing of large groups of client requests without requiring controller intervention for every new flow. In tandem, operation in reactive mode may still be used when traffic bursts are detected. The controller application needs to monitor the network traffic and use some sort of threshold in the flow counters to redistribute clients among the servers when bottlenecks are likely to happen.

SDN load balancing also simplifies the placement of network services in the network [285]. Every time a new server is installed, the load balancing service can take the appropriate actions to seamlessly distribute the traffic among the available servers, taking into consideration both the network load and the available computing capacity of the respective servers. This simplifies network management and provides more flexibility to network operators.

Existing southbound interfaces can be used for actively monitoring the data plane load. This information can be leveraged to optimize the energy consumption of the network [274]. By using specialized optimization algorithms and diversified configuration options, it is possible to meet the infrastructure goals of latency, performance, and fault tolerance, for instance, while reducing power consumption. With the use of simple techniques, such as shutting down links and devices intelligently in response to traffic load dynamics, data center operators can save up to 50% of the network energy in normal traffic conditions [274].

One of the important goals of data center networks is to avoid or mitigate the effect of network bottlenecks on the operation of the computing services offered. Linear bisection bandwidth is a technique that can be adopted for traffic patterns that stress the network by exploring path diversity in a data center topology. Such technique has been proposed in an SDN setting, allowing the maximization of aggregated network utilization with minimal scheduling overhead [276]. SDN can also be used to provide a fully automated system for controlling the configuration of routers. This can be particularly useful in scenarios that apply virtual aggregation [342]. This technique allows network operators to reduce the data replicated on routing tables, which is one of the causes of routing tables' growth [343]. A specialized routing application [293] can calculate, divide, and configure the routing tables of the different routing devices through a southbound API such as OpenFlow.

Traffic optimization is another interesting application for large-scale service providers, where dynamic scale-out is required. For instance, the dynamic and scalable provisioning of VPNs in cloud infrastructures, using protocolols such as ALTO [272], can be simplified through an SDN-based approach [270]. Recent work has also shown that optimizing rules placement can increase network efficiency [339]. Solutions such as ProCel [282], designed for cellular core networks, are capable of reducing the signaling traffic up to 70%, which represents a significant achievement.

Other applications that perform routing and traffic engineering include application-aware networking for video and data streaming [344], [345] and improved QoS by employing multiple packet schedulers [290] and other techniques [279], [287], [289], [346]. As traffic engineering is a crucial issue in all kinds of networks, upcoming methods, techniques, and innovations can be expected in the context of SDNs.

2) Mobility and Wireless: The current distributed control plane of wireless networks is suboptimal for managing the limited spectrum, allocating radio resources, implementing handover mechanisms, managing interference, and performing efficient load balancing between cells. SDN-based approaches represent an opportunity for making it easier to deploy and manage different types of wireless networks, such as WLANs and cellular networks [236], [296], [300], [302], [347], [348]. Traditionally hard-to-implement but desired features are indeed becoming a

Table 9 Network Applications

Group	Solution/Application	Main purpose	Controller	Southbound API
Traffic engineering	ALTO VPN [270]	on-demand VPNs	NMS [271], [272]	SNMP
	Aster*x [273]	load balancing	NOX	OpenFlow
	ElasticTree [274]	energy aware routing	NOX	OpenFlow
	FlowQoS [275]	QoS for broadband access networks	POX	OpenFlow
	Hedera [276]	scheduling / optimization	—	OpenFlow
	In-packet Bloom filter [277]	load balancing	NOX	OpenFlow
	MicroTE [278]	traffic engineering with minimal overhead	NOX	OpenFlow
	Middlepipes [27]	Middleboxes as a PaaS	middlepipe controller	—
	OpenQoS [279]	dynamic QoS routing for multimedia apps	Floodlight	OpenFlow
	OSP [280]	fast recovery through fast-failover groups	NOX	OpenFlow
	PolicyCop [281]	QoS policy management framework	Floodlight	OpenFlow
	ProCel [282]	Efficient traffic handling for software EPC	ProCel controller	—
	Pronto [283], [284]	Efficient queries on distributed data stores	Beacon	OpenFlow
	Plug-n-Serve [285]	load balancing	NOX	OpenFlow
	Pythia [286]	traffic engineering for big data apps	OpenDaylight	OpenFlow
	QNOX [287]	QoS enforcement	NOX	Generalized OpenFlow
	QoS for SDN [288]	QoS over heterogeneous networks	Floodlight	OpenFlow
	QoS framework [289]	QoS enforcement	NOX	OF with QoS extensions
	QoSFlow [290]	multiple packet schedulers to improve QoS	—	OpenFlow
	QueuePusher [291]	Queue management for QoS enforcement	Floodlight	OpenFlow
	SIMPLE [292]	middlebox-specific “traffic steering”	Extended POX	OpenFlow
	ViAggre SDN [293]	divide and spread forwarding tables	NOX	OpenFlow
Mobility & Wireless	AeroFlux [294], [295]	scalable hierarchical WLAN control plane	Floodlight	OpenFlow, Odin
	CROWD [296]	overlapping of LTE and WLAN cells	—	OpenFlow
	CloudMAC [297]	outsourced processing of WLAN MACs	—	OpenFlow
	C-RAN [298]	RAN [236] virtualization for mobile nets	—	—
	FAMS [299]	flexible VLAN system based on OpenFlow	ProgrammableFlow	OpenFlow
	MobileFlow [222]	flow-based model for mobile networks	MobileFlow	SDMN API
	Odin [300]	programmable virtualized WLANs	Floodlight	OpenFlow, Odin
	OpenRAN [301]	vertical programmability and virtualization	—	—
	OpenRoads [302]	control of the data path using OpenFlow	FlowVisor	OpenFlow
	SoftRAN [236]	load balancing and interference management	—	Femto API [253], [303]
Measurement & Monitoring	BISmark [6]	active and passive measurements	Procera framework	OpenFlow
	DCM [304]	distributed and coactive traffic monitoring	DCM controller	OpenFlow
	FleXam [305]	flexible sampling extension for OpenFlow	—	—
	FlowSense [306]	measure link utilization in OF networks	—	OpenFlow
	measurement model [307]	model for OF switch measurement tasks	—	OpenFlow
	OpenNetMon [308]	monitoring of QoS parameters to improve TE	POX	OpenFlow
	OpenSample [309]	low-latency sampling-based measurements	Floodlight	modified sFlow [310]
	OpenSketch [311]	separated measurement data plane	OpenSketch	“OpenSketch sketches”
	OpenTM [261]	traffic matrix estimation tool	NOX	OpenFlow
	PaFloMon [312]	passive monitoring tools defined by users	FlowVisor	OpenFlow
	PayLess [313]	query-based real-time monitoring framework	Floodlight	OpenFlow
Data Center Networking	Big Data Apps [314]	optimize network utilization	—	OpenFlow
	CloudNaaS [315]	networking primitives for cloud applications	NOX	OpenFlow
	FlowComb [316]	predicts application workloads	NOX	OpenFlow
	FlowDiff [317]	detects operational problems	FlowVisor	OpenFlow
	LIME [318]	live network migration	Floodlight	OpenFlow
	NetGraph [319]	graph queries for network management	—	OpenFlow, SNMP
	OpenTCP [320]	dynamic and programmable TCP adaptation	—	—

Table 10 Network Applications

Group	Solution/Application	Main purpose	Controller	Southbound API
Security & Dependability	Active security [321]	integrated security using feedback control	Floodlight	OpenFlow
	AVANT-GUARD [322]	DoS security specific extensions to OF	POX	OpenFlow
	CloudWatcher [323]	framework for monitoring clouds	NOX	OpenFlow
	Cognition [324]	cognitive functions to enhanced security mechanisms in network applications	—	—
	DDoS detection [325]	attacks detection and mitigation	NOX	OpenFlow
	Elastic IP & Security [326]	SDN-based elastic IP and security groups	NOX	OpenFlow
	Ethane [101]	flow-rule enforcement (match/action)	Ethane controller	first instance of OpenFlow
	FlowNAC [327]	flow-based network access control	NOX	OpenFlow
	FortNOX [201]	security flow rules prioritization	NOX	OpenFlow
	FRESCO [259]	framework for security services composition	NOX	OpenFlow
	LiveSec [328]	security policy enforcement	NOX	OpenFlow
	MAPPER [329]	fine-grained access control	—	—
	NetFuse [330]	protection against OF traffic overload	—	OpenFlow
	OF-RHM [331]	random host mutation (defense)	NOX	OpenFlow
	OpenSAFE [332]	direct spanned net traffic in arbitrary ways	NOX	OpenFlow
	OrchSec [333]	architecture for developing security apps	*any*	Flow-RT [334], OpenFlow
	Reliable multicasting [335]	reduce packet loss when failures occur	Trema	OpenFlow
	SANE [100]	security policy enforcement	SANE controller	SANE header (pre-OF)
	SDN RTBH [336]	DoS attack mitigation	POX	OpenFlow
	VAVE [337]	source address validation with a global view	NOX	OpenFlow

reality with the SDN-based wireless networks. These include seamless mobility through efficient handovers [300], [347], [349], load balancing [236], [300], creation of on-demand virtual access points (VAPs) [297], [300], downlink scheduling (e.g., an OpenFlow switch can do a rate shaping or time division) [297], dynamic spectrum usage [297], enhanced intercell interference coordination [297], [347], device-to-device offloading (i.e., decide when and how LTE transmissions should be offloaded to users adopting the device to device paradigm [296]) [350], per client and/or base station resource block allocations (i.e., time and frequency slots in LTE/orthogonal frequency-division multiple access (OFDMA) networks, which are known as resource blocks) [236], [296], [348], control and assign transmission and power parameters in devices or in a group basis (e.g., algorithms to optimize the transmission and power parameters of WLAN devices define and assign transmission power values to each resource block, at each base station, in LTE/OFDMA networks) [236], [296], simplified administration [236], [300], [302], easy management of heterogeneous network technologies [236], [302], [351], interoperability between different networks [348], [351], shared wireless infrastructures [351], seamless subscriber mobility and cellular networks [347], QoS and access control policies made feasible and easier [347], [348], and easy deployment of new applications [236], [300], [351].

One of the first steps toward realizing these features in wireless networks is to provide programmable and flexible stack layers for wireless networks [236], [352]. One of the first examples is OpenRadio [352], which proposes a software abstraction layer for decoupling the wireless protocol definition from the hardware, allowing shared MAC layers across different protocols using commodity multicore platforms. OpenRadio can be seen as the "OpenFlow for wireless networks." Similarly, SoftRAN [236] proposes to rethink the radio access layer of current LTE infrastructures. Its main goal is to allow operators to improve and optimize algorithms for better handovers, fine-grained control of transmit powers, resource block allocation, among other management tasks.

Light virtual access points (LVAPs) is another interesting way of improving the management capabilities of wireless networks, as proposed by the Odin framework [300]. Differently from OpenRadio, it works with existing wireless hardware and does not impose any change on IEEE 802.11 standards. An LVAP is implemented as a unique basic service set identification associated with a specific client, which means that there is a one-to-one mapping between LVAPs and clients. This per-client access point (AP) abstraction simplifies the handling of client associations, authentication, handovers, and unified slicing of both wired and wireless portions of the network. Odin achieves control logic isolation between slices, since LVAPs are the primitive type upon which applications make control decisions, and applications do not have visibility of LVAPs from outside their slice. This empowers infrastructure operators to provide services through Odin applications, such as a mobility manager, client-based load balancer, channel selection algorithm, and wireless troubleshooting application within different network slices. For instance, when a user moves from one AP to another, the network mobility management application can automatically and proactively act and move the client LVAP from one AP to the other. In this way, a wireless client will not even notice that it started to use a different AP because there is no perceptive handoff delay, as it would be the case in traditional wireless networks.

Very dense heterogeneous wireless networks have also been a target for SDN. These DenseNets have limitations due to constraints such as radio access network bottlenecks, control overhead, and high operational costs [296]. A dynamic two-tier SDN controller hierarchy can be adapted to address some of these constraints [296]. Local controllers can be used to take fast and fine-grained decisions, while regional (or "global") controllers can have a broader, coarser grained scope, i.e., that take slower but more global decisions. In such a way, designing a single integrated architecture that encompasses LTE (macro/pico/femto) and WiFi cells, while challenging, seems feasible.

3) Measurement and Monitoring: Measurement and monitoring solutions can be divided into two classes: first, applications that provide new functionality for other networking services; and second, proposals that target to improve features of OpenFlow-based SDNs, such as to reduce control plane overload due to the collection of statistics.

An example of the first class of applications is improving the visibility of broadband performance [6], [353]. An SDN-based broadband home connection can simplify the addition of new functions in measurement systems such as BISmark [353], allowing the system to react to changing conditions in the home network [6]. As an example, a home gateway can perform reactive traffic shaping considering the current measurement results of the home network.

The second class of solutions typically involve different kinds of sampling and estimation techniques to be applied, in order to reduce the burden of the control plane with respect to the collection of data plane statistics. Different techniques have been applied to achieve this goal, such as stochastic and deterministic packet sampling techniques [354], traffic matrix estimation [261], fine-grained monitoring of wildcard rules [355], two-stage Bloom filters [356] to represent monitoring rules and provide high measurement accuracy without incurring in extra memory or control plane traffic overhead [304], and special monitoring functions (extensions to OpenFlow) in forwarding devices to reduce traffic and processing load on the control plane [357]. Point-to-point traffic matrix estimation, in particular, can help in network design and operational tasks such as load balancing, anomaly detection, capacity planning, and network provisioning. With information on the set of active flows in the network, routing information (e.g., from the routing application), flow paths, and flow counters in the switches, it is possible to construct a traffic matrix using diverse aggregation levels for sources and destinations [261].

Other initiatives of this second class propose a stronger decoupling between basic primitives (e.g., matching and counting) and heavier traffic analysis functions such as the detection of anomaly conditions attacks [358]. A stronger separation favors portability and flexibility. For instance, a functionality to detect abnormal flows should not be constrained by the basic primitives or the specific hardware implementation. Put in another way, developers should be empowered with streaming abstractions and higher level programming capabilities.

In that vein, some data and control plane abstractions have been specifically designed for measurement purposes. OpenSketch [311] is a special-purpose southbound API designed to provide flexibility for network measurements. For instance, by allowing multiple measurement tasks to execute concurrently without impairing accuracy. The internal design of an OpenSketch switch can be thought of as a pipeline with three stages (hashing, classification, and counting). Input packets first pass through a hashing function. Then, they are classified according to a matching rule. Finally, the match rule identifies a counting index, which is used to calculate the counter location in the counting stage. While a TCAM with few entries is enough for the classification stage, the flexible counters are stored in SRAM. This makes the OpenSketch's operation efficient (fast matching) and cost-effective (cheaper SRAMs to store counters).

Other monitoring frameworks, such as OpenSample [309] and PayLess [313], propose different mechanisms for delivering real-time, low-latency, and flexible monitoring capabilities to SDN without impairing the load and performance of the control plane. The proposed solutions take advantage of sampling technologies like sFlow [310] to monitor high-speed networks, and flexible collections of loosely coupled (plug-and-play) components to provide abstract network views yielding high-performance and efficient network monitoring approaches [309], [313], [355].

4) Security and Dependability: An already diverse set of security and dependability proposals is emerging in the context of SDNs. Most take advantage of SDN for improving services required to secure systems and networks, such as policy enforcement (e.g., access control, firewalling, middleboxes as middlepipes [27]) [27], [100], [326], [328], [337], DoS attacks detection and mitigation [325], [336], random host mutation [326] (i.e., randomly and frequently mutate the IP addresses of end-hosts to break the attackers' assumption about static IPs, which is the common case) [331], monitoring of cloud infrastructures for fine-grained security inspections (i.e., automatically analyze and detour suspected traffic to be further inspected by specialized network security appliances, such as deep packet inspection systems) [323], traffic anomaly detection [325], [336], [354], fine-grained flow-based network access control [327], fine-grained policy enforcement for personal mobile applications [329], and so on [100], [323], [325], [326],

[328], [331], [337], [354]. Others address OpenFlow-based networks issues, such as flow rule prioritization, security services composition, protection against traffic overload, and protection against malicious administrators [199], [201], [259], [322], [330].

There are essentially two approaches, one involves using SDNs to improve network security, and another for improving the security of the SDN itself. The focus has been, thus far, in the latter.

5) Using SDN to Improve the Security of Current Networks: Probably the first instance of SDN was an application for security policies enforcement [100]. An SDN allows the enforcement to be done on the first entry point to the network (e.g., the Ethernet switch to which the user is connected to). Alternatively, in a hybrid environment, security policy enforcement can be made on a wider network perimeter through programmable devices (without the need to migrate the entire infrastructure to OpenFlow) [328]. With either application, malicious actions are blocked before entering the critical regions of the network. SDN has been successfully applied for other purposes, namely for the detection (and reaction) against distributed denial of service (DDoS) flooding attacks [325], and active security [321]. OpenFlow forwarding devices make it easier to collect a variety of information from the network, in a timely manner, which is very handy for algorithms specialized in detecting DDoS flooding attacks.

The capabilities offered by SDNs in increasing the ability to collect statistics data from the network and of allowing applications to actively program the forwarding devices, are powerful for proactive and smart security policy enforcement techniques such as Active security [321]. This novel security methodology proposes a novel feedback loop to improve the control of defense mechanisms of a networked infrastructure, and is centered around five core capabilities: protect, sense, adjust, collect, and counter. In this perspective, active security provides a centralized programming interface that simplifies the integration of mechanisms for detecting attacks by: 1) collecting data from different sources (to identify attacks); 2) converging to a consistent configuration for the security appliances; and 3) enforcing countermeasures to block or minimize the effect of attacks.

6) Improving the Security of SDN Itself: There are already some research efforts in identifying the critical security threats of SDNs and in augmenting its security and dependability [201], [259], [359]. Early approaches try to apply simple techniques, such as classifying applications and using rule prioritization, to ensure that rules generated by security applications will not be overwritten by lower priority applications [201]. Other proposals try to go a step further by providing a framework for developing security-related applications in SDNs [259]. However, there is still a long way to go in the development of secure and dependable SDN infrastructures [359]. An in-deep overview of SDN security issues and challenges can be found in Section V-F.

7) Data Center Networking: From small enterprises to large-scale cloud providers, most of the existing IT systems and services are strongly dependent on highly scalable and efficient data centers. Yet, these infrastructures still pose significant challenges regarding computing, storage, and networking. Concerning the latter, data centers should be designed and deployed in such a way as to offer high and flexible cross-section bandwidth and low latency, QoS based on the application requirements, high levels of resilience, intelligent resource utilization to reduce energy consumption and improve overall efficiency, agility in provisioning network resources, for example by means of network virtualization and orchestration with computing and storage, and so forth [360]–[362]. Not surprisingly, many of these issues remain open due to the complexity and inflexibility of traditional network architectures.

The emergence of SDN is expected to change the current state of affairs. Early research efforts have indeed showed that data center networking can significantly benefit from SDN in solving different problems such as live network migration [318], improved network management [317], [318], eminent failure avoidance [317], [318], rapid deployment from development to production networks [318], troubleshooting [318], [319], optimization of network utilization [314], [316], [317], [319], dynamic and elastic provisioning of middleboxes-as-a-service [27], and minimization of flow setup latency and reduction of controller operating costs [363]. SDN can also offer networking primitives for cloud applications, solutions to predict network transfers of applications [314], [316], mechanisms for fast reaction to operation problems, network-aware VM placement [315], [319], QoS support [315], [319], real-time network monitoring and problem detection [316], [317], [319], security policy enforcement services and mechanisms [315], [319], and enable programmatic adaptation of transport protocols [314], [320].

SDN can help infrastructure providers to expose more networking primitives to their customers by allowing virtual network isolation, custom addressing, and the placement of middleboxes and virtual desktop cloud applications [315], [364]. To fully explore the potential of virtual networks in clouds, an essential feature is virtual network migration. Similarly to traditional virtual machine migration, a virtual network may need to be migrated when its virtual machines move from one place to another. Integrating live migration of virtual machines and virtual networks is one of the forefront challenges [318]. To achieve this goal, it is necessary to dynamically reconfigure all affected networking devices (physical or virtual). This was shown to be possible with SDN platforms, such as NVP [112].

Another potential application of SDN in data centers is in detecting abnormal behaviors in network operation [317]. By using different behavioral models and collecting the necessary information from elements involved in the operation of a data center (infrastructure, operators, applications), it is possible to continuously build signatures for applications by passively capturing control traffic. Then, the signature history can be used to identify differences in behavior. Every time a difference is detected, operators can reactively or proactively take corrective measures. This can help to isolate abnormal components and avoid further damage to the infrastructure.

8) Toward SDN App Stores: As can be observed in Tables 9 and 10, most SDN applications rely on NOX and OpenFlow. NOX was the first controller available for general use, making it a natural choice for most use cases so far. As indicated by the sheer number of security-related applications, security is probably one of the killer applications for SDNs. Curiously, while most use cases rely on OpenFlow, new solutions such as SoftRAN are considering different APIs, as is the case of the Femto API [253], [303]. This diversity of applications and APIs will most probably keep growing in SDN.

There are other kinds of network applications that do not easily fit in our taxonomy, such as Avior [365], OESS [366], and SDN App Store [367], [368]. Avior and OESS are graphical interfaces and sets of software tools that make it easier to configure and manage controllers (e.g., Floodlight) and OpenFlow-enabled switches, respectively. By leveraging their graphical functions it is possible to program OpenFlow enabled devices without coding in a particular programming language.

The SDN App Store [367], [368], owned by HP, is probably the first SDN application market store. Customers using HP's OpenFlow controller have access to the online SDN App Store and are able to select applications to be dynamically downloaded and installed in the controller. The idea is similar to the Android Market or the Apple Store, making it easier for developers to provide new applications and for customers to obtain them.

I. Cross-Layer Issues

In this section, we look at cross-layer problems such as debugging and troubleshooting, testing, verification, simulation, and emulation. A summary of the existing tools for dealing with these cross-layer issues can be found on Table 11.

1) Debugging and Troubleshooting: Debugging and troubleshooting have been important subjects in computing infrastructures, parallel and distributed systems, embedded systems, and desktop applications [369]–[375]. The two predominant strategies applied to debug and troubleshoot are runtime debugging (e.g., `gdb`-like tools) and post-mortem analysis (e.g., tracing, replay, and visualization). Despite the constant evolution and the emergence of new techniques to improve debugging and troubleshooting, there are still several open avenues and research questions [370].

Debugging and troubleshooting in networking is at a very primitive stage. In traditional networks, engineers and developers have to use tools such as `ping`, `traceroute`, `tcpdump`, `nmap`, `netflow`, and SNMP statistics for debugging and troubleshooting. Debugging a complex network with such primitive tools is very hard. Even when one considers frameworks such as XTrace [374], Netreplay [376], and NetCheck [377], which improve debugging capabilities in networks, it is still difficult to troubleshoot networking infrastructures. For instance, these frameworks require a huge effort in terms of network instrumentation. The additional complexity introduced by different types of devices, technologies, and vendor-specific components and features makes matters worse. As a consequence, these solutions may find it hard to be widely implemented and deployed in current networks.

SDN offers some hope in this respect. The hardware-agnostic software-based control capabilities and the use of open standards for control communication can potentially make debugging and troubleshooting easier. The flexibility and programmability introduced by SDN is indeed opening new avenues for developing better tools to debug, troubleshoot, verify, and test networks [378]–[385].

Early debugging tools for OpenFlow-enabled networks, such as `ndb` [378], OFRewind [379], and NetSight [386], make it easier to discover the source of network problems such as faulty device firmware [378], inconsistent or non-existing flow rules [378], [379], lack of reachability [378], [379], and faulty routing [378], [379]. Similarly to the well-known `gdb` software debugger, `ndb` provides basic debugging actions such as breakpoint, watch, backtrace, single step, and continue. These primitives help application developers to debug networks in a similar way to traditional software. By using `ndb`'s postcards (i.e., a unique packet identifier composed of a truncated copy of the packet's header, the matching flow entry, the switch, and the output port), for instance, a programmer is able to quickly identify and isolate a buggy OpenFlow switch with hardware or software problems. If the switch is presenting abnormal behavior such as corrupting parts of the packet header, by analyzing the problematic flow sequences with a debugging tool, one can find (in a matter of few seconds) where the packets of a flow are being corrupted, and take the necessary actions to solve the problem.

The OFRewind [379] tool works differently. The idea is to record and replay network events, in particular control messages. These usually account for less than 1% of the data traffic and are responsible for 95%–99% of the bugs [385]. This tool allows operators to perform fine-grained tracing of network behavior, being able to decide which subsets of the network will be recorded and, afterwards, select specific parts of the traces to be replayed. These replays provide valuable information to find the root cause of the network misbehavior. Likewise, NetRevert [387] also records the state of OpenFlow networks. However, the primary goal is not to reproduce network behavior, but

Table 11 Debugging, Verification, and Simulation

Group	Solution	Main purpose	Short description
Debugging	ndb [378]	*gdb* alike SDN debugging	Basic debugging primitives that help developers to debug their networks.
	NetSight [386]	multi purpose packet history	Allows to build flexible debugging, monitoring and profiling applications.
	OFRewind [379]	tracing and replay	OFRewind allows operators to do a fine-grained tracing of the network behavior. Operators can decide which subsets of the network will be recorded.
	PathletTracer [411]	inspect layer 2 paths	Allows to inspect low-level forwarding behavior through on-demand packet tracing capabilities.
	SDN traceroute [412]	query OpenFlow paths	Allows users to discover the forwarding behavior of any Ethernet packet and debug problems regarding both forwarding devices and applications.
Verification	Assertion language [401]	debug SDN apps	Enables assertions about the data plane on the apps with support to dynamic changing verification conditions.
	Cbench [392]	evaluate OpenFlow controllers	The Cbench framework can be used to emulate OpenFlow switches which are configured to generate workload to the controller.
	FLOVER [260]	model checking for security policies	FLOVER provides a provably correct and automatic method for verifying security properties with respect to a set of flow rules committed by an OF controller.
	FlowChecker [382]	flow table config verification	A tool used to verify generic properties of global behaviors based on flow tables.
	FLOWGUARD [396]	verify security policy	Provides mechanisms for accurate detection and resolution of firewall policy violations in OpenFlow-based networks.
	FlowTest [413]	verify network policies	Provides the means for testing stateful and dynamic network policies by systematically exploring the state space of the network data plane.
	NetPlumber [400]	real time policy checking	NetPlumber uses a set of policies and invariants to do real time checking. It leverages header space analysis and keeps a dependency graph between rules.
	NICE [380]	remove bugs in controllers	Its main goal is to test controller programs without requiring any type of modification or extra work for application programmers.
	OFCBenchmark [393]	evaluate OpenFlow controllers	creates independent virtual switches, making is possible to emulate different scenarios. Each switch has its how configuration and statistics.
	OFTEN [384]	catch correctness property violations	A framework designed to check SDN systems, analyzing controller and switch interaction, looking for correctness condition violation.
	OFLOPS [381]	evaluate OpenFlow switches	A framework with a rich set of tests for OpenFlow protocol, enabling to measure capabilities of both switch and applications.
	OFLOPS-Turbo [395]	evaluate OpenFlow switches	A framework that integrates OFLOPS with OSNT [414], a 10GbE traffic generation and monitoring system based on NetFPGA.
	PktBlaster [394]	emulation / benchmarking	Integrated test and benchmarking solution that emulates large scale software-defined networks.
	SDLoad [415]	evaluate OpenFlow controllers	A traffic generation framework with customizable workloads to realistically represent different types of applications.
	VeriCon [397]	verify SDN apps	Is a tool for verifying the correctness of SDN applications on large range of topologies and sequences of network events.
	VeriFlow [383]	online invariant verification	It provides real time verification capabilities, while the network state is still evolving.
Simulation & Emulation	DOT [406]	network emulation	Leverages VMs for large scale OpenFlow-based network emulations with resource allocation guarantees.
	fs-sdn [416]	fast simulation	Like Mininet, it provides a simulation environment, but with speed and scalability advantages.
	MaxiNet [405]	network simulation	Similarly to Mininet CE, it is a combination of Mininet tools for large scale simulation of network topologies and architectures.
	Mininet [110]	fast prototyping	It emulates and OpenFlow network using Open vSwitches to provide the exact same semantics of hardware devices.
	Mininet CE [403]	global network modeling	It is a combination of tools to create a Mininet cluster for large scale simulation of network topologies and architectures.
	Mininet-HiFi [402]	reproducible experimentation	Evolution of Mininet to enable repeatable and high fidelity experiments.
	ns-3 [408]	network simulation	The latest version of ns-3 simulator provides support to OpenFlow, enabling to create programmable network devices.
	SDN Cloud DC [404]	cloud data center emulation	The SDN Cloud DC solution allows users to evaluate the performance of their controllers at scale.
	STS [410]	troubleshooting	It simulates a network, allows to generate tricky test cases, and allows interactively examine the state of the network.
	VND-SDN [417]	simulation and analysis	Makes it easier to define experiments via GUI authoring of SDN scenarios and automatic generation of NSDL.

rather to provide rollback recovery in case of failures, which is a common approach used in distributed systems for eliminating transient errors in nodes [388], [389].

Despite the availability of these debugging and verification tools, it is still difficult to answer questions such as: What is happening to my packets that are flowing from point A to point B? What path do they follow? What header modifications do they undergo on the way? To answer some of these questions one could recur to the history of the packets. A packet's history corresponds to the paths it uses to traverse the network, and the header modifications in each hop of the path. NetSight [386] is a platform whose primary goal is to allow applications that use the history of the packets to be built, in order to find out problems in a network. This platform is composed of three essential elements: 1) NetSight, with its dedicated servers that receive and process the postcards for building the packet history; 2) the NetSigh-SwitchAssist, which can be used in switches to reduce the processing burden on the dedicated servers; and 3) the NetSight-HostAssist to generate and process postcards on end hosts (e.g., in the hypervisor on a virtualized infrastructure).

`netwatch` [386], `netshark` [386], and `nprof` [386] are three examples of tools built over NetSight. The first one is a live network invariant monitor. For instance, an alarm can be triggered every time a packet violates any invariant (e.g., no loops). The second one, `netshark`, enables users to define and execute filters on the entire history of packets. With this tool, a network operator can view a complete list of properties of packets at each hop, such as input port, output port, and packet header values. Finally, `nprof` can be used to profile sets of network links to provide data for analyzing traffic patterns and routing decisions that might be contributing to link load.

2) Testing and Verification: Verification and testing tools can complement debugging and troubleshooting. Recent research [380], [382]–[385], [390], [391] has shown that verification techniques can be applied to detect and avoid problems in SDN, such as forwarding loops and black holes. Verification can be done at different layers (at the controllers, network applications, or network devices). Additionally, there are different network properties—mostly topology specific—that can be formally verified, provided a network model is available. Examples of such properties are connectivity, loop freedom, and access control [29]. A number of tools have also been proposed to evaluate the performance of OpenFlow controllers by emulating the load of large-scale networks (e.g., Cbench [392], OFCBenchmark [393], PktBlaster [394]). Similarly, benchmarking tools for OpenFlow switches are also available (e.g., OFLOPS [381] and FLOPS-Turbo [395]).

Tools such as NICE [380] generate sets of diverse streams of packets to test as many events as possible, exposing corner cases such as race conditions. Similarly, OFLOPS [381] provides a set of features and functions that allow the development and execution of a rich set of tests on OpenFlow-enabled devices. Its ultimate goal is to measure the processing capacity and bottlenecks of control applications and forwarding devices. With this tool, users are able to observe and evaluate forwarding table consistency, flow setup latency, flow space granularity, packet modification types, and traffic monitoring capabilities (e.g., counters).

FlowChecker [382], OFTEN [384], and VeriFlow [383] are three examples of tools to verify correctness properties violations on the system. While the former two are based on offline analysis, the latter is capable of online checking of network invariants. Verification constraints include security and reachability issues, configuration updates on the network, loops, black holes, etc.

Other formal modeling techniques, such as Alloy, can be applied to SDNs to identify unexpected behavior [390]. For instance, a protocol specification can be weak when it under-specifies some aspects of the protocol or due to a very specific sequence of events. In such situations, model checking techniques such as Alloy can help to find and correct unexpected behaviors.

Tools such as FLOWGUARD [396] are specifically designed to detect and resolve security policy violations in OpenFlow-enabled networks. FLOWGUARD is able to examine on-the-fly network policy updates, check indirect security violations (e.g., OpenFlow's `Set-Field` actions modification) and perform stateful monitoring. The framework uses five resolution strategies for real-time security policy violation resolution, flow rejecting, dependency breaking, update rejecting, flow removing, and packet blocking [396]. These resolutions are applied over diverse update situations in OpenFlow-enabled networks.

More recently, tools such as VeriCon [397] have been designed to verify the correctness of SDN applications in a large range of network topologies and by analyzing a broad range of sequences of network events. In particular, VeriCon confirms, or not, the correct execution of the SDN program.

One of the challenges in testing and verification is to verify forwarding tables in very large networks to find routing errors, which can cause traffic losses and security breaches, as quickly as possible. In large-scale networks, it is not possible to assume that the network snapshot, at any point, is consistent, due to the frequent changes in routing state. Therefore, solutions such as HSA [398], Anteater [399], NetPlumber [400], Veri-Flow [383], and assertion languages [401] are not suited for this kind of environment. Another important issue is related on how fast the verification process is done, especially in modern data centers that have very tight timing requirements. Libra [391] represents one of the first attempts to address these particular challenges of large-scale networks. This tool provides the means for capturing stable and consistent snapshots of large-scale network deployments, while also applying long prefix matching techniques to increase the scalability of the system. By using MapReduce computations, Libra is capable of verifying the correctness of a network with up to 10 000 nodes within one minute.

Anteater [399] is a tool that analyzes the data plane state of network devices by encoding switch configurations as boolean satisfiability problem (SAT) instances, allowing to use a SAT solver to analyze the network state. The tool is capable of verifying violations of invariants such as loop-free forwarding, connectivity, and consistency. These invariants usually indicate a bug in the network, i.e., their detection helps to increase the reliability of the network data plane.

3) Simulation and Emulation: Simulation and emulation software is of particular importance for fast prototyping and testing without the need for expensive physical devices. Mininet [110] is the first system that provides a quick and easy way to prototype and evaluate SDN protocols and applications. One of the key properties of Mininet is its use of software-based OpenFlow switches in virtualized containers, providing the exact same semantics of hardware-based OpenFlow switches. This means that controllers or applications developed and tested in the emulated environment can be (in theory) deployed in an OpenFlow-enabled network without any modification. Users can easily emulate an OpenFlow network with hundreds of nodes and dozens of switches by using a single personal computer. Mininet-HiFi [402] is an evolution of Mininet that enhances the container-based (lightweight) virtualization with mechanisms to enforce performance isolation, resource provisioning, and accurate monitoring for performance fidelity. One of the main goals of Mininet-HiFi is to improve the reproducibility of networking research.

Mininet CE [403] and SDN Cloud DC [404] are extensions to Mininet for enabling large-scale simulations. Mininet CE combines groups of Mininet instances into one cluster of simulator instances to model global-scale networks. SDN Cloud DC enhances Mininet and POX to emulate an SDN-based intra-DC network by implementing new software modules such as data center topology discovery and network traffic generation. Recent emulation platform proposals that enable large-scale experiments following a distributed approach include Max-iNet [405], DOT [406], and CityFlow [407]. The latter is a project with the main goal of building an emulated control plane for a city of one million inhabitants. Such initiatives are a starting point to provide experimental insights for large-scale SDN deployments.

The capability of simulating OpenFlow devices has also been added to the popular ns-3 simulator [408]. Another simulator is *fs-sdn*, which extends the *fs* simulation engine [409] by incorporating a controller and switching components with OpenFlow support. Its main goal is to provide a more realistic and scalable simulation platform as compared to Mininet. Finally, STS [410] is a simulator designed to allow developers to specify and apply a variety of test cases, while allowing them to interactively examine the state of the network.

V. ONGOING RESEARCH EFFORTS AND CHALLENGES

The research developments we have surveyed so far seek to overcome the challenges of realizing the vision and fulfilling the promises of SDN. While Section IV provided a perspective structured across the layers of the "SDN stack," this section highlights research efforts we consider of particular importance for unleashing the full potential of SDN, and that therefore deserves a specific coverage in this survey.

A. Switch Designs

Currently available OpenFlow switches are very diverse and exhibit notable differences in terms of feature set (e.g., flow table size, optional actions), performance (e.g., fast versus slow path, control channel latency/throughput), interpretation and adherence to the protocol specification (e.g., `BARRIER` command), and architecture (e.g., hardware versus software designs).

1) Heterogeneous Implementations: Implementation choices have a fundamental impact on the behavior, accuracy, and performance of switches, ranging from differences in flow counter behavior [418] to a number of other performance metrics [381]. One approach to accommodate such heterogeneity is through NOSIX, a portable API that separates the application expectations from the switch heterogeneity [246]. To do so, NOSIX provides a pipeline of multiple virtual flow tables and switch drivers. Virtual flow tables are intended to meet the expectations of applications and are ultimately translated by the drivers into actual switch flow tables. Toward taming the complexity of multiple OpenFlow protocol versions with different sets of required and optional capabilities, a roadblock for SDN practitioners, tinyNBI [419], has been proposed as a simple API providing a unifying set of core abstractions of five OpenFlow protocol versions (from 1.0 to 1.4). Ongoing efforts to introduce a new HAL for non-OpenFlow capable devices [420] include the development of open source artifacts like Revised OpenFlow Library (ROFL) and the eXtensible DataPath daemon (xDPd), a framework for creating new OpenFlow data path implementations based on a diverse set of hardware and software platforms. A related open source effort to develop a common library to implement OpenFlow 1.0 and 1.3 protocol endpoints (switch agents and controllers) is libfluid [421], winner of the OpenFlow driver competition organized by the ONF.

Within the ONF, the Forwarding Abstraction Working Group (FAWG) is pursuing another solution to the heterogeneity problem, through table type patterns (TTPs) [121]. A TTP is a standards-based and negotiated switch-level behavioral abstraction. It consists of the relationships between tables forming a graph structure, the types of tables in the graph, a set of the parameterized table properties for each table in the graph, the legal `flow-mod` and `table-mod` commands for each flow table, and the metadata mask that can be passed between each table pair in the graph.

2) Flow Table Capacity: Flow matching rules are stored in flow tables inside network devices. One practical challenge is to provide switches with large and efficient flow tables to store the rules [422]. TCAMs are a common choice to hold flow tables. While flexible and efficient in terms of matching capabilities, TCAMs are costly and usually small (from 4000 to 32 000 entries). Some TCAM chips today integrate 18 million-bit (configured as 500 000 entries * 36 bit per entry) into a single chip working at 133 MHz [423], i.e., capable of 133 million lookups per second. However, these chips are expensive and have a high-power consumption [424], representing a major power drain in a switching device [425]. These are some of the reasons why currently available OpenFlow devices have TCAMs with roughly 8000 entries, where the actual capacity in terms of OpenFlow table size has a nontrivial relationship to the type of flow entries being used [426], [427]. OpenFlow version 1.1 introduced multiple tables, thereby adding extra flexibility and scalability. Indeed, OpenFlow 1.0 implied state explosion due to its flat table model [121]. However, supporting multiple tables in hardware is challenging and limited—yet another motivation for the ongoing ONF FAWG work on TTPs [121].

Some efforts focus on compression techniques to reduce the number of flow entries in TCAMs [428]–[430]. The Espresso heuristic [430] can be used to compress wild cards of OpenFlow-based interdomain routing tables, reducing the forwarding information base (FIB) by 17% and, consequently, saving up to 40 000 flow table entries [428]. Shadow MACs [429] propose label switching for solving two problems, consistent updates and rule space exhaustion, by using opaque values (similar to MPLS labels) to encode fine-grained paths as labels. A major benefit of fixed-size labels is relying on exact-math lookups which can be easily and cost-effectively implemented by simple hardware tables instead of requiring rules to be encoded in expensive TCAM tables.

3) Performance: Today, the throughput of commercial OpenFlow switches varies from 38 to 1000 `flow-mod` per second, with most devices achieving a throughput lower than 500 `flow-mod` per second [431], [432]. This is clearly a limiting factor that will be addressed in the switch design process—support of OpenFlow in existing product lines has been more a retrofitting activity than a clean feature planning and implementation activity. Deployment experiences [433] have pointed to a series of challenges stemming from the limited embedded CPU power of current commercial OpenFlow switches. One approach to handle the problem consists of adding more powerful CPUs into the switches, as proposed in [434]. Others have proposed to rethink the distribution of control actions between external controllers and the OpenFlow agent inside the switch [418]. Our current understanding indicates that an effective way forward is a native design of SDN switches consistent with the evolution of the southbound API standardization activities [121], [435].

4) Evolving Switch Designs and Hardware Enhancements: As in any software/hardware innovation cycle, a number of advancements are to be expected from the hardware perspective to improve SDN capabilities and performance. New SDN switch designs are appearing in a myriad of hardware combinations to efficiently work together with TCAMs, such as static random-access memory (SRAM), dynamic random-access memory (DRAM), reduced-latency DRAM, graphics processing unit (GPU), field-programmable gate array (FPGA), network processors, CPUs, among other specialized network processors [436]–[441]. These early works suggest the need for additional efforts into new hardware architectures for future SDN switching devices. For instance, some proposals target technologies such as GPUs that have demonstrated 20 gigabits per second (Gb/s) with flow tables of up to 1 million exact match entries and up to 1000 wildcard entries [438]. Alternatives to TCAM-based designs include new hardware architectures and components, as well as new and more scalable forwarding planes, such as the one proposed by the Rain Man firmware [442]. Other design solutions, such as parallel lookup models [443], can also be applied to SDN to reduce costs in switching and routing devices. Recent proposals on cache-like OpenFlow switch arrangements [444] shed some light on overcoming the practical limitations of flow table sizes with clever switching designs. Additionally, counters represent another practical challenge in SDN hardware implementations. Many counters already exist, and they could lead to significant control plane monitoring overhead [418]. Software-defined counters (SDCs) [434] have been proposed to provide both scalability and flexibility.

Application-aware SDN architectures are being proposed to generalize the standard OpenFlow forwarding abstractions by including stateful actions to allow processing information from layers 4 to 7 [445]. To this end, application flow tables are proposed as data plane application modules that require only local state, i.e., do not depend on a global view of the network. Those tiny application modules run inside the forwarding devices (and can be installed on demand), alleviating the overhead on the control plane and augmenting the efficiency of certain tasks, which can be kept in the data plane. Similarly, other initiatives propose solutions based on preinstalled state machines. Flow-level State Transition (FAST) [446] allows controllers to proactively program state transitions in forwarding devices, allowing switches to run dynamic actions that require only local information.

Other approaches toward evolving switch designs include CAching in Buckets (CAB), a reactive wildcard caching proposal that uses a geometric representation of the rule set, which is divided into small logical structures (buckets) [447]. Through this technique, CAB is able to solve the rule dependency problem and achieve efficient usage of control plane resources, namely bandwidth, controller processing load, and flow setup latency.

New programmable Ethernet switch chips, such as XPliant Ethernet [448], are emerging into this new market of programmable networks. Its main aim is enabling new protocol support and the addition of new features through software updates, increasing flexibility. One example of such flexibility is the support of GENEVE [39], a recent effort toward generic network virtualization encapsulation protocols, and OpenFlow. The throughput of the first family of XPliant Ethernet chip varies from 880 Gb/s to 3.2 Terabits per second (Tb/s), supporting up to 64 ports of 40 GbE or 50 GbE, for instance.

Microchip companies like Intel are already shipping processors with flexible SDN capabilities to the market [449]. Recent advances in general-purpose CPU technology include a data plane development kit (DPDK) [450] that allows high-level programming of how data packets will be processed directly within network interface cards. Prototype implementations of Intel DPDK accelerated switch shows the potential to deliver high-performance SDN software switches [441]. This trend is likely to continue since high-speed and specialized hardware is needed to boost SDN performance and scalability for large, real-world networks. Hardware-programmable technologies such as FPGA are widely used to reduce time and costs of hardware-based feature implementations. NetFPGA, for instance, has been a pioneering technology used to implement OpenFlow 1.0 switches [437], providing a commodity cost-effective prototyping solution. Another line of work on SDN data planes proposes to augment switches with FPGA to (remotely) define the queue management and scheduling behavior of packet switches [451]. Finally, recent developments have shown that state-of-the-art system-on-chip (SoC) platforms, such as the Xilinx Zynq ZC706 board, can be used to implement OpenFlow devices yielding 88 Gb/s throughput for 1000 flow supporting dynamic updates [452].

5) Native SDN Switch Designs: Most of the SDN switch (re)design efforts so far follow an evolutionary approach to retrofit OpenFlow-specific programmable features into existing hardware layouts, following common wisdom on switch/router designs and consolidated technologies (e.g., SRAM, TCAM, FPGA). One departure from this approach is the ongoing work on forwarding meta-morphosis [435], a reconfigurable match table model inspired from RISC-like pipeline architecture applied to switching chips. This work illustrates the feasibility of realizing a minimal set of action primitives for flexible header processing in hardware, at almost no additional cost or power. Also in line with the core SDN goals of highly flexible and programmable (hardware-based) data planes, POF [120] aims at overcoming some of the limitations of OpenFlow (e.g., expressiveness, support of user-defined protocols, memory efficiency), through generic flow instruction sets. Open source prototypes are available [31] as well as evaluation results showing the line-speed capabilities using a network processing unit (NPU)-based [453] proof of concept implementation. In this line, we already mentioned OpenState [155], another initiative that aims to augment the capability and flexibility of forwarding devices. By taking advantage of eXtended Finite State Machines (XFSMs) [454], [455], OpenState proposes an abstraction—as a super set of OpenFlow primitives—to enable stateful handling of OpenFlow rules inside forwarding devices.

In the same way as TTPs allow controllers to compile the right set of low-lever instructions known to be supported by the switches, a new breed of switch referred to as programmable, protocol-independent packet processor (P4) [456] suggests an evolution path for OpenFlow, based on a high-level compiler. This proposal would allow the functionality of programmable switches (i.e., pipeline, header parsing, field matching) to be not only specified by the controller but also changed in the field. In this model, programmers are able to decide how the forwarding plane processes packets without caring about implementation details. It is then the compiler that transforms the imperative program into a control flow graph that can be mapped to different target switches.

B. Controller Platforms

In the SDN model, the controller platform is a critical pillar of the architecture, and, as such, efforts are being devoted to turn SDN controllers into high-performance, scalable, distributed, modular, and highly available programmer-friendly software. Distributed controller platforms, in particular, have to address a variety of challenges. Deserving special consideration are the latency between forwarding devices and controller instances, fault tolerance, load balancing, consistency, and synchronization, among other issues [7], [457], [458]. Operators should also be able to observe and understand how the combination of different functions and modules can impact their network [459].

As the SDN community learns from the development and operational experiences with OpenFlow controllers (e.g., Beacon [186]), further advancements are expected in terms of raw performance of controller implementations, including the exploitation of hierarchical designs and optimized buffer sizing [460]. One approach to increase the performance of controllers is the IRIS IO engine [461], enabling significant increases in the flow-setup rate of SDN controllers. Another way of reducing the control plane overhead is by keeping a compressed copy of the flow tables in the controller's memory [462].

1) Modularity and Flexibility: A series of ongoing research efforts target the modular and flexible composition of controllers. RAON [463] proposes a recursive abstraction of OpenFlow controllers where each controller sees the controllers below as OpenFlow switches. Open research issues include the definition of suitable interfaces between the different layers in such a hierarchy of controllers. Other open issues to be further investigated in this context are the east/westbound APIs, and their use in enabling suitable hierarchical designs to achieve scalability, modularity, and security [218]. For instance, each level of a hierarchy of controllers can offer different abstractions and scopes for either intradata and interdata center routing, thus increasing scalability and modularity. Similarly, from a security perspective, each hierarchical level may be a part of a different trust domain. Therefore, east/westbound interfaces between the different layers of controllers should be capable of enforcing both intradomain and interdomain security policies.

Another important observation is that, currently, the lack of modularity in most SDN controllers forces developers to re-implement basic network services from scratch in each new application [29].

As in software engineering in general, lack of modularity results in controller implementations that are hard to build, maintain, and extend—and ultimately become resistant to further innovations, resembling traditional "hardware-defined" networks. As surveyed in Section IV-G, SDN programming abstractions (e.g., Pyretic [225]) introduce modularity in SDN applications and simplify their development altogether. Further research efforts (e.g., Corybantic [464]) try to achieve modularity in SDN control programs. Other contributions toward achieving modular controllers can be expected from other areas of computer science (e.g., principles from Operating System [196]) and best practices of modern cloud-scale software applications.

2) Interoperability and Application Portability: Similarly to forwarding device vendor agnosticism that stems from standard southbound interfaces, it is important to foster interoperability between controllers. Early initiatives toward more interoperable control platforms include portable programming languages such as Pyretic [225] and east/westbound interfaces among controllers, such as SDNi [209], ForCES CE–CE interface [30], [211], and ForCES Intra-NE mechanisms [212]. However, these efforts are yet far from fully realizing controller interoperability and application portability.

In contrast to Pyretic [248], PANE [197], Maple [263], and Corybantic [464], which are restricted to traffic engineering applications and/or impose network state conflict resolution at the application level (making application design and testing more complicated), Statesman [465] proposes a framework to enable a variety of loosely coupled network applications to coexist on the same control plane without compromising network safety and performance. This framework makes application development simpler by automatically and transparently resolving conflicts. In other words, Statesman allows a safe composition of uncoordinated or conflicting application's actions.

Another recent approach to simplify network management is the idea of compositional SDN hypervisors [181]. Its main feature is allowing applications written in different languages, or on different platforms, to work together in processing the same traffic. The key integration component is a set of simple prioritized lists of OpenFlow rules, which can be generated by different programming languages or applications.

3) High Availability: In production, SDN controllers need to sustain healthy operation under the pressure of different objectives from the applications they host. Many advances are called for in order to deal with potential risk vectors of controller-based solutions [359]. Certainly, many solutions will leverage on results from the distributed systems and security communities made over the last decade. For instance, recent efforts propose consistent, fault-tolerant data stores for building reliable distributed controllers [198], [213], [458].

Another possible approach toward building low latency, highly available SDN controllers is to exploit controller locality [457], [466]. Classical models of distributed systems, such as LOCAL and CONGEST [467], can be explored to solve this problem. Those models can be used to develop coordination protocols that enable each controller to take independent actions over events that take place in its local neighborhood [457].

Another core challenge relates to the fundamental tradeoffs between the consistency model of state distribution in distributed SDN controllers, the consistency requirements of control applications, and performance [466]. To ease development, the application should ideally not be aware of the vagaries of distributed state. This implies a strong consistency model, which can be achieved with distributed data stores as proposed recently [213]. However, keeping all control data in a consistent distributed data store is unfeasible due to the inherent performance penalties. Therefore, hybrid solutions are likely to coexist requiring application developers to be aware of the tradeoffs and penalties of using, or not, a strong consistency model, a tenet of the distributed Onix controller [7].

High availability can also be achieved through improved southbound APIs and controller placement heuristics and formal models [468]–[470]. These aim to maximize resilience and scalability by allowing forwarding devices to connect to multiple controllers in a cost-effective and efficient way [469]. Early efforts in this direction have already shown that forwarding devices connecting to two or three controllers can typically achieve high availability (up to five nines) and robustness in terms of control plane connectivity [468], [470]. It has also been shown that the number of required controllers is more dependent on the topology than on network size [468]. Another finding worth mentioning is the fact that for most common topologies and network sizes fewer than ten controllers seem to be enough [468].

4) Delegation of Control: To increase operational efficiency, SDN controllers can delegate control functions to report state and attribute value changes, threshold crossing alerts, hardware failures, and so forth. These notifications typically follow a publish/subscribe model, i.e., controllers and applications subscribe (on demand) to the particular class of notifications they are interested in. In addition, these subsystems may provide resilience and trustworthiness properties [471].

Some reasons for delegating control to the data plane include [218]:

- low latency response to a variety of network events;
- the amount of traffic that must be processed in the data plane, in particular in large-scale networks such as data centers;
- low-level functions such as those (byte or bit oriented) required by repetitive synchronous digital hierarchy (SDH) [472] multiplex section overhead;
- functions well understood and standardized, such as encryption, BIP [473], AIS [474] insertion, MAC learning, and codec control message (CCM) [475] exchanges;
- controller failure tolerance, i.e., essential network functions should be able to keep a basic network operation even when controllers are down;
- basic low-level functions usually available in data plane silicon, such as protection switching state machines, CCM counters, and timers;
- all those functions that do not add any value when moved from the data to the control plane.

Strong candidates for execution in the forwarding devices instead of being implemented in the control platforms thus include OAM, ICMP processing, MAC learning, neighbor discovery, defect recognition, and integration [218]. This would not only reduce the overhead (traffic and computing) of the control plane, but also improve network efficiency by keeping basic networking functions in the data plane.

C. Resilience

Achieving resilient communication is a top purpose of networking. As such, SDNs are expected to yield the same levels of availability as legacy and any new alternative technology. Split control architectures as SDN are commonly questioned [476] about their actual capability of being resilient to faults that may compromise the control-to-data plane communications and thus result in "brainless" networks. Indeed, the malfunctioning of particular SDN elements should not result in the loss of availability. The relocation of SDN control plane functionality, from inside the boxes to remote, logically centralized loci, becomes a challenge when considering critical control plane functions such as those related to link failure detection or fast reaction decisions. The resilience of an OpenFlow network depends on fault tolerance in the data plane (as in traditional networks) but also on the high availability of the (logically) centralized control plane functions. Hence, the resilience of SDN is challenging due to the multiple possible failures of the different pieces of the architecture.

As noted in [477], there is a lack of sufficient research and experience in building and operating fault-tolerant SDNs. Google B4 [8] may be one of the few examples that have proven that SDN can be resilient at scale. A number of related efforts [357], [262], [363], [478]–[483] have started to tackle the concerns around control plane split architectures. The distributed controller architectures surveyed in Section IV-D are examples of approaches toward resilient SDN controller platforms with different tradeoffs in terms of consistency, durability, and scalability.

On a detailed discussion on whether the CAP theorem [484] applies to networks, Panda *et al.* [479] argue that the tradeoffs in building consistent, available, and partition-tolerant distributed databases (i.e., CAP theorem) are applicable to SDN. The CAP theorem demonstrates that it is impossible for data store systems to simultaneously achieve strong consistency, availability, and partition tolerance. While availability and partition tolerance problems are similar in both distributed databases and networks, the problem of consistency in SDN relates to the consistent application of policies.

Considering an OpenFlow network, when a switch detects a link failure (`port-down` event), a notification is sent to the controller, which then takes the required actions (reroute computation, precomputed backup path lookup) and installs updated flow entries in the required switches to redirect the affected traffic. Such reactive strategies imply high restoration time due to the necessary interaction with the controller and additional load on the control channel. One experimental work on OpenFlow for carrier-grade networks investigated the restoration process and measured a restoration times in the order of 100 ms [478]. The delay introduced by the controller may, in some cases, be prohibitive.

In order to meet carrier grade requirements (e.g., 50 ms of recovery time), protection schemes are required to mitigate the effects of a separate control plane. Suitable protection mechanisms (e.g., installation of preestablished backup paths in the forwarding devices) can be implemented by means of OpenFlow group table entries using "fast-fail-over" actions. An OpenFlow fault management approach [357] similar to MPLS global path protection could also be a viable solution, provided that OpenFlow switches are extended with end-to-end path monitoring capabilities similarly to those specified by bidirectional forwarding detection (BFD) [485]. Such protection schemes are a critical design choice for larger scale networks and may also require considerable additional flow space. By using primary and secondary path pairs programmed as OpenFlow fast fail-over group table entries, a path restoration time of 3.3 ms has been reported [486] using BFD sessions to quickly detect link failures.

On a related line of data plane resilience, SlickFlow [482] leverages the idea of using packet header space to carry alternative path information to implement resilient source routing in OpenFlow networks. Under the presence of failures along a primary path, packets can be rerouted to alternative paths by the switches themselves without involving the controller. Another recent proposal that uses in-packet information is INFLEX [483], an SDN-based architecture for cross-layer network resilience which provides on-demand path fail-over by having endpoints tag packets with virtual routing plane information that can be used by egress routers to reroute by changing tags upon failure detection.

Similarly to SlickFlow, OSP [280] proposes a protection approach for data plane resilience. It is based on protecting individual segments of a path avoiding the intervention of the controller upon failure. The recovery time depends on the failure detection time, i.e., a few tens of milliseconds in the proposed scenarios. In the same direction, other proposals are starting to appear for enabling fast-fail-over mechanisms for link protection and restoration in OpenFlow-based networks [487].

Language-based solutions to the data plane fault-tolerance problem have also been proposed [262]. In this work, the authors propose a language that compiles regular expressions into OpenFlow rules to express what network paths packets may take and what degree of (link level) fault tolerance is required. Such abstractions around fault tolerance allow developers to build fault recovery capabilities into applications without huge coding efforts.

D. Scalability

Scalability has been one of the major concerns of SDNs from the outset. This is a problem that needs to be addressed in any system—e.g., in traditional networks—and is obviously also a matter of much discussion in the context of SDN [11]. Most of the scalability concerns in SDNs are related to the decoupling of the control and data planes. Of particular relevance are reactive network configurations where the first packet of a new flow is sent by the first forwarding element to the controller. The additional control plane traffic increases network load and makes the control plane a potential bottleneck. Additionally, as the flow tables of switches are configured in real time by an outside entity, there is also the extra latency introduced by the flow setup process. In large-scale networks, controllers will need to be able to process millions of flows per second [488] without compromising the quality of its service. Therefore, these overheads on the control plane and on flow setup latency are (arguably) two of the major scaling concerns in SDN.

As a result, several efforts have been devoted to tackle the SDN scaling concerns, including DevoFlow [418], SDCs [434], DIFANE [489], Onix [7], HyperFlow [195], Kandoo [229], Maestro [188], NOX–MT [187], and Maple [263]. Still related to scalability, the notion of elasticity in SDN controllers is also being pursued [228], [363], [481]. Elastic approaches include dynamically changing the number of controllers and their locations under different conditions [490].

Most of the research efforts addressing scaling limitations of SDN can be classified in three categories: data plane, control plane, and hybrid. While targeting the data plane, proposals such as DevoFlow [418] and SDCs [434] actually reduce the overhead of the control plane by delegating some work to the forwarding devices. For instance, instead of requesting a decision from the controller for every flow, switches can selectively identify the flows (e.g., elephant flows) that may need higher level decisions from the control plane applications. Another example is to introduce more powerful general purpose CPUs in the forwarding devices to enable SDCs. A general purpose CPU and SDCs offer new possibilities for reducing the control plane overhead by allowing software-based implementations of functions for data aggregation and compression, for instance.

Maestro [188], NOX–MT [187], Kandoo [229], Beacon [186], and Maple [263] are examples of the effort on designing and deploying high-performance controllers, i.e., trying to increase the performance of the control plane. These controllers mainly explore well-known techniques from networking, computer architectures, and high-performance computing, such as buffering, pipelining, and parallelism, to increase the throughput of the control platform.

The hybrid category is composed of solutions that try to split the control logic functions between specialized data plane devices and controllers. In this category, DIFANE [489] proposes authoritative (intermediate) switches to keep all traffic in the data plane, targeting a more scalable and efficient control plane. Authoritative switches are responsible for installing rules on the remaining switches, while the controller is still responsible for generating all the rules required by the logic of applications. By dividing the controller work with these special switches, the overall system scales better.

Table 12 provides a nonexhaustive list of proposals addressing scalability issues of SDN. We characterize these issues by application domain (control or data plane), their purpose, the throughput in terms of number of flows per second (when the results of the experiments are reported), and the strategies used. As can be observed, the vast majority are control plane solutions that try to increase scalability by using distributed and multicore architectures.

Some figures are relatively impressive, with some solutions achieving up to 20 million flows/s. However, we should caution the reader that current evaluations

Table 12 Summary and Characterization of Scalability Proposals for SDNs

Solution	Domain	Proposes	Main purpose	Flows/s	Resorts to
Beacon [186]	control plane	a multi-threaded controller	improve controller performance	12.8M	High performance flow processing capabilities using pipeline threads and shared queues.
Beacon cluster [491]	control plane	coordination framework	create clusters of controllers	6.2M	A coordination framework to create high-performance clusters of controllers.
DevoFlow [418]	data plane	thresholds for counters, type of flow detection	reduce the control plane overhead	—	Reduce the control traffic generated by counters statistics monitoring.
DIFANE [489]	control and data plane	authoritative specialized switches	improve data plane performance	500K	Maintain flows in the data plane reducing controller work.
Floodlight [186]	control plane	a multi-threaded controller	Improve controller performance	1.2M	High performance flow processing capabilities.
HyperFlow [195]	control plane	a distributed controller	distribute the control plane	—	Application on top of NOX to provide control message distribution among controllers.
Kandoo [229]	control plane	a hierarchical controller	distribute the control plane hierarchically	—	Use two levels of controller (local and root) to reduce control traffic.
Maestro [188]	control plane	a multi-threaded controller	improve controller performance	4.8M	High performance flow processing capabilities.
Maestro cluster [491]	control plane	coordination framework	create clusters of controllers	1.8M	A coordination framework to create high-performance clusters of controllers.
Maple [263]	control plane	programming language	scaling algorithmic policies	20M	Algorithmic policies and user- and OS-level threads on multicore systems (e.g., 40+ cores).
NOX [186]	control plane	a multi-threaded controller	improve controller performance	5.3M	High performance flow processing capabilities.
NOX-MT [187]	control plane	a multi-threaded controller	improve controller performance	1.8M	High performance flow processing capabilities.
NOX cluster [491]	control plane	coordination framework	create clusters of controllers	3.2M	A coordination framework to create high-performance clusters of controllers.
Onix [7]	control plane	a distributed control platform	robust and scalable control platform	—	Provide a programmable and flexible distributed NIB for application programmers.
SDCs [434]	data plane	Software-Defined Counters	reduce the control plane overhead	—	Remove counters from the ASIC to a general purpose CPU, improving programmability.

consider only simple applications and count basically the number of `packet-in` and `packet-out` messages to measure throughput. The actual performance of controllers will be affected by other factors, such as the number and complexity of the applications running on the controller and security mechanisms implemented. For example, a routing algorithm consumes more computing resources and needs more time to execute than a simple learning switch application. Also, current evaluations are done using plain TCP connections. The performance is very likely to change when basic security mechanisms are put in place, such as TLS, or more advanced mechanisms to avoid eavesdropping, man-in-the-middle and DoS attacks on the control plane.

Another important issue concerning scalability is data distribution among controller replicas in distributed architectures. Distributed control platforms rely on data distribution mechanisms to achieve their goals. For instance, controllers such as Onix, HyperFlow, and ONOS need mechanisms to keep a consistent state in the distributed control platform. Recently, experimental evaluations have shown that high-performance distributed and fault-tolerant data stores can be used to tackle such challenges [213]. Nevertheless, further work is necessary to properly understand state distribution tradeoffs [466].

E. Performance Evaluation

As introduced in Section IV-A, there are already several OpenFlow implementations from hardware and software vendors being deployed in different types of networks, from small enterprise to large-scale data centers. Therefore, a growing number of experiments over SDN-enabled networks is expected in the near future. This will naturally create new challenges, as questions regarding SDN performance and scalability have not yet been properly investigated. Understanding the performance and limitation of the SDN concept is a requirement for its implementation in production networks. There are very few performance evaluation studies of OpenFlow and SDN architecture. Although simulation studies and experimentation are among the most widely used performance evaluation techniques, analytical modeling has its own benefits as well. A closed-form description of a networking architecture paves the way for network designers to have a quick (and approximate) estimate of the performance of their design, without the need to spend considerable time for simulation studies or expensive experimental setup [433].

Some work has investigated ways to improve the performance of switching capabilities in SDN. These mainly consist of observing the performance of OpenFlow-enabled networks regarding different aspects, such as lookup performance [492], hardware acceleration [439], the influence of types of rules and packet sizes [493], performance bottlenecks of current OpenFlow implementations [418], how reactive settings impact the performance on data center networks [494], and the impact of configuration on OpenFlow switches [392].

Design choices can have a significant impact on the lookup performance of OpenFlow switching in Linux operating system using standard commodity network interface cards [492]. Just by using commodity network hardware the packet switching throughput can be improved by up to 25% when compared to one based on soft OpenFlow switching [492]. Similarly, hardware acceleration based on network processors can also be applied to perform OpenFlow switching. In such cases, early reports indicate that performance, in terms of packet delay, can be improved by 20% when compared to conventional designs [439].

By utilizing Intel's DPDK library [450], it has been shown that it is possible to provide flexible traffic steering capability at the hypervisor level (e.g., KVM) without the performance limitations imposed by traditional hardware switching techniques [495], such as SR–IOV [496]. This is particularly relevant since most of the current enterprise deployments of SDN are in virtualized data center infrastructures, as in VMware's NVP solution [112].

Current OpenFlow switch implementations can lead to performance bottlenecks with respect to the CPU load [418]. Yet, modifications on the protocol specification can help reduce the occurrence of these bottlenecks. Further investigations provide measurements regarding the performance of the OpenFlow switch for different types of rules and packet sizes [493].

In data centers, a reactive setting of flow rules can lead to an unacceptable performance when only eight switches are handled by one OpenFlow controller [494]. This means that large-scale SDN deployments should probably not rely on a purely reactive "modus operandi," but rather on a combination of proactive and reactive flow setup.

To foster the evaluation of different performance aspects of OpenFlow devices, frameworks such as OFLOPS [381], OFLOPS-Turbo [395], Cbench [187], and OFCBenchmark [393] have been proposed. They provide a set of tools to analyze the performance of OpenFlow switches and controllers. Cbench [187], [392] is a benchmark tool developed to evaluate the performance of OpenFlow controllers. By taking advantage of the Cbench, it is possible to identify performance improvements for OpenFlow controllers based on different environment and system configurations, such as the number of forwarding devices, network topology, overall network workload, type of equipments, forwarding complexity, and overhead of the applications being executed on top of controllers [187]. Therefore, such tools can help system designers make better decisions regarding the performance of devices and the network, while also allowing end users to measure the device performance and better decide which one is best suited for the target network infrastructure.

Surprisingly, despite being designed to evaluate the performance of controllers, Cbench is currently a single-threaded tool. Therefore, multiple instances have to be started to utilize multiple CPUs. It also only establishes one controller connection for all emulated switches. Unfortunately, this means little can be derived from the results in terms of controller performance and behavior or estimation of different bounds at the moment. For instance, aggregated statistics are gathered for all switches but not for each individual switch. As a result, it is not possible to identify whether all responses of the controller are for a single switch, or whether the capacity of the controller is actually shared among the switches. Flexible OpenFlow controller benchmarks are available though. OFCBenchmark [393] is one of the recent developments. It creates a set of message-generating virtual switches, which can be configured independently from each other to emulate a specific scenario and to maintain their own statistics.

Another interesting question to pose when evaluating the performance of SDN architectures is what is the required number of controllers for a given network topology and where to place the controllers [469], [497]. By analyzing the performance of controllers in different network topologies, it is possible to conclude that one controller is often enough to keep the latency at a reasonable rate [497]. Moreover, as observed in the same experiments, in the general case adding k controllers to the network can reduce the latency by a factor of k. However, there are cases, such as large-scale networks and WANs, where more controllers should be deployed to achieve high reliability and low control plane latency.

Recent studies also show that the SDN control plane cannot be fully physically centralized due to responsiveness, reliability, and scalability metrics [466], [469]. Therefore, distributed controllers are the natural choice for creating a logically centralized control plane, while being capable of coping with the demands of large-scale networks. However, distributed controllers bring additional challenges, such as the consistency of the global network view, which can significantly affect the performance of the network if not carefully engineered. Taking two applications as examples, one that ignores inconsistencies and another that takes inconsistency into consideration, it is possible to observe that optimality is significantly affected when inconsistencies are not considered and that the robustness of an application is increased when the controller is aware of the network state distribution [466].

Most of these initiatives toward identifying the limitations and bottlenecks of SDN architectures can take a lot of time and effort to produce consistent outputs due to the practical development and experimentation requirements. As mentioned before, analytic models can quickly provide performance indicators and potential scalability bottlenecks for an OpenFlow switch-controller system before detailed data are available. While simulation can provide detailed insight into a certain configuration, the analytical model greatly simplifies a conceptual deployment decision. For instance, a Network calculus-based model can be used to evaluate the performance of an SDN switch and the interaction of SDN switches and controllers [498]. The proposed SDN switch model captured the closed form of the packet delay and buffer length inside the SDN switch according to the parameters of a cumulative arrival process. Using recent measurements, the authors have reproduced the packet processing delay of two variants of OpenFlow switches and computed the buffer requirements of an OpenFlow controller. Analytic models based on queuing theory for the forwarding speed and blocking probability of current OpenFlow switches can also be used to estimate the performance of the network [492].

F. Security and Dependability

Cyber attacks against financial institutions, energy facilities, government units, and research institutions are becoming one of the top concerns of governments and agencies around the globe [499]–[504]. Different incidents, such as Stuxnet [503], have already shown the persistence of threat vectors [505]. Put another way, these attacks are capable of damaging a nation's wide infrastructure, which represent a significant and concerning issue. As expected, one of the most common means of executing those attacks is through the network, either the Internet or the local area network. It can be used as a simple transport infrastructure for the attack or as a potentialized weapon to amplify the impact of the attack. For instance, high capacity networks can be used to launch large-scale attacks, even though the attacker has only a low capacity network connection at his premises.

Due to the danger of cyber attacks and the current landscape of digital threats, security and dependability are top priorities in SDN. While research and experimentation on SDNs is being conducted by some commercial players (e.g., Google, Yahoo!, Rackspace, Microsoft), commercial adoption is still in its early stage. Industry experts believe that security and dependability are issues that need to be addressed and further investigated in SDN [359], [506], [507].

Additionally, from the dependability perspective, availability of Internet routers is today a major concern with the widespread of clouds and their strong expectations about the network [508]. It is, therefore, crucial to achieve high levels of availability on SDN control platforms if they are to become the main pillars of networked applications [468].

Different threat vectors have already been identified in SDN architectures [359], as well as several security issues and weaknesses in OpenFlow-based networks [194], [201], [509]–[514]. While some threat vectors are common to existing networks, others are more specific to SDN, such as attacks on control plane communication and logically centralized controllers. It is worth mentioning that most threats vectors are independent of the technology or the protocol (e.g., OpenFlow, POF, and ForCES), because they represent threats on conceptual and architectural layers of SDN itself.

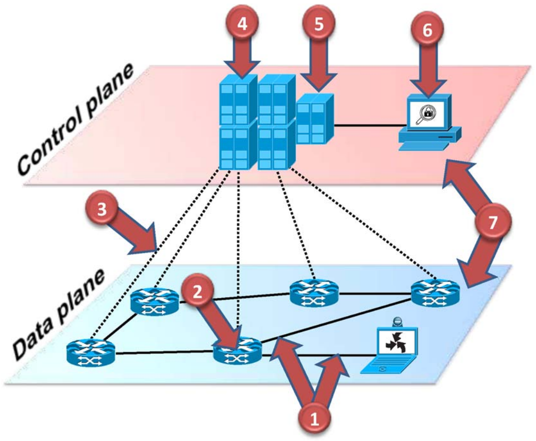

Fig. 10. ***Main threat vectors of SDN architectures.***

As shown in Fig. 10 and Table 13, there are at least seven identified threats vector in SDN architectures. The first threat vector consists of forged or faked traffic flows in the data plane, which can be used to attack forwarding devices and controllers. The second allows an attacker to exploit vulnerabilities of forwarding devices and consequently wreak havoc with the network. Threat vectors three, four, and five are the most dangerous ones, since they can compromise the network operation. Attacks on the control plane, controllers, and applications can easily grant an attacker the control of the network. For instance, a faulty or malicious controller or application could be used to reprogram the entire network for data theft purposes, e.g., in a data center. The sixth threat vector is linked to attacks on and vulnerabilities in administrative stations. A compromised critical computer, directly connected to the control network will empower the attacker

Table 13 SDN Specific Versus Nonspecific Threats

Threat vectors	Specific to SDN?	Consequences in software-defined networks
Vector 1	no	Open door for DDoS attacks.
Vector 2	no	Potential attack inflation.
Vector 3	yes	Exploiting logically centralized controllers.
Vector 4	yes	Compromised controller may compromise the entire network.
Vector 5	yes	Development and deployment of malicious applications on controllers.
Vector 6	no	Potential attack inflation.
Vector 7	no	Negative impact on fast recovery and fault diagnosis.

Table 14 Attacks to OpenFlow Networks

Attack	Security Property	Examples
Spoofing	Authentication	MAC and IP address spoofing, forged ARP and IPv6 router advertisement
Tampering	Integrity	Counter falsification, rule installation, modification affecting data plane.
Repudiation	Non-repudiation	Rule installation, modification for source address forgery.
Information disclosure	Confidentiality	Side channel attacks to figure out flow rule setup.
Denial of service	Availability	Flow requests overload of the controller.
Elevation of privilege	Authorization	Controller take-over exploiting implementation flaws.

with resources to launch more easily an attack to the controller, for instance. Last, threat vector number seven represents the lack of trusted resources for forensics and remediation, which can compromise investigations (e.g., forensics analysis) and preclude fast and secure recovery modes for bringing the network back into a safe operation condition.

As can be observed in Table 13, threat vectors 3 to 5 are specific to SDN as they stem from the separation of the control and data planes and the consequent introduction of a new entity in these networks—the logically centralized controller. The other vectors were already present in traditional networks. However, the impact of these threats could be larger than today—or at least it may be expressed differently—and as a consequence it may need to be dealt with differently.

OpenFlow networks are subject to a variety of security and dependability problems such as spoofing [509], tampering [509], repudiation [509], information disclosure [509], denial of service [509], [511], [512], elevation of privileges [509], and the assumption that all applications are benign and will not affect SDN operation [194]. The lack of isolation, protection, access control, and stronger security recommendations [194], [201], [510]–[512] are some of the reasons for these vulnerabilities. We will explore these next.

1) OpenFlow Security Assessment: There is already a number of identified security issues in OpenFlow-enabled networks. Starting from a STRIDE methodology [515], it is possible to identify different attacks to OpenFlow-enabled networks. Table 14 summarizes these attacks (based on [509]). For instance, information disclosure can be achieved through side channel attacks targeting the flow rule setup process. When reactive flow setup is in place, obtaining information about network operation is relatively easy. An attacker that measures the delay experienced by the first packet of a flow and the subsequent can easily infer that the target network is a reactive SDN, and proceed with a specialized attack. This attack—known as fingerprinting [511]—may be the first step to launch a DoS attack intended to exhaust the resources of the network, for example. If the SDN is proactive, guessing its forwarding rule policies is harder, but still feasible [509]. Interestingly, all reported threats and attacks affect all versions (1.0 to 1.3.1) of the OpenFlow specification. It is also worth emphasizing that some attacks, such as spoofing, are not specific to SDN. However, these attacks can have a larger impact in SDNs. For instance, by spoofing the address of the network controller, the attacker (using a fake controller) could take over the control of the entire network. A smart attack could persist for only a few seconds, i.e., just the time needed to install special rules on all forwarding devices for its malicious purposes (e.g., traffic cloning). Such attack could be very hard to detect.

Taking counter falsification as another example, an attacker can try to guess installed flow rules and, subsequently, forge packets to artificially increase the counter. Such attack would be specially critical for billing and load balancing systems, for instance. A customer could be charged for more traffic than she, in fact used, while a load balancing algorithm may take nonoptimal decisions due to forged counters.

Flow networks include the lack of strong security recommendations for developers, the lack of TLS and access control support on most switch and controller implementations [510], the belief that TCP is enough because links are "physically secure" [510], [512], the fact that many switches have listener mode activated by default (allowing the establishment of malicious TCP connections, for instance) [512] or that flow table verification capabilities are harder to implement when TLS is not in use [260], [510]. In addition, the high denial of service risk posed to centralized controllers is worth mentioning [260], [511], as well as the vulnerabilities in the controllers themselves [260], [359], bugs and vulnerabilities in applications [516], targeted flooding attacks [16], insecure northbound interfaces that can lead to security breaches [16], and the risk of resource depletion attacks [511], [512]. For instance, it has been shown that an attacker can easily compromise control plane communications through DoS attacks and launch a resource depletion attack on control platforms by exploiting a single application such as a learning switch [511], [512].

Another point of concern is the fact that current controllers, such as Floodlight, OpenDaylight, POX, and Beacon, have several security and resiliency issues [194]. Common application development problems (bugs), such as the sudden exit of an application or the continuous

Table 15 Countermeasures for Security Threats in OpenFlow Networks

Measure	Short description
Access control	Provide strong authentication and authorization mechanisms on devices.
Attack detection	Implement techniques for detecting different types of attacks.
Event filtering	Allow (or block) certain types of events to be handled by special devices.
Firewall and IPS	Tools for filtering traffic, which can help to prevent different types of attacks.
Flow aggregation	Coarse-grained rules to match multiple flows to prevent information disclosure and DoS attacks.
Forensics support	Allow reliable storage of traces of network activities to find the root causes of different problems.
Intrusion tolerance	Enable control platforms to maintain correct operation despite intrusions.
Packet dropping	Allow devices to drop packets based on security policy rules or current system load.
Rate limiting	Support rate limit control to avoid DoS attacks on the control plane.
Shorter timeouts	Useful to reduce the impact of an attack that diverts traffic.

allocation of memory space, are enough to crash existing controllers. On the security perspective, a simple malicious action such as changing the value of a data structure in memory can also directly affect the operation and reliability of current controllers. These examples are illustrative that, from a security and dependability perspective, there is still a long way to go.

2) Countermeasures for OpenFlow-Based SDNs: Several countermeasures can be put in place to mitigate the security threats in SDNs. Table 15 summarizes a number of countermeasures that can be applied to different elements of an SDN/OpenFlow-enabled network. Some of these measures, namely rate limiting, event filtering, packet dropping, shorter timeouts, and flow aggregation, are already recommended in the most recent versions of the OpenFlow specification (version 1.3.1 and later). However, most of them are not yet supported or implemented in SDN deployments.

Traditional techniques such as access control, attack detection mechanisms, event filtering (e.g., controller decides which asynchronous messages he is not going to accept), firewalls, and intrusion detection systems can be used to mitigate the impact of or to avoid attacks. They can be implemented in different devices, such as controllers, forwarding devices, middleboxes, and so forth. Middleboxes can be a good option for enforcing security policies in an enterprise because they are (in general) more robust and special purpose (high-performance) devices. Such a strategy also reduces the potential overhead cause by implementing these countermeasures directly on controllers or forwarding devices. However, middleboxes can add extra complexity to the network management, i.e., increase the OPEX at the cost of better performance.

Rate limiting, packet dropping, shorter timeouts, and flow aggregations are techniques that can be applied on controllers and forwarding devices to mitigate different types of attacks, such as denial of service and information disclosure. For instance, reduced timeouts can be used to mitigate the effect of an attack exploring the reactive operation mode of the network to make the controller install rules that divert traffic to a malicious machine. With reduced timeouts, the attacker would be forced to constantly generate a number of forged packets to avoid timeout expiration, making the attack more likely to be detected. Rate limiting and packet dropping can be applied to avoid DoS attacks on the control plane or stop ongoing attacks directly on the data plane by installing specific rules on the devices where the attacks is being originated.

Forensics and remediation encompass mechanisms such as secure logging, event correlation, and consistent reporting. If anything wrong happens with the network, operators should be able to safely figure out the root cause of the problem and put the network to work on a secure operation mode as fast as possible. Additionally, techniques to tolerate faults and intrusions, such as state machine replication [517], proactive–reactive recovery [518], and diversity [210], can be added to control platforms for increasing the robustness and security properties by automatically masking and removing faults. Put differently, SDN controllers should be able to resist against different types of events (e.g., power outages, network disruption, communication failures, network partitioning) and attacks (e.g., DDoS, resource exhaustion) [213], [359]. One of the most traditional ways of achieving high availability is through replication. Yet, proactive–reactive recovery and diversity are two examples of crucial techniques that add value to the system for resisting against different kinds of attacks and failures (e.g., those exploring common vulnerabilities or caused by software aging problems).

Other countermeasures to address different threats and issues of SDN include enhancing the security and dependability of controllers, protection, and isolation of applications [194], [201], [359], [506], trust management between controllers and forwarding devices [359], integrity checks of controllers and applications [359], forensics and remediation [359], [506], verification frameworks [201], [519], [520], and resilient control planes [359], [506], [521], [520]. Protection and isolation mechanisms should be part of any controller. Applications should be isolated from each other and from the controller.

Different techniques such as security domains (e.g., kernel, security, and user level) and data access protection mechanisms should be put in place in order to avoid security threats from network applications.

Implementing trust between controllers and forwarding is another requirement for ensuring that malicious elements cannot harm the network without being detected. An attacker can try to spoof the IP address of the controller and make switches connect to its own controller. This is currently the case since most controllers and switches only establish insecure TCP connections. Complementary, integrity checks on controller and application software can help to ensure that safe code is being bootstrapped, which eliminates harmful software from being started once the system restarts. Besides integrity checks, other things such as highly specialized malware detection systems should be developed for SDN. Third-party network applications should always be scanned for bad code and vulnerabilities because a malicious application represents a significant security threat to the network.

It is worth mentioning that there are also other approaches for mitigating security threats in SDN, such as declarative languages to eliminate network protocol vulnerabilities [265]. This kind of descriptive languages can specify semantic constraints, structural constraints, and safe access properties of OpenFlow messages. Then, a compiler can use these inputs to find programmers' implementation mistakes on message operations. In other words, such languages can help find and eliminate implementation vulnerabilities of southbound specifications.

Proposals providing basic security properties such as authentication [522] and access control [523] are starting to appear. C–BAS [522] is a certificate-based authentication, authorization, and accounting (AAA) architecture for improving the security control on SDN experimental facilities. Solutions in the spirit of C–BAS can be made highly secure and dependable through hybrid system architectures, which combine different technologies and techniques from distributed systems, security, and fault and intrusion tolerance [524]–[526].

G. Migration and Hybrid Deployments

The promises by SDN to deliver easier design, operation, and management of computer networks are endangered by challenges regarding incremental deployability, robustness, and scalability. A prime SDN adoption challenge relates to organizational barriers that may arise due to the first (and second) order effects of SDN automation capabilities and "layer/domain blurring." Some level of human resistance is to be expected and may affect the decision and deployment processes of SDN, especially by those that may regard the control refactorization of SDN as a risk to the current chain of control and command, or even to their job security. This complex social challenge is similar (and potentially larger) to known issues between the transport and IP network divisions of service providers, or the system administrator, storage, networking, and security teams of enterprise organizations. Such a challenge is observable on today's virtualized data centers, through the shift in role and decision power between the networking and server people. Similarly, the development and operations (DevOps) movement has caused a shift in the locus of influence, not only on the network architecture but also on purchasing, and this is an effect that SDN may exacerbate. These changes in role and power causes a second-order effect on the sales division of vendors that are required to adapt accordingly.

Pioneering SDN operational deployments have been mainly greenfield scenarios and/or tightly controlled single administrative domains. Initial rollout strategies are mainly based on virtual switch overlay models or OpenFlow-only network-wide controls. However, a broader adoption of SDN beyond data center silos—and between themselves—requires considering the interaction and integration with legacy control planes providing traditional switching; routing; and operation, administration, and management (OAM) functions. Certainly, rip-and-replace is not a viable strategy for the broad adoption of new networking technologies.

Hybrid networking in SDN should allow deploying OpenFlow for a subset of all flows only, enable OpenFlow on a subset of devices and/or ports only, and provide options to interact with existing OAM protocols, legacy devices, and neighboring domains. As in any technology transition period where forklift upgrades may not be a choice for many, migration paths are critical for adoption.

Hybrid networking in SDN spans several levels. The Migration Working Group of the ONF is tackling the scenario where hybrid switch architectures and hybrid (OpenFlow and non-OpenFlow) devices coexist. Hybrid switches can be configured to behave as a legacy switch or as an OpenFlow switch and, in some cases, as both simultaneously. This can be achieved, for example, by partitioning the set of ports of a switch, where one subset is devoted to OpenFlow-controlled networks, and the other subset to legacy networks. For these subsets to be active at the same time, each one having its own data plane, multitable support at the forwarding engine (e.g., via TCAM partitioning) is required. Besides port-based partitioning, it is also possible to rely on VLAN-based (prior to entering the OpenFlow pipeline) or flow-based partitioning using OpenFlow matching and the `LOCAL` and/or `NORMAL` actions to redirect packets to the legacy pipeline or the switch's local networking stack and its management stack. Flow-based partitioning is the most flexible option, as it allows each packet entering a switch to be classified by an OpenFlow flow description and treated by the appropriate data plane (OpenFlow or legacy).

There are diverse controllers, such as OpenDaylight [13], HP VAN SDN [184], and OpenContrail [183], that have been designed to integrate current non-SDN technologies (e.g., SNMP, PCEP, BGP, and NETCONF) with SDN interfaces such as OpenFlow and OVSDB. Nonetheless, controllers such as ClosedFlow [219] have been recently proposed with the aim of introducing SDN-like programming capabilities in traditional network infrastructures, making the integration of legacy and SDN-enabled networks a reality without side effects in terms of programmability and global network control. ClosedFlow is designed to control legacy Ethernet devices (e.g., Cisco 3550 switches with a minimum IOS of 12.2 SE) in a similar way as OpenFlow controller allows administrators to control OpenFlow-enabled devices. More importantly, ClosedFlow does not impose any change on the forwarding devices. It only takes advantage of the existing hardware and firmware capabilities to mimic an SDN control over the network, i.e., allow dynamic and flexible programmability in the data plane. The next step could be the integration of controllers like ClosedFlow- and OpenFlow-based controllers, promoting interoperability among controllers and a smooth transition from legacy infrastructures to SDN-enabled infrastructure with nearly all the capabilities of a clean-slate SDN-enabled infrastructure.

Furthermore, controllers may have to be separated into distinct peer domains for different reasons, such as scalability, technology, controllers from different vendors, controllers with different service functionality, and diversity of administrative domains [218]. Controllers from different domains, or with distinct purposes, are also required to be backward compatible either by retrofitting or extending existing multidomain protocols (e.g., BGP) or by proposing new SDN-to-SDN protocols (also known as east/westbound APIs).

Some efforts have been already devoted to the challenges of migration and hybrid SDNs. RouteFlow [527] implements an IP level control plane on top of an OpenFlow network, allowing the underlying devices to act as IP routers under different possible arrangements. The Cardigan project [50], [528] has deployed RouteFlow at a live Internet eXchange now for over a year. LegacyFlow [529] extends the OpenFlow-based controlled network to embrace non-OpenFlow nodes. There are also some other early use cases on integrating complex legacy system such as DOCSIS [161], Gigabit Ethernet passive optical network, and DWDM reconfigurable optical add/drop multiplexer (ROADM) [157], [158]. The common grounds of these pieces of work are: 1) considering hybrid as the coexistence of traditional environments of closed vendor's routers and switches with new OpenFlow-enabled devices; 2) targeting the interconnection of both control and data planes of legacy and new network elements; and 3) taking a controller-centric approach, drawing the hybrid line outside of any device itself, but into the controller application space.

Panopticon [530] defines an architecture and methodology to consistently implement SDN inside enterprise legacy networks through network orchestration under strict budget constraints. The proposed architecture includes policy configurations, troubleshooting, and maintenance tasks establishing transitional networks (SDN and legacy) in structures called solitary confinement trees (SCTs), where VLAN IDs are efficiently used by orchestration algorithms to build paths in order to steer traffic through SDN switches. Defying the partial SDN implementation concept, they confirm that this could be a long-term operational strategy solution for enterprise networks.

HybNET [531] presents a network management framework for hybrid OpenFlow-legacy networks. It provides a common centralized configuration interface to build virtual networks using VLANs. An abstraction of the physical network topology is taken into account by a centralized controller that applies a path finder mechanism, in order to calculate network paths and program the OpenFlow switches via REST interfaces and legacy devices using NETCONF [44].

More recently, frameworks such as ESCAPE [532] and its extensions have been proposed to provide multilayer service orchestration in multidomains. Such frameworks combine different tools and technologies such as Click [533], POX [231], OpenDaylight [13], and NETCONF [44]. In other words, those frameworks integrate different SDN solutions with traditional ones. Therefore, they might be useful tools on the process of integrating or migrating legacy networking infrastructure to SDN.

Other hybrid solutions starting to emerge include open source hybrid IP/SDN (OSHI) [534]. OSHI combines Quagga for open shortest path first routing and SDN capable switching devices (e.g., Open vSwitch) on Linux to provide backward compatibility for supporting incremental SDN deployments, i.e., enabling interoperability with non-OF forwarding devices in carrier-grade networks.

While full SDN deployments are straightforward only in some green field deployments such as data center networks or by means of an overlay model approach, hybrid SDN approaches represent a very likely deployment model that can be pursued by different means, including the following [535].

- Topology-based hybrid SDN: Based on a topological separation of the nodes controlled by traditional and SDN paradigms. The network is partitioned in different zones and each node belongs to only one zone.
- Service-based hybrid SDN: Conventional networks and SDN provide different services, where overlapping nodes, controlling a different portion of the FIB (or generalized flow table) of each node. Examples include network-wide services like forwarding that can be based on legacy distributed control, while SDN provides edge-to-edge services such as enforcement of traffic engineering and access policies, or services requiring full traffic visibility (e.g., monitoring).
- Class-based hybrid SDN: Based on the partition of traffic in classes, some controlled by SDN and the remaining by legacy protocols. While each paradigm controls a disjoint set of node forwarding entries, each paradigm is responsible for all network services for the assigned traffic classes.
- Integrated hybrid SDN: A model where SDN is responsible for all the network services and uses traditional protocols (e.g., BGP) as an interface to node FIBs. For example, it can control forwarding paths by injecting carefully selected routes into a

routing system or adjusting protocol settings (e.g., IGP weights). Past efforts on RCPs [85] and the ongoing efforts within ODL [13] can be considered examples of this hybrid model.

In general, benefits of hybrid approaches include enabling flexibility (e.g., easy match on packet fields for middleboxing) and SDN-specific features (e.g., declarative management interface) while partially keeping the inherited characteristics of conventional networking such as robustness, scalability, technology maturity, and low deployment costs. On the negative side, the drawbacks of hybridization include the need for ensuring profitable interactions between the networking paradigms (SDN and traditional) while dealing with the heterogeneity that largely depends on the model.

Initial tradeoff analyses [535] suggest that the combination of centralized and distributed paradigms may provide mutual benefits. However, future work is required to devise techniques and interaction mechanisms that maximize such benefits while limiting the added complexity of the paradigm coexistence.

H. Meeting Carrier-Grade and Cloud Requirements

A number of carrier-grade infrastructure providers (e.g., NTT, AT&T, Verizon, Deutsche Telekom) are at the core of the SDN community with the ultimate goal of solving their long standing networking problems. In the telecom world, NTT can be considered one of the forefront runners in terms of investing in the adoption and deployment of SDN in all kinds of network infrastructures, from backbone, data center, to edge customers [269]. In 2013, NTT launched an SDN-based, on-demand elastic provisioning platform of network resources (e.g., bandwidth) for HD video broadcasters [536]. Similarly, as a global cloud provider with data centers spread across the globe [537], the same company launched a similar service for its cloud customers, who are now capable of taking advantage of dynamic networking provisioning intradata and interdata centers [538]. AT&T is another telecom company that is investing heavily in new services, such as user-defined network clouds, that take advantage of recent developments in NFV and SDN [539]. As we mentioned before, SDN and NFV are complementary technologies that can be applicable to different types of networks, from local networks and data centers to transport networks [540]–[545]. Recently, several research initiatives have worked toward combining SDN and NFV through Intel's DPDK, a set of libraries and drivers that facilitates the development of network-intensive applications and allows the implementation of fine-grained network functions [546]. Early work toward service chaining has been proposed by combining SDN and NFV technologies [27], [547]–[550], and studies around the ForCES's [30] applicability to SDN-enhanced NFV have also come to light [540]. These are some of the early examples of the opportunities SDNs seem to bring to telecom and cloud providers.

Carrier networks are using the SDN paradigm as the technology means for solving a number of long standing problems. Some of these efforts include new architectures for a smooth migration from the current mobile core infrastructure to SDN [222], and techno-economic models for virtualization of these networks [551], [552]; carrier-grade OpenFlow virtualization schemes [112], [553], including virtualized broadband access infrastructures [554], techniques that are allowing the offer of network-as-a-service [555]; programmable GEPON and DWDM ROADM [157]–[160]; large-scale interautonomous systems (ASs) SDN-enabled deployments [556]; flexible control of network resources [557], including offering MPLS services using an SDN approach [558]; and the investigation of novel network architectures, from proposals to separate the network edge from the core [559], [560], with the latter forming the fabric that transports packets as defined by an intelligent edge, to software-defined Internet exchange points [528], [561].

Use case analysis [562] of management functions required by carrier networks have identified a set of requirements and existing limitations in the SDN protocol toolbox. For instance, it has been pinpointed that OF-Config [54] needs a few extensions in order to meet the carrier requirements, such as physical resource discovery, logical link configuration, logical switch instantiation, and device and link OAM configuration [562]. Similarly, OpenFlow extensions have also been proposed to realize packet-optical integration with SDN [563]. In order to support SDN concepts in large-scale wide area networks, different extensions and mechanisms are required, both technology specific (e.g., MPLS BFD) and technology agnostic, such as: resiliency mechanisms for surviving link failures [486], failures of controller or forwarding elements; solutions for integrating residential customer services in different forms (i.e., support also current technologies); new energy-efficient networking approaches; QoS properties for packet classification, metering, coloring, policing, shaping, and scheduling; and multilayer aspects outlining different stages of packet-optical integration [563]–[565].

SDN technology also brings new possibilities for cloud providers. By taking advantage of the logically centralized control of network resources [8], [566], it is possible to simplify and optimize network management of data centers and achieve: 1) efficient intradata-center networking, including fast recovery mechanisms for the data and control planes [478], [567], [568], adaptive traffic engineering with minimal modifications to DC networks [278], simplified fault-tolerant routing [569], performance isolation [570], and easy and efficient resource migration (e.g., of VMs and virtual networks) [478]; 2) improved interdata-center communication, including the ability to fully utilize the expensive high-bandwidth links without impairing quality of service [8], [571]; 3) higher levels of reliability (with novel fault management mechanisms, etc.) [478], [486], [567], [569]; and 4) cost reduction by replacing

Table 16 Carrier-Grade and Cloud Provider Expectations and Challenges

What	Currently	Expected with SDN
Resource Provisioning	Complex load balancing configuration.	Automatic load balancing reconfiguration. [573], [8]
	Low virtualization capabilities across hardware platforms	NFV for virtualizing network functionality across hardware appliances. [572], [539]
	Hard and costly to provide new services.	Create and deploy new network service quickly. [572], [539]
	No bandwidth on demand.	Automatic bandwidth on demand. [552]
	Per network element scaling.	Better incremental scaling. [573], [567]
	Resources statically pre-provisioned.	Dynamic resource provisioning in response to load. [573], [8], [572], [551], [566]
Traffic Steering	All traffic is filtered.	Only targeted traffic is filtered. [573]
	Fixed only.	Fixed and mobile. [573]
	Per network element scaling.	Better incremental scaling. [551], [567]
	Statically configured on a per-device basis.	Dynamically configurable. [8], [552], [574]
Ad Hoc Topologies	All traffic from all probes collected.	Only targeted traffic from targeted probes is collected.
	Massive bandwidth required.	Efficient use of bandwidth. [8], [552]
	Per network element scaling.	Better incremental scaling. [573], [552]
	Statically configured.	Dynamically configured. [573], [575], [536]
Managed Router Services	Complex configuration, management and upgrade.	Simplified management and upgrade. [8], [573], [572], [552], [567]
	Different kinds of routers, such as changeover (CO).	No need for CO routers, reducing aggregation costs. [573], [572], [551]
	Manual provisioning.	Automated provisioning. [573], [552], [574]
	On-premises router deployment.	Virtual routers (either on-site or not). [552], [573], [551]
	Operational burden to support different equipments.	Reduced technology obsolescence. [551]
	Router change-out as technology or needs change.	Pay-as-you grow CAPEX model. [551]
	Systems complex and hard to integrate.	Facilitates simplified system integrations. [573], [572], [575]
Revenue Models	Fixed long term contracts.	More flexible and on-demand contracts. [552], [557]
	Traffic consumption.	QoS metrics per-application. [552], [567], [567], [576]
Middleboxes Deployment & Management	Composition of services is hard to implement.	Easily expand functionality to meet the infrastructure needs. [572]
	Determine where to place middleboxes a priori (e.g., large path inflation problems).	Dynamic placement using shortest or least congested path. [292], [576], [575]
	Excessive over-provisioning to anticipate demands.	Scale up to meet demands, and scale down to conserve resources (elastic middleboxes). [573], [551]
Other Issues	Energy saving strategies are hard to implement.	Flexible and easy to deploy energy saving strategies. [567]
	Complex and static control and data plane restoration techniques.	Automated and flexible restoration techniques for both control and data plane. [567]

complex, expensive hardware by simple and cheaper forwarding devices [8], [572].

Table 16 summarizes some of the carrier-grade network and cloud infrastructure providers' requirements. In this table, we show the current challenges and what is to be expected with SDN. As we saw before, some of the expectations are already becoming a reality, but many are still open issues. What seems to be clear is that SDN represents an opportunity for telecom and cloud providers, in providing flexibility, cost effectiveness, and easier management of their networks.

I. SDN: The Missing Piece Toward Software-Defined Environments

The convergence of different technologies is enabling the emergence of fully programmable IT infrastructures. It is already possible to dynamically and automatically configure or reconfigure the entire IT stack, from the network infrastructure up to the applications, to better respond to workload changes. Recent advances makes on-demand provisioning of resources possible, at nearly all infrastructural layers. The fully automated provisioning and orchestration of IT infrastructures as been recently named software-defined environments (SDEs) [171], [172], by IBM. This is a novel approach that is expected to have significant potential in simplifying IT management, optimizing the use of the infrastructure, reduce costs, and reduce the time to market of new ideas and products. In an SDE, workloads can be easily and automatically assigned to the appropriate IT resources based on application characteristics, security and service level policies, and the best-available resources to deliver continuous, dynamic optimization and reconfiguration to address infrastructure issues in a rapid and responsive manner. Table 17 summarizes the traditional approaches and some of the key features being enabled by SDEs [577], [578].

In an SDE, the workloads are managed independently of the systems and underlying infrastructure, i.e., are not

Table 17 SDE Pushing IT to the Next Frontier

Traditionally	Expected with SDEs
IT operations manually map the resources for apps for software deployment.	Software maps resources to the workload and deploys the workload.
Networks are mostly statically configured and hard to change.	Networks are virtualized and dynamically configured on-demand.
Optimization and reconfiguration to reactively address issues are manual.	Analytics-based optimization and reconfiguration of infrastructure issues.
Workloads are typically manually assigned to resources.	Workloads are dynamically assigned.

tied to a specific technology or vendor [171], [172]. Another characteristic of this new approach is to offer a programmatic access to the environment as a whole, selecting the best available resources based on the current status of the infrastructure, and enforcing the policies defined. In this sense, it shares much of the philosophy of SDN. Interestingly, one of the missing key pieces of an SDE was, until now, SDN.

The four essential building blocks of an SDE [171], [172], [578] are:

- SDNs [579], [580];
- software-defined storage (SDS) [577];
- software-defined compute (SDC) [171];
- software-defined management (SDM) [581].

In the last decade, the advances in virtualization of compute and storage, together with the availability of sophisticated cloud orchestration tools have enabled SDS, SDC, and SDM. These architectural components have been widely used by cloud providers and for building IT infrastructures in different enterprise environments. However, the lack of programmable network control has so far hindered the realization of a complete SDE. SDN is seen as the technology that may fill this gap, as attested by the emergence of cloud-scale network virtualization platforms based on this new paradigm [112].

The IBM SmartCloud Orchestrator is one of the first examples of an SDE [171], [172]. It integrates compute, storage, management, and networking in a structured way. Fig. 11 gives a simplified overview of an SDE, by taking the approach developed by IBM as its basis. The main idea of an SDE-based infrastructure is that the business needs that define the workloads trigger the reconfiguration of the global IT infrastructure (compute, storage, network). This is an important step toward a more customizable IT infrastructure that focuses on the business requirements rather than on the limitations of the infrastructure itself.

VI. CONCLUSION

Traditional networks are complex and hard to manage. One of the reasons is that the control and data planes are vertically integrated and vendor specific. Another, concurring reason, is that typical networking devices are also tightly tied to line products and versions. In other words, each line of product may have its own particular configuration and management interfaces, implying long cycles for producing product updates (e.g., new firmware) or upgrades (e.g., new versions of the devices). All this has given rise to vendor lock-in problems for network infrastructure owners, as well as posing severe restrictions to change and innovation.

SDN created an opportunity for solving these long-standing problems. Some of the key ideas of SDN are the introduction of dynamic programmability in forwarding devices through open southbound interfaces, the decoupling of the control and data plane, and the global view of the

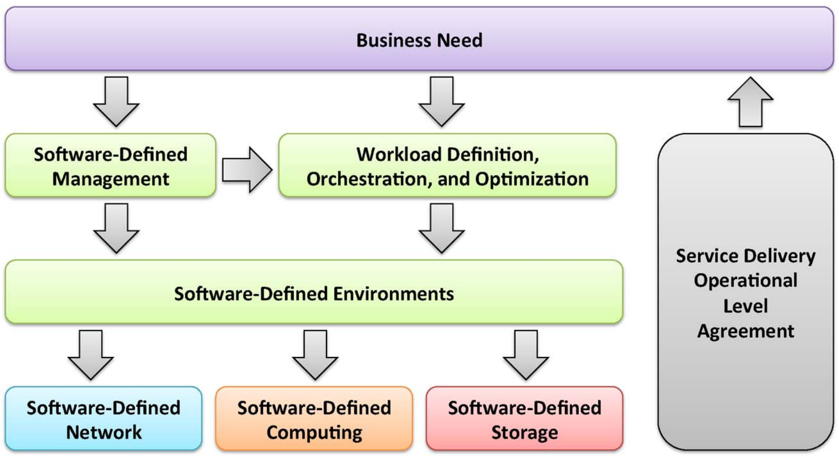
Business Need
Software-Defined Management
Workload Definition, Orchestration, and Optimization
Software-Defined Environments
Software-Defined Network
Software-Defined Computing
Software-Defined Storage
Service Delivery Operational Level Agreement

Fig. 11. *Overview of an IT infrastructure based on an SDE.*

network by logical centralization of the "network brain." While data plane elements became dumb, but highly efficient and programmable packet forwarding devices, the control plane elements are now represented by a single entity, the controller or NOS. Applications implementing the network logic run on top of the controller and are much easier to develop and deploy when compared to traditional networks. Given the global view, consistency of policies is straightforward to enforce. SDN represents a major paradigm shift in the development and evolution of networks, introducing a new pace of innovation in networking infrastructure.

In spite of recent and interesting attempts to survey this new chapter in the history of networks [14]–[16], the literature was still lacking, to the best of our knowledge, a single extensive and comprehensive overview of the building blocks, concepts, and challenges of SDNs. Trying to address this gap, this paper used a layered approach to methodically dissect the state of the art in terms of concepts, ideas, and components of SDN, covering a broad range of existing solutions, as well as future directions.

We started by comparing this new paradigm with traditional networks and discussing how academy and industry helped shape SDN. Following a bottom-up approach, we provided an in-depth overview of what we consider the eight fundamental facets of the SDN problem: 1) hardware infrastructure; 2) southbound interfaces; 3) network virtualization (hypervisor layer between the forwarding devices and the NOSs); 4) NOSs (SDN controllers and control platforms); 5) northbound interfaces (common programming abstractions offered to network applications); 6) virtualization using slicing techniques provided by special purpose libraries and/or programming languages and compilers; 7) network programming languages; and finally, 8) network applications.

SDN has successfully managed to pave the way toward a next-generation networking, spawning an innovative research and development environment, promoting advances in several areas: switch and controller platform design, evolution of scalability and performance of devices and architectures, promotion of security and dependability.

We will continue to witness extensive activity around SDN in the near future. Emerging topics requiring further research are, for example: the migration path to SDN, extending SDN toward carrier transport networks, realization of the network-as-a-service cloud computing paradigm, or SDEs. As such, we would like to receive feedback from the networking/SDN community as this novel paradigm evolves, to make this a "living document" that gets updated and improved based on the community feedback. We have set up a *github* repository (https://github.com/SDN-Survey/latex/wiki) for this purpose, and we invite our readers to join us in this communal effort. Additionally, new releases of the survey will be available at http://arxiv.org/abs/1406.0440. ■

Acknowledgment

The authors would like to thank the anonymous reviewers and a number of fellows that have contributed to this work: J. Rexford for her feedback on an early version of this work and encouragement to get it finished; S. Seetharaman for reviewing the draft and providing inputs to alternative SDN views; D. Meyer for his thoughts on organizational challenges; T. Nadeau for his inputs on OpenDaylight; and L. M. Contreras Murillo for his contributions to SDN standardization. In addition, the authors would also like to acknowledge the several contributions from the community, namely, from A. A. Lazar, C. Cascone, G. Patra, H. Evangelos, J. Ancieta, J. Stringer, K. Pentikousis, L. de Paula, M. Canini, P. Wette, R. Fontes, R. Rosa, R. Costa, R. de Freitas Gesuatto, and W. John.

REFERENCES

[1] T. Benson, A. Akella, and D. Maltz, "Unraveling the complexity of network management," in *Proc. 6th USENIX Symp. Networked Syst. Design Implement.*, 2009, pp. 335–348.

[2] B. Raghavan *et al.*, "Software-defined internet architecture: Decoupling architecture from infrastructure," in *Proc. 11th ACM Workshop Hot Topics Netw.*, 2012, pp. 43–48.

[3] A. Ghodsi *et al.*, "Intelligent design enables architectural evolution," in *Proc. 10th ACM Workshop Hot Topics Netw.*, 2011, pp. 3:1–3:6.

[4] N. Mckeown, "How SDN will shape networking," Oct. 2011. [Online]. Available: http://www.youtube.com/watch?v=c9-K5O_qYgA

[5] S. Schenker, "The future of networking, the past of protocols," Oct. 2011. [Online]. Available: http://www.youtube.com/watch?v=YHeyuD89n1Y

[6] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Commun. Mag.*, vol. 51, no. 2, pp. 114–119, Feb. 2013.

[7] T. Koponen *et al.*, "Onix: A distributed control platform for large-scale production networks," in *Proc. 9th USENIX Conf. Oper. Syst. Design Implement.*, 2010, pp. 1–6.

[8] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined wan," in *Proc. ACM SIGCOMM Conf.*, 2013, pp. 3–14.

[9] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[10] Open Networking Foundation (ONF), 2014. [Online]. Available: https://www.opennetworking.org/

[11] S. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Commun. Mag.*, vol. 51, no. 2, pp. 136–141, Feb. 2013.

[12] VMware, Inc., NSX Virtualization Platform, 2013. [Online]. Available: https://www.vmware.com/products/nsx/

[13] OpenDaylight, A Linux Foundation Collaborative Project, 2013. [Online]. Available: http://www.opendaylight.org

[14] A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using OpenFlow: A survey," *IEEE Commun. Surv. Tut.*, vol. 16, no. 1, pp. 493–512, First Quart. 2014.

[15] B. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, future of programmable networks," *IEEE Commun. Surv. Tut.*, vol. 16, no. 3, pp. 1617–1634, Third Quart. 2014.

[16] Y. Jarraya, T. Madi, and M. Debbabi, "A survey and a layered taxonomy of software-defined networking," *IEEE Commun. Surv. Tut.*, vol. 16, no. 4, pp. 1955–1980, Fourth Quart. 2014.

[17] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN," *Queue*, vol. 11, no. 12, pp. 20:20–20:40, Dec. 2013.

[18] R. Presuhn, "Version 2 of the protocol operations for the simple network management protocol (SNMP)," Internet Engineering Task Force, RFC 3416 (Internet Standard), Dec. 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3416.txt

[19] J. Pan, S. Paul, and R. Jain, "A survey of the research on future internet architectures,"

IEEE Commun. Mag., vol. 49, no. 7, pp. 26–36, Jul. 2011.

[20] N. Feamster and H. Balakrishnan, "Detecting BGP configuration faults with static analysis," in *Proc. 2nd Conf. Symp. Netw. Syst. Design Implement.*, 2005, vol. 2, pp. 43–56.

[21] R. Barrett, S. Haar, and R. Whitestone, "Routing snafu causes internet outage," *Interactive Week*, vol. 25, 1997.

[22] K. Butler, T. Farley, P. McDaniel, and J. Rexford, "A survey of BGP security issues and solutions," *Proc. IEEE*, vol. 98, no. 1, pp. 100–122, Jan. 2010.

[23] J. Sherry and S. Ratnasamy, "A survey of enterprise middlebox deployments," Electr. Eng. Comput. Sci. Dept., Univ. California, Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2012-24, Feb. 2012.

[24] K. Greene, "10 Breakthrough Technologies: Software-defined Networking *MIT Technol. Rev.*, 2009. [Online]. Available: http://www2.technologyreview.com/article/412194/tr10-software-defined-networking/

[25] P. Newman, G. Minshall, and T. L. Lyon, "IP switching—ATM under IP," *IEEE/ACM Trans. Netw.*, vol. 6, no. 2, pp. 117–129, Apr. 1998.

[26] N. Gude *et al.*, "NOX: Towards an operating system for networks," *Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, 2008.

[27] H. Jamjoom, D. Williams, and U. Sharma, "Don't call them middle-boxes, call them middlepipes," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 19–24.

[28] H. Alkhatib *et al.*, "IEEE CS 2022 Report (Draft)," IEEE Computer Society, Tech. Rep., Feb. 2014.

[29] M. Casado, N. Foster, and A. Guha, "Abstractions for software-defined networks," *ACM Commun.*, vol. 57, no. 10, pp. 86–95, Sep. 2014.

[30] A. Doria *et al.*, "Forwarding and control element separation (ForCES) protocol specification," Internet Engineering Task Force, Mar. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5810.txt

[31] H. Song, "Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 127–132.

[32] T. D. Nadeau and K. Gray, *SDN: Software Defined Networks*, 1st ed. Sebastopol, CA, USA: O'Reilly Media, 2013.

[33] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Comput. Netw.*, vol. 54, no. 5, pp. 862–876, 2010.

[34] B. Davie and J. Gross, "A stateless transport tunneling protocol for network virtualization (STT)," Internet Engineering Task Force, Apr. 2014. [Online]. Available: http://tools.ietf.org/html/draft-davie-stt-06

[35] M. Mahalingam *et al.*, "VXLAN: A framework for overlaying virtualized layer 2 networks over layer 3 networks," Internet Engineering Task Force, Internet Draft, Nov. 2013. [Online]. Available: http://www.ietf.org/id/draft-mahalingam-dutt-dcops-vxlan-06.txt

[36] M. Sridharan *et al.*, "NVGRE: Network virtualization using generic routing encapsulation," Internet Engineering Task Force, Internet Draft, Aug. 2013. [Online]. Available: http://tools.ietf.org/id/draft-sridharan-virtualization-nvgre-03.txt

[37] F. Maino, V. Ermagan, Y. Hertoghs, D. Farinacci, and M. Smith "LISP control plane for network virtualization overlays," Internet Engineering Task Force, Oct. 2013. [Online]. Available: http://tools.ietf.org/html/draft-maino-nvo3-lisp-cp-03

[38] Y. Hertoghs *et al.*, "A unified LISP mapping database for L2 and L3 network virtualization overlays," Internet Engineering Task Force, Feb. 2014. [Online]. Available: http://tools.ietf.org/html/draft-hertoghs-nvo3-lisp-controlplane-unified-01

[39] J. Gross, T. Sridhar, P. Garg, C. Wright, and I. Ganga, "Geneve: Generic network virtualization encapsulation," Internet Engineering Task Force, Internet Draft, Feb. 2014. [Online]. Available: http://tools.ietf.org/id/draft-gross-geneve-00.txt

[40] R. Jain and S. Paul, "Network virtualization and software defined networking for cloud computing: A survey," *IEEE Commun. Mag.*, vol. 51, no. 11, pp. 24–31, Nov. 2013.

[41] E. Haleplidis *et al.*, "SDN layers and architecture terminology," Internet Engineering Task Force, Internet Draft, Sep. 2014. [Online]. Available: http://www.ietf.org/id/draft-irtf-sdnrg-layer-terminology-02.txt

[42] Y. Rekhter, T. Li, and S. Hares, "A border gateway protocol 4 (BGP-4)," Internet Engineering Task Force, RFC 4271 (Draft Standard), Jan. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4271.txt

[43] J. Vasseur and J. L. Roux, "Path computation element (PCE) communication protocol (PCEP)," Internet Engineering Task Force, RFC 5440 (Proposed Standard), Mar. 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5440.txt

[44] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman "Network configuration protocol (NETCONF)," Internet Engineering Task Force, RFC 6241 (Proposed Standard), Jun. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6241.txt

[45] A. Corradi, M. Fanelli, and L. Foschini, "VM consolidation: A real case based on openstack cloud," *Future Generat. Comput. Syst.*, vol. 32, pp. 118–127, 2014.

[46] A. Shang, J. Liao, and L. Du, "Pica8 Xorplus, 2014. [Online]. Available: http://sourceforge.net/projects/xorplus/

[47] P. Jakma and D. Lamparter, "Introduction to the quagga routing suite," *IEEE Network*, vol. 28, no. 2, pp. 42–48, Mar. 2014.

[48] NetFPGA, 2014. [Online]. Available: http://netfpga.org/

[49] Linux Foundation, "Open platform for NFV," Sep. 2014. [Online]. Available: https://www.opnfv.org

[50] C. E. Rothenberg *et al.*, "When open source meets network control planes," *IEEE Computer, Special Issue on Software-Defined Networking*, vol. 47, no. 11, pp. 46–54, Nov. 2014.

[51] Open Networking Foundation (ONF), "SDN architecture," Tech. Rep., Jun. 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf

[52] Open Networking Foundation (ONF), "Conformance test specification for OpenFlow switch specification," Tech. Rep., Jun. 2013. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-test/conformance-test-spec-openflow-1.0.1.pdf

[53] Open Networking Foundation (ONF), "OpenFlow switch specification," Tech. Rep., Oct. 2013. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf

[54] Open Networking Foundation (ONF), "OpenFlow management and configuration protocol (OF-CONFIG) v1.2," Tech. Rep., 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-config-1.2.pdf

[55] Open Networking Foundation (ONF), "OpenFlow notifications framework OpenFlow management," Oct. 2013. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-notifications-framework-1.0.pdf

[56] Open Networking Foundation (ONF), "OpenFlow table type patterns," Tech. Rep., Aug. 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/OpenFlow%20Table%20Type%20Patterns%20v1.0.pdf

[57] Open Networking Foundation (ONF), "Optical transport use cases," Tech. Rep., 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/optical-transport-use-cases.pdf

[58] Open Networking Foundation (ONF), "Requirements analysis for transport OpenFlow/SDN," Tech. Rep., Aug. 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/onf2014.227_Optical_Transport_Requirements.pdf

[59] Open Networking Foundation (ONF), "Migration use cases and methods," Migration Working Group, Tech. Rep., 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/use-cases/Migration-WG-Use-Cases.pdf

[60] Open Networking Foundation (ONF), "Software-defined networking: The new norm for networks," Tech. Rep., Apr. 2012. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[61] H. Xie, T. Tsou, D. Lopez, and H. Yin, "Use cases for ALTO with software defined networks," Internet Engineering Task Force, Internet Draft, Jun. 2012. [Online]. Available: http://tools.ietf.org/html/draft-xie-alto-sdn-use-cases-01

[62] A. Atlas, J. Halpern, S. Hares, D. Ward, and T. Nadeau, "An architecture for the interface to the routing system," Internet Engineering Task Force, Internet Draft, Jul. 2014. [Online]. Available: https://tools.ietf.org/html/draft-ietf-i2rs-architecture-05

[63] R. Enns, "NETCONF configuration protocol," Internet Engineering Task Force, Internet Draft, Dec. 2004. [Online]. Available: http://tools.ietf.org/html/rfc4741

[64] L. Kreeger, D. Dutt, T. Narten, and D. Black, "Network virtualization NVE to NVA control protocol requirements," Internet Engineering Task Force, Internet Draft, Apr. 2014. [Online]. Available: http://tools.ietf.org/html/draft-ietf-nvo3-nve-nva-cp-req-02

[65] D. King and A. Farrel, "A PCE-based architecture for application-based network operations," Internet Engineering Task Force, Internet Draft, Aug. 2014. [Online]. Available: https://tools.ietf.org/html/draft-farrkingel-pce-abno-architecture-11

[66] D. Dhody, L. C. Y. Lee, O. Gonzalez, and N. Ciulli, "Cross stratum optimization enabled path computation," Internet Engineering Task Force, Internet Draft, Jul. 2014. [Online]. Available: http://tools.ietf.org/html/draft-dhody-pce-cso-enabled-path-computation-06

[67] B. K. F. Hu and H. Cankaya, "SPRING OpenFlow interworking requirements," Internet Engineering Task Force, Internet Draft, Sep. 2014. [Online]. Available: https://tools.ietf.org/html/draft-khc-spring-openflow-interworking-req-00

[68] E. P. S. Kim, J. Park, and L. Contreras, "SPRING use cases for software-defined networking," Internet Engineering Task Force, Internet Draft, Jul. 2014. [Online]. Available: http://tools.ietf.org/html/draft-kim-spring-use-cases-00

[69] D. Ceccarelli, L. Fang, Y. Lee, and D. Lopez, "Framework for abstraction and control of transport networks," Internet Engineering Task Force, Internet Draft, Feb. 2014. [Online]. Available: https://xml.resource.org/html/draft-ceccarelli-actn-framework-01

[70] M. Boucadair and C. Jacquenet, "Software-defined networking: A perspective from within a service provider environment," Internet Engineering Task Force, Internet Draft, Mar. 2014. [Online]. Available: https://tools.ietf.org/html/rfc7149

[71] E. Haleplidis *et al.*, "SDN layers and architecture terminology," Internet Engineering Task Force, Internet Draft, Aug. 2014. [Online]. Available: http://tools.ietf.org/html/draft-haleplidis-sdnrg-layer-terminology-07

[72] C. B. L. Contreras and D. Lopez, "Cooperating layered architecture for SDN," Internet Engineering Task Force, Internet Draft, Aug. 2014. [Online]. Available: http://tools.ietf.org/html/draft-contreras-sdnrg-layered-sdn-00

[73] Y. Cheng and C. Zhou, "Framework of signalling for SDN—Working document," International Telecommunication Union Telecommunication Standardization Sector (ITU-T), Tech. Rep., work item: Q.Suppl.-SDN. [Online]. Available: http://www.itu.int

[74] International Telecommunication Union Telecommunication Standardization Sector (ITU-T), "Scenarios and signalling requirements for software-defined BAN (SBAN)—Working document," ITU-T SG 11, Tech. Rep., work item: Q.SBAN, Jul. 2014. [Online]. Available: http://www.itu.int

[75] International Telecommunication Union Telecommunication Standardization Sector (ITU-T), "Framework of software-defined networking," Tech. Rep., Recommendation ITU-T Y.3300, Jun. 2014. [Online]. Available: http://www.itu.int/rec/T-REC-Y.3300-201406-I/en

[76] Broadband Forum, "High level requirements and framework for SDN in telecommunication broadband networksBroadband Forum SD-313," Tech. Rep., Sep. 2014. [Online]. Available: http://www.broadband-forum.org/technical/technicalwip.php

[77] Optical Interworking Forum (OIF), "Requirements on transport networks in SDN architectures," Tech. Rep., Sep. 2013. [Online]. Available: http://www.oiforum.com/public/documents/OIF_Carrier_WG_Requirements_on_Transport_Networks_in_SDN_Architectures_Sept2013.pdf

[78] Open Data Center Alliance, "Software-defined networking Rev. 2.0," Tech. Rep., 2014. [Online]. Available: http://www.opendatacenteralliance.org/docs/software_defined_networking_master_usage_model_rev2.pdf

[79] European Telecommunications Standards Institute (ETSI), "Network functions virtualization (NFV); architectural framework v1.1.1ETSI GS NFV 002," Tech. Rep., Oct. 2013. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf

[80] Automatic Terminal Information Service (ATIS), "Operational opportunities and challenges of SDN/NFV programmable infrastructure," Tech. Rep. ATIS-I-0000044, Oct. 2014. [Online]. Available: https://www.atis.org/docstore/product.aspx?id=28143

[81] D. Tennenhouse, J. Smith, W. Sincoskie, D. Wetherall, and G. Minden, "A survey of active network research," *IEEE Commun. Mag.*, vol. 35, no. 1, pp. 80–86, Jan. 1997.

[82] A. Lazar, K.-S. Lim, and F. Marconcini, "Realizing a foundation for programmability of ATM networks with the binding architecture," *IEEE J. Sel. Areas Commun.*, vol. 14, no. 7, pp. 1214–1227, Sep. 1996.

[83] A. Lazar, "Programming telecommunication networks," *IEEE Network*, vol. 11, no. 5, pp. 8–18, Sep. 1997.

[84] D. Sheinbein and R. P. Weber, "800 service using SPC network capability," *Bell Syst. Tech. J.*, vol. 61, no. 7, pp. 1737–1744, Sep. 1982.

[85] M. Caesar *et al.*, "Design and implementation of a routing control platform," in *Proc. 2nd Conf. Symp. Netw. Syst. Design Implement.*, 2005, vol. 2, pp. 15–28.

[86] J. Biswas *et al.*, "The IEEE P1520 standards initiative for programmable network interfaces," *Commun. Mag.*, vol. 36, no. 10, pp. 64–70, Oct. 1998.

[87] B. Schwartz *et al.*, "Smart packets for active networks," in *Proc. IEEE 2nd Conf. Open Architect. Netw. Programm.*, Mar. 1999, pp. 90–97.

[88] D. Wetherall, J. V. Guttag, and D. Tennenhouse, "Ants: A toolkit for building and dynamically deploying network protocols," in *Proc. IEEE Conf. Open Architect. Netw. Programm.*, Apr. 1998, pp. 117–129.

[89] D. Alexander *et al.*, "The switchware active network architecture," *IEEE Network*, vol. 12, no. 3, pp. 29–36, May 1998.

[90] K. Calvert, S. Bhattacharjee, E. Zegura, and J. Sterbenz, "Directions in active networks," *IEEE Commun. Mag.*, vol. 36, no. 10, pp. 72–78, Oct. 1998.

[91] T. Wolf and J. Turner, "Design issues for high performance active routers," in *Proc. Int. Zurich Seminar Broadband Commun.*, 2000, pp. 199–205.

[92] S. da Silva, Y. Yemini, and D. Florissi, "The NetScript active network system," *IEEE J. Sel. Areas Commun.*, vol. 19, no. 3, pp. 538–551, Mar. 2001.

[93] D. L. Tennenhouse and D. J. Wetherall, "Towards an active network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 5, pp. 81–94, Oct. 2007.

[94] P. Newman *et al.*, "Ipsilon's general switch management protocol specification version 1.1," Internet Engineering Task Force, RFC 1987 (Informational), Aug. 1996, updated by RFC 2297. [Online]. Available: http://www.ietf.org/rfc/rfc1987.txt

[95] A. Doria and K. Sundell, "General switch management protocol (GSMP) applicability," Internet Engineering Task Force, RFC 3294 (Informational), Jun. 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3294.txt

[96] J. Van der Merwe, S. Rooney, I. Leslie, and S. Crosby, "The tempest-a practical framework for network programmability," *IEEE Network*, vol. 12, no. 3, pp. 20–28, May 1998.

[97] T. Lakshman, T. Nandagopal, R. Ramjee, K. Sabnani, and T. Woo, "The SoftRouter architecture," in *Proc. 3rd ACM Workshop Hot Topics Netw.*, San Diego, CA, USA, Nov. 2004, vol. 2004, pp. 1–6.

[98] A. Greenberg *et al.*, "A clean slate 4D approach to network control and management," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 5, pp. 41–54, Oct. 2005.

[99] J. Van der Merwe *et al.*, "Dynamic connectivity management with an intelligent route service control point," in *Proc. SIGCOMM Workshop Internet Netw. Manage.*, 2006, pp. 29–34.

[100] M. Casado *et al.*, "SANE: A protection architecture for enterprise networks," in *Proc. 15th Conf. USENIX Security Symp.*, 2006, vol. 15, Article 10.

[101] M. Casado *et al.*, "Ethane: Taking control of the enterprise," in *Proc. Conf. Appl. Technol. Architect. Protocols Comput. Commun.*, 2007, DOI: 10.1145/1282380.1282382.

[102] M. Macedonia and D. Brutzman, "Mbone provides audio and video across the internet," *Computer*, vol. 27, no. 4, pp. 30–36, 1994.

[103] R. Fink and R. Hinden, "6bone (IPv6 Testing Address Allocation) Phaseout," Internet Engineering Task Force, RFC 3701 (Informational), Mar. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3701.txt

[104] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient overlay networks," *SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, pp. 131–145, Oct. 2001.

[105] B. Chun *et al.*, "Planetlab: An overlay testbed for broad-coverage services," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 3, pp. 3–12, Jul. 2003.

[106] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the internet impasse through virtualization," *Computer*, vol. 38, no. 4, pp. 34–41, Apr. 2005.

[107] L. Peterson *et al.*, "Geni design principles," *Computer*, vol. 39, no. 9, pp. 102–105, Sep. 2006.

[108] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In VINI veritas: Realistic and controlled network experimentation," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 4, pp. 3–14, Aug. 2006.

[109] B. Pfaff *et al.*, "Extending networking into the virtualization layer," in *Proc. Workshop Hot Topics Netw.*, 2009, pp. 1–6.

[110] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proc. 9th ACM SIGCOMM Workshop Hot Topics Netw.*, 2010, pp. 19:1–19:6.

[111] R. Sherwood *et al.*, "Can the production network be the testbed?" in *Proc. 9th USENIX Conf. Oper. Syst. Design Implement.*, 2010, pp. 1–6.

[112] T. Koponen *et al.*, "Network virtualization in multi-tenant datacenters," in *Proc. 11th USENIX Symp. Netw. Syst. Design Implement.*, Apr. 2014, pp. 203–216.

[113] V. Bollapragada, C. Murphy, and R. White, *Inside Cisco IOS Software Architecture*, 1st ed. Indianapolis, IN, USA: Cisco Press, Jul. 2000.

[114] Juniper Networks, "Junos OS architecture overview," 2012. [Online]. Available: http://www.juniper.net/techpubs/en_US/junos12.1/topics/concept/junos-software-architecture.html

[115] Extreme Networks, "ExtremeXOS operating system, version 15.4," 2014. [Online]. Available: http://learn.extremenetworks.com/rs/extreme/images/EXOS-DS.pdf

[116] Alcatel-Lucent, "SR OS," 2014. [Online]. Available: http://www3. alcatel-lucent.com/products/sros/

[117] U. Krishnaswamy *et al.*, "ONOS: An open source distributed SDN OS," 2013. [Online]. Available: http://www.slideshare.net/umeshkrishnaswamy/open-network-operating-system

[118] A. T. Campbell, I. Katzela, K. Miki, and J. Vicente, "Open signaling for ATM, internet and mobile networks (OPENSIG'98)," *SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 1, pp. 97–108, Jan. 1999.

[119] R. Sherwood *et al.*, "Carving research slices out of your production networks with OpenFlow," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 129–130, Jan. 2010.

[120] H. Song, J. Gong, J. Song, and J. Yu, "Protocol oblivious forwarding (POF)," 2013. [Online]. Available: http://www.poforwarding.org/

[121] Open Networking Foundation (ONF), "Charter: Forwarding abstractions working group," Apr. 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/working-groups/charter-forwarding-abstractions.pdf

[122] Centec Networks, "V350—Centec open SDN platform," 2013. [Online]. Available: http://www.valleytalk.org/wp-content/uploads/2013/04/Centec-Open-SDN-Platform.pdf

[123] NECProgrammable, "Flow UNIVERGE PF5820," 2013. [Online]. Available: http://www.nec.com/en/global/prod/pflow/images_documents/ProgrammableFlow_Switch_PF5820.pdf

[124] NoviFlow, "NoviSwitch 1248 high performance OpenFlow switch," 2013. [Online]. Available: http://205.236.122.20/gestion/NoviSwitch1248Datasheet.pdf

[125] HP, "8200 ZL switch series," 2013. [Online]. Available: http://h17007.www1.hp.com/us/en/networking/products/switches/HP_8200_zl_Switch_Series/

[126] Arista Networks, "7150 Series," 2013. [Online]. Available: http://www.aristanetworks.com/media/system/pdf/Datasheets/7150S_Datasheet.pdf

[127] Extreme Networks, "Blackdiamond x8," 2013. [Online]. Available: http://www.extremenetworks.com/libraries/products/DSBDX_1832.pdf

[128] Huawei Technologies Co., Ltd., "Cx600 metro services platform," 2013. [Online]. Available: http://www.huawei.com/ucmf/groups/public/documents/attachments/hw_132369.pdf

[129] Juniper Networks, "Ex9200 Ethernet switch," 2013. [Online]. Available: http://www.juniper.net/us/en/local/pdf/datasheets/1000432-en.pdf

[130] I. Yokneam, "EZchip announces OpenFlow 1.1 implementations on its NP-4 100-gigabit network processor," 2011. [Online]. Available: http://www.ezchip.com/pr_110713.htm

[131] BROCADE, "MLX Series," 2013. [Online]. Available: http://www.brocade.com/products/all/routers/product-details/netiron-mlx-series/system-options.page

[132] IBM, "System networking RackSwitch G8264," 2013. [Online]. Available: http://www-03.ibm.com/systems/networking/switches/rack/g8264/

[133] NEC, "ProgrammableFlow family of products," 2013. [Online]. Available: http://www.necam.com/SDN/

[134] Pica8, "3920," 2013. [Online]. Available: http://www.pica8.org/documents/pica8-datasheet-64x10gbe-p3780-p3920.pdf

[135] Plexxi, "Switch 1," 2013. [Online]. Available: http://www.plexxi.com/wp-content/themes/plexxi/assets/pdf/Plexxi_Switch_1_Datasheet_Dec_2012.pdf

[136] Centec Networks, "v330 OpenFlow switch reference design," 2013. [Online]. Available: http://www.centecnetworks.com/en/SolutionList.asp?ID=42

[137] Cyan, Inc., "Z-Series," 2013. [Online]. Available: http://www.cyaninc.com/en/our-solutions/z-series/

[138] Juniper Networks, Inc., "Contrail virtual router," 2013. [Online]. Available: https://github.com/Juniper/contrail-vrouter

[139] FlowForwarding, "LINC-Switch," 2013. [Online]. Available: http://www.flowforwarding.org/

[140] K. Rutka, K. Kaplita, S. Narayan, and S. Bailey, "LINC switch," 2013. [Online]. Available: http://www.opennetsummit.org/pdf/2013/research_track/poster_papers/ons2013-final36.pdf

[141] E. L. Fernandes and C. E. Rothenberg, "OpenFlow 1.3 software switch, SBRC'2014," 2014. [Online]. Available: https://github.com/CPqD/ofsoftswitch13

[142] Open vSwitch, 2013. [Online]. Available: http://vswitch.org/

[143] OpenFlow Community, "Switching reference system," 2009. [Online]. Available: http://www.openflow.org/wp/downloads/

[144] Y. Mundada, R. Sherwood, and N. Feamster, "An OpenFlow switch element for click," *Proc. Symp. Click Modular Router*, 2009, 1 p. [Online]. Available: http://www.cc.gatech.edu/?yogeshm3/click_symposium2009.pdf

[145] Big Switch Networks, "Project floodlight," 2013. [Online]. Available: http://www.projectfloodlight.org/

[146] Y. Yiakoumis, J. Schulz-Zander, and J. Zhu, "Pantou: OpenFlow 1.0 for OpenWRT," 2011. [Online]. Available: http://www.openflow.org/wk/index.php/Open_Flow1.0_forOpenWRT

[147] A. Weissberger, "VMware's network virtualization poses huge threat to data center switch fabric vendors," 2013. [Online]. Available: http://viodi.com/2013/05/06/vmwares-network-virtualization-poses-huge-threat-to-data-center-switch-fabric-vendors/

[148] S. Shenker, "Stanford Seminar—Software-defined networking at the crossroads," Jun. 2013. [Online]. Available: http://www.youtube.com/watch?v=WabdXYzCAOU

[149] M. Casado, "OpenStack and network virtualization," Apr. 2013. [Online]. Available: http://blogs.vmware.com/vmware/2013/04/openstack-and-network-virtualization.html

[150] Pica8 Open Networking, "Pica8's OS for Open Switches," 2013. [Online]. Available: http://www.pica8.org/open-switching/open-switching-overview.php

[151] Open Network Install Environment (ONIE), 2013. [Online]. Available: http://onie.org/

[152] T. Kato *et al.*, "Case study of applying SPLE to development of network switch products," in *Proc. 17th Int. Softw. Product Line Conf.*, 2013, pp. 198–207.

[153] B. Pfaff and B. Davie, "The Open vSwitch database management protocol," Internet Engineering Task Force, RFC 7047 (Informational), Dec. 2013. [Online]. Available: http://www.ietf.org/rfc/rfc7047.txt

[154] M. Smith *et al.*, "OpFlex control protocol," Internet Engineering Task Force," Internet Draft, Apr. 2014. [Online]. Available: http://tools.ietf.org/html/draft-smith-opflex-00

[155] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "OpenState: Programming platform-independent stateful OpenFlow applications inside the switch," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 44–51, Apr. 2014.

[156] M. Sune, V. Alvarez, T. Jungel, U. Toseef, and K. Pentikousis, "An OpenFlow implementation for network processors," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, 2 pp.

[157] D. Parniewicz *et al.*, "Design and implementation of an OpenFlow hardware abstraction layer," in *Proc. ACM SIGCOMM Workshop Distrib. Cloud Comput.*, 2014, pp. 71–76.

[158] B. Belter *et al.*, "Hardware abstraction layer as an SDN-enabler for non-OpenFlow network equipment," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, 6 pp.

[159] B. Belter *et al.*, "Programmable abstraction of datapath," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, pp. 7–12.

[160] R. G. Clegg *et al.*, "Pushing software defined networking to the access," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, pp. 1–6.

[161] V. Fuentes *et al.*, "Integrating complex legacy systems under OpenFlow control: The DOCSIS use case," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, 6 pp.

[162] T. J. Bittman, G. J. Weiss, M. A. Margevicius, and P. Dawson, "Magic quadrant for x86 server virtualization infrastructure," Gartner, Tech. Rep., Jun. 2013.

[163] D. W. Cearley, D. Scott, J. Skorupa, and T. J. Bittman, "Top 10 technology trends, 2013: Cloud computing and hybrid IT drive future IT models," Feb. 2013. [Online]. Available: http://www.gartnersummit.com/Gartnertop_10_technology_trends_2012_37716.pdf

[164] C. Peng, M. Kim, Z. Zhang, and H. Lei, "VDN: Virtual machine image distribution network for cloud data centers," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 181–189.

[165] Z. Zhang *et al.*, "VMThunder: Fast provisioning of large-scale virtual machine clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 12, pp. 3328–3338, Dec. 2014.

[166] R. Sherwood *et al.*, "FlowVisor: A network virtualization layer," Deutsche Telekom Inc. R&D Lab, Stanford, Nicira Networks, Tech. Rep., 2009.

[167] S. Azodolmolky *et al.*, "Optical FlowVisor: An OpenFlow-based optical network virtualization approach," in *Proc. Nat. Fiber Optic Eng. Conf.*, Mar. 2012.

[168] D. A. Drutskoy, "Software-defined network virtualization with FlowN," Ph.D. dissertation, Dept. Comput. Sci., Princeton Univ., Princeton, NJ, USA, Jun. 2012.

[169] A. Al-Shabibi *et al.*, "OpenVirteX: A Network Hypervisor," 2014. [Online]. Available: http://ovx.onlab.us/wp-content/uploads/2014/04/ovx-ons14. pdf

[170] A. Al-Shabibi *et al.*, "OpenVirteX: Make your virtual SDNs programmable," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 25–30.

[171] S. Racherla *et al.*, *Implementing IBM Software Defined Network for Virtual Environments*. Durham, NC, USA: IBM RedBooks, May 2014.

[172] C. Li *et al.*, "Software defined environments: An introduction," *IBM J. Res. Develop.*, vol. 58, no. 2, pp. 1–11, Mar. 2014.

[173] A. Gudipati, L. E. Li, and S. Katti, "Radiovisor: A slicing plane for radio access networks," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 237–238.

[174] H. Yamanaka, E. Kawai, S. Ishii, and S. Shimojo, "AutoVFlow: Autonomous virtualization for wide-area OpenFlow networks," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, 6 pp.

[175] Berlin Institute for Software Defined Networks (BISDN) GmbH, "The eXtensible OpenFlow Datapath Daemon (xdpd) bringing innovation into the fast path," 2014. [Online]. Available: http://xdpd.org/

[176] R. Doriguzzi-Corin, E. Salvadori, M. Gerola, M. Sune, and H. Woesner, "A datapath-centric virtualization mechanism for OpenFlow networks," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, pp. 1–6.

[177] T. Szyrkowiec *et al.*, "Demonstration of SDN based optical network virtualization and multidomain service orchestration," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, 2 pp.

[178] D. Depaoli, R. Doriguzzi-Corin, M. Gerola, and E. Salvadori, "Demonstrating a distributed and version-agnostic OpenFlow slicing mechanism," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, 2 pp.

[179] Z. Bozakov and P. Papadimitriou, "AutoSlice: Automated and scalable slicing for software-defined networks," in *Proc. ACM Conf. CoNEXT Student Workshop*, 2012, pp. 3–4.

[180] D. Drutskoy, E. Keller, and J. Rexford, "Scalable network virtualization in software-defined networks," *IEEE Internet Comput.*, vol. 17, no. 2, pp. 20–27, Mar./Apr. 2013.

[181] X. Jin, J. Rexford, and D. Walker, "Incremental update for a compositional SDN hypervisor," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 187–192.

[182] S. Ghorbani and B. Godfrey, "Towards correct network virtualization," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 109–114.

[183] Juniper Networks, "Opencontrail," 2013. [Online]. Available: http://opencontrail.org/

[184] HP, "SDN controller architecture," Tech. Rep., Sep. 2013.

[185] K. Phemius, M. Bouet, and J. Leguay, "DISCO: Distributed multi-domain SDN controllers," Aug. 2013. [Online]. Available: http://arxiv.org/abs/1308.6138

[186] D. Erickson, "The Beacon OpenFlow controller," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 13–18.

[187] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proc. 2nd USENIX Conf. Hot Topics Manage. Internet Cloud Enterprise Netw. Services*, 2012, p. 10.

[188] Z. Cai, A. L. Cox, and T. S. E. Ng, "Maestro: A system for scalable OpenFlow control," Rice Univ., Houston, TX, USA, Tech. Rep., 2011.

[189] Project Floodlight, "Floodlight," 2012. [Online]. Available: http://floodlight.openflowhub.org/

[190] Y. Takamiya and N. Karanatsios, "Trema OpenFlow controller framework," 2012. [Online]. Available: https://github.com/trema/trema

[191] Nippon Telegraph and Telephone Corporation, "RYU network operating system," 2012. [Online]. Available: http://osrg.github.com/ryu/

[192] M. Banikazemi, D. Olshefski, A. Shaikh, J. Tracey, and G. Wang, "Meridian: An SDN platform for cloud network services," *IEEE Commun. Mag.*, vol. 51, no. 2, pp. 120–127, Feb. 2013.

[193] NEC, "Award-winning software-defined networking NEC ProgrammableFlow networking suite," Sep. 2013. [Online]. Available: http://www.necam.com/docs/?id=67c33426-0a2b-4b87-9a7a-d3cecc14d26a

[194] S. Shin *et al.*, "Rosemary: A robust, secure, high-performance network operating system," in *Proc. 21st ACM Conf. Comput. Commun. Security*, Scottsdale, AZ, USA, Nov. 2014, pp. 78–89.

[195] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow," in *Proc. Internet Netw. Manage. Conf. Res. Enterprise Netw.*, 2010, p. 3.

[196] M. Monaco, O. Michel, and E. Keller, "Applying operating system principles to SDN controller design," in *Proc. 12th ACM Workshop Hot Topics Netw.*, College Park, MD, USA, Nov. 2013, DOI: 10.1145/2535771.2535789.

[197] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi, "Participatory networking: An API for application control of SDNs," in *Proc. ACM SIGCOMM Conf.*, 2013, pp. 327–338.

[198] F. Botelho, A. Bessani, F. M. V. Ramos, and P. Ferreira, "On the design of practical fault-tolerant SDN controllers," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, 6 pp.

[199] S. Matsumoto, S. Hitz, and A. Perrig, "Fleet: Defending SDNs from malicious administrators," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 103–108.

[200] A. Bierman, M. Bjorklund, K. Watsen, and R. Fernando, "RESTCONF protocol," Internet Engineering Task Force, Internet Draft, Jul. 2014. [Online]. Available: http://tools.ietf.org/html/draft-ietf-netconf-restconf-01

[201] P. Porras *et al.*, "A security enforcement kernel for OpenFlow networks," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 121–126.

[202] L. Richardson and S. Ruby, *RESTful Web Services*. Sebastopol, CA, USA: O'Reilly Media, 2008.

[203] T. L. Hinrichs, N. S. Gude, M. Casado, J. C. Mitchell, and S. Shenker, "Practical declarative network management," in *Proc. 1st ACM Workshop Res. Enterprise Netw.*, 2009, DOI: 10.1145/1592681.1592683.

[204] N. Foster *et al.*, "Frenetic: A network programming language," *SIGPLAN Notes*, vol. 46, no. 9, pp. 279–291, 2011.

[205] C. Monsanto, N. Foster, R. Harrison, and D. Walker, "A compiler and run-time system for network programming languages," *SIGPLAN Notes*, vol. 47, no. 1, pp. 217–230, Jan. 2012.

[206] A. Singla and B. Rijsman, "Contrail architecture," Juniper Networks, Tech. Rep., 2013.

[207] Open Networking Foundation (ONF), "OpenFlow management and configuration protocol (OF-Config 1.1.1)," Mar. 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-config-1-1-1.pdf

[208] D. Harrington, R. Presuhn, and B. Wijnen, "An architecture for describing simple network management protocol (SNMP) management frameworks," Internet Engineering Task Force, Dec. 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3411.txt

[209] H. Yin *et al.*, "SDNi: A message exchange protocol for software defined networks (SDNS) across multiple domains," Internet Engineering Task Force, Internet Draft, Jun. 2012. [Online]. Available: http://tools.ietf.org/id/draft-yin-sdn-sdni-00.txt

[210] M. Garcia, A. Bessani, I. Gashi, N. Neves, and R. Obelheiro, "Analysis of operating system diversity for intrusion tolerance," *Softw., Practice Experience*, vol. 44, no. 6, pp. 735–770, 2014.

[211] Z. Wang, T. Tsou, J. Huang, X. Shi, and X. Yin, "Analysis of comparisons between OpenFlow and ForCES," Internet Engineering Task Force, Internet Draft, Dec. 2011. [Online]. Available: http://tools.ietf.org/id/draft-wang-forces-compare-openflow-forces-00.txt

[212] K. Ogawa, W. M. Wang, E. Haleplidis, and J. H. Salim, "ForCES Intra-NE high availability," Internet Engineering Task Force, Internet Draft, Oct. 2013. [Online]. Available: http://tools.ietf.org/id/draft-ietf-forces-ceha-08.txt

[213] F. A. Botelho, F. M. V. Ramos, D. Kreutz, and A. N. Bessani, "On the feasibility of a consistent and fault-tolerant data store for SDNs," in *Proc. 2nd Eur. Workshop Softw. Defined Netw.*, 2013, pp. 38–43.

[214] S. Vinoski, "Advanced message queuing protocol," *IEEE Internet Comput.*, vol. 10, no. 6, pp. 87–89, Nov. 2006.

[215] M. Canini, P. Kuznetsov, D. Levin, and S. Schmid, "Software transactional networking: Concurrent and consistent policy composition," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2013, DOI: 10.1145/2491185.2491200.

[216] A. Ghodsi, "Distributed k-ary system: Algorithms for distributed Hash tables," Ph.D. dissertation, Dept. Electr., Comp. Software Syst., Royal Inst. Technol. (KTH), Stockholm, Sweden, Oct. 2006.

[217] W. Stallings, "Software-defined networks and OpenFlow," *Internet Protocol J.*, vol. 16, no. 1, pp. 1–6, 2013.

[218] Open Networking Foundation (ONF), "SDN architecture," Jun. 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf

[219] R. Hand and E. Keller, "ClosedFlow: OpenFlow-like control over proprietary devices," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 7–12.

[220] M. Jarschel, T. Zinner, T. Hoßfeld, P. Tran-Gia, and W. Kellerer, "Interfaces, attributes, use cases: A compass for SDN," *IEEE Commun. Mag.*, vol. 52, no. 6, pp. 210–217, Jun. 2014.

[221] E. Mannie, "Generalized multi-protocol label switching (GMPLS) architecture," Internet Engineering Task Force, RFC 3945 (Proposed Standard), Oct. 2004, updated by RFC 6002. [Online]. Available: http://www.ietf.org/rfc/rfc3945.txt

[222] K. Pentikousis, Y. Wang, and W. Hu, "MobileFlow: Toward software-defined mobile networks," *IEEE Commun. Mag.*, vol. 51, no. 7, pp. 44–53, Jul. 2013.

[223] A. Voellmy and P. Hudak, "Nettle: Taking the sting out of programming network routers," in *Proc. 13th Int. Conf. Practical Aspects Declarative Lang.*, 2011, pp. 235–249.

[224] A. Voellmy, H. Kim, and N. Feamster, "Procera: A language for high-level reactive network control," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 43–48.

[225] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing software-defined networks," in *Proc. 10th USENIX Conf. Netw. Syst. Design Implement.*, 2013, pp. 1–14.

[226] C. J. Anderson *et al.*, "NetKAT: Semantic foundations for networks," *SIGPLAN Notes*, vol. 49, no. 1, pp. 113–126, Jan. 2014.

[227] S. Narayana, J. Rexford, and D. Walker, "Compiling path queries in software-defined networks," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 181–186.

[228] A. A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Elasticon: An elastic distributed SDN controller," in *Proc. 10th ACM/IEEE Symp. Architect. Netw. Commun. Syst.*, 2014, pp. 17–28.

[229] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 19–24.

[230] D. Saikia, "MuL OpenFlow Controller," 2013. [Online]. Available: http://sourceforge.net/projects/mul/

[231] M. McCauley, "POX," 2012. [Online]. Available: http://www.noxrepo.org/

[232] H. Shimonishi and S. Ishii, "Virtualized network infrastructure using OpenFlow," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp. Workshops*, 2010, pp. 74–79.

[233] G. Appenzeller, "SNAC," 2011. [Online]. Available: http://www. openflowhub.org/display/Snac

[234] B. Casemore, "SDN controller ecosystems critical to market success," 2012. [Online]. Available: http://nerdtwilight.wordpress.com/2012/06/05/sdn-controller-ecosystems-critical-to-market-success/

[235] R. Kwan and C. Leung, "A survey of scheduling and interference mitigation in LTE," *J. Electr. Comput. Eng.*, vol. 2010, Jan. 2010, DOI: 10.1155/2010/273486.

[236] A. Gudipati, D. Perry, L. E. Li, and S. Katti, "SoftRAN: Software defined radio access network," in *Proc. 2nd Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 25–30.

[237] J. Dix, "Clarifying the role of software-defined networking northbound APIs," May 2013. [Online]. Available: http://www.networkworld.com/news/2013/050213-sherwood-269366.html

[238] I. Guis, "The SDN gold rush to the northbound API," Nov. 2012. [Online]. Available: http://www.sdncentral.com/technology/the-sdn-gold-rush-to-the-northbound-api/2012/11/

[239] B. Salisbury, "The northbound API—A big little problem," 2012.

[240] G. Ferro, "Northbound API, southbound API, east/north LAN navigation in an OpenFlow world and an SDN compass," Aug. 2012.

[241] B. Casemore, "Northbound API: The standardization debate," Sep. 2012. [Online]. Available: http://nerdtwilight.wordpress.com/2012/09/18/northbound-api-the-standardization-debate/

[242] I. Pepelnjak, "SDN controller northbound API is the crucial missing piece," Sep. 2012. [Online]. Available: http://blog.ioshints.info/2012/09/sdn-controller-northbound-api-is.html

[243] S. Johnson, "A primer on northbound APIs: Their role in a software-defined network," Dec. 2012. [Online]. Available: http://searchsdn.techtarget.com/feature/A-primer-on-northbound-APIs-Their-role-in-a-software-defined-network

[244] R. G. Little, "ONF to standardize northbound API for SDN applications?" Oct. 2013. [Online]. Available: http://searchsdn.techtarget.com/news/2240206604/ONF-to-standardize-northbound-API-for-SDN-applications

[245] Austin Common Standards Revision Group, "POSIX," 2014. [Online]. Available: http://standards.ieee.org/develop/wg/POSIX.html

[246] M. Yu, A. Wundsam, and M. Raju, "NOSIX: A lightweight portability layer for the SDN OS," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 28–35, Apr. 2014.

[247] R. Chua, "OpenFlow northbound API: A new Olympic sport," 2012. [Online]. Available: http://www.sdncentral.com/sdn-blog/openflow-northbound-api-olympics/2012/07/

[248] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, "Modular SDN programming with pyretic," *USENIX Mag.*, vol. 38, no. 5, Oct. 2013.

[249] K.-K. Yap, T.-Y. Huang, B. Dodson, M. S. Lam, and N. McKeown, "Towards software-friendly networks," in *Proc. 1st ACM Asia-Pacific Workshop Syst.*, 2010, pp. 49–54.

[250] S. Gutz, A. Story, C. Schlesinger, and N. Foster, "Splendid isolation: A slice abstraction for software-defined networks," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 79–84.

[251] D. Turull, M. Hidell, and P. Sjödin, "Evaluating OpenFlow in libnetvirt," in *Proc. 8th Swedish Nat. Comput. Netw. Workshop*, Oct. 2012, pp. 1–5.

[252] Quantum Community, "OpenStack networking ('Quantum')," 2012.

[253] Small Cell Forum, "Femto APIs," 2013. [Online]. Available: http://www.smallcellforum.org/developers/

[254] M. Guzdial, "Education: Paving the way for computational thinking," *Commun. ACM*, vol. 51, no. 8, pp. 25–27, Aug. 2008.

[255] M. S. Farooq, S. A. Khan, F. Ahmad, S. Islam, and A. Abid, "An evaluation framework and comparative analysis of the widely used first programming languages," *PLoS ONE*, vol. 9, no. 2, 2014, DOI: 10.1371/journal.pone.0088941.

[256] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi, "Hierarchical policies for software defined networks," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 37–42.

[257] T. Nelson, A. D. Ferguson, M. J. Scheer, and S. Krishnamurthi, "Tierless programming and reasoning for software-defined networks," in *Proc. 11th USENIX Symp. Netw. Syst. Design Implement.*, Apr. 2014, pp. 519–531.

[258] N. P. Katta, J. Rexford, and D. Walker, "Logic programming for software-defined networks," in *Proc. ACM SIGPLAN Workshop Cross-Model Lang. Design Implement.*, 2012, pp. 1–3.

[259] S. Shin *et al.*, "FRESCO: Modular composable security services for software-defined networks," Internet Society NDSS, Feb. 2013.

[260] S. Son, S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Model checking invariant security properties in OpenFlow," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2013, pp. 1974–1979.

[261] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: Traffic matrix estimator for OpenFlow networks," in *Proc. 11th Int. Conf. Passive Active Meas.*, 2010, pp. 201–210.

[262] M. Reitblatt, M. Canini, A. Guha, and N. Foster, "FatTire: Declarative fault tolerance for software defined networks," in *Proc. 2nd Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 109–114.

[263] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak, "Maple: Simplifying SDN programming using algorithmic policies," in *Proc. ACM SIGCOMM Conf.*, 2013, pp. 87–98.

[264] R. Soule, S. Basu, R. Kleinberg, E. G. Sirer, and N. Foster, "Managing the network with Merlin," in *Proc. 12th ACM Workshop Hot Topics Netw.*, Nov. 2013, DOI: 10.1145/2535771.2535792.

[265] C. Jasson Casey, A. Sutton, G. Dos Reis, and A. Sprintson, "Eliminating network protocol vulnerabilities through abstraction and systems language design," Nov. 2013. [Online]. Available: http://arxiv.org/abs/1311.3336

[266] X. Wen *et al.*, "Compiling minimum incremental update for modular SDN languages," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 193–198.

[267] P. Pereini, M. Kuzniar, and D. Kostic, "OpenFlow needs you! A call for a discussion about a cleaner OpenFlow API," in *Proc. 2nd Eur. Workshop Softw. Defined Netw.*, Oct. 2013, pp. 44–49.

[268] F. Facca *et al.*, "NetIDE: First steps towards an integrated development environment for portable network apps," in *Proc. 2nd Eur. Workshop Softw. Defined Netw.*, Oct. 2013, pp. 105–110.

[269] E. Reinecke, "Mapping the future of software-defined networking," 2014. [Online]. Available: http://goo.gl/fQCvRF

[270] M. Scharf *et al.*, "Dynamic VPN optimization by ALTO guidance," in *Proc. 2nd Eur. Workshop Softw. Defined Netw.*, Oct. 2013, pp. 13–18.

[271] M. Stiemerling, S. Kiesel, S. Previdi, and M. Scharf, "ALTO deployment

considerations," Internet Engineering Task Force, Internet Draft, Feb. 2014. [Online]. Available: http://tools.ietf.org/html/draft-ietf-alto-deployments-09

[272] R. Alimi, R. Penno, and Y. Yang, "ALTO protocol," Internet Engineering Task Force, Internet Draft, Mar. 2014. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-alto-protocol/

[273] N. Handigol *et al.*, "Aster*x: Load-balancing web traffic over wide-area networks," 2009.

[274] B. Heller *et al.*, "ElasticTree: Saving energy in data center networks," in *Proc. 7th USENIX Conf. Netw. Syst. Design Implement.*, 2010, p. 17.

[275] M. S. Seddiki *et al.*, "FlowQoS: QoS for the rest of us," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 207–208.

[276] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. 7th USENIX Conf. Netw. Syst. Design Implement.*, 2010, p. 19.

[277] C. Macapuna, C. Rothenberg, and M. Magalhaes, "In-packet bloom filter based data center networking with distributed OpenFlow controllers," in *Proc. IEEE GLOBECOM Workshops*, 2010, pp. 584–588.

[278] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine grained traffic engineering for data centers," in *Proc. 7th Conf. Emerging Netw. Exp. Technol.*, 2011, pp. 8:1–8:12.

[279] H. Egilmez, S. Dane, K. Bagci, and A. Tekalp, "OpenQoS: An Open-Flow controller design for multimedia delivery with end-to-end quality of service over software-defined networks," in *Proc. Asia-Pacific Signal Inf. Process. Assoc. Annu. Summit Conf.*, 2012, pp. 1–8.

[280] A. Sgambelluri, A. Giorgetti, F. Cugini, F. Paolucci, and P. Castoldi, "OpenFlow-based segment protection in Ethernet networks," *IEEE/OSA J. Opt. Commun. Netw.*, vol. 5, no. 9, pp. 1066–1075, Sep. 2013.

[281] M. Bari, S. Chowdhury, R. Ahmed, and R. Boutaba, "PolicyCop: An autonomic QoS policy enforcement framework for software defined networks," in *Proc. IEEE SDN Future Netw. Services*, Nov. 2013, DOI: 10.1109/SDN4FNS.2013.6702548.

[282] K. Nagaraj and S. Katti, "ProCel: Smart traffic handling for a scalable software EPC," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 43–48.

[283] P. Xiong and H. Hacigümüs, "Pronto: A software-defined networking based system for performance management of analytical queries on distributed data stores," *PVLDB*, vol. 7, no. 13, pp. 1661–1664, 2014.

[284] P. Xiong, H. Hacigumus, and J. F. Naughton, "A software-defined networking based approach for performance management of analytical queries on distributed data stores," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2014, pp. 955–966.

[285] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and R. Johari, "Plug-n-serve: Load-balancing web traffic using OpenFlow," 2009.

[286] M. Veiga Neves, C. De Rose, K. Katrinis, and H. Franke, "Pythia: Faster big data in motion through predictive software-defined network optimization at runtime," in *Proc. IEEE 28th Int. Parallel Distrib. Process. Symp.*, May 2014, pp. 82–90.

[287] K. Jeong, J. Kim, and Y.-T. Kim, "QoS-aware network operating system for software defined networking with generalized OpenFlows," in *Proc. IEEE Netw. Oper. Manage. Symp.*, Apr. 2012, pp. 1167–1174.

[288] S. Sharma *et al.*, "Implementing quality of service for the software defined networking enabled future internet," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, pp. 49–54.

[289] W. Kim *et al.*, "Automated and scalable QoS control for network convergence," in *Proc. Internet Netw. Manage. Conf. Res. Enterprise Netw.*, 2010, p. 1.

[290] A. Ishimori, F. Farias, E. Cerqueira, and A. Abelem, "Control of multiple packet schedulers for improving QoS on OpenFlow/SDN networking," in *Proc. 2nd Eur. Workshop Softw. Defined Netw.*, Oct. 2013, pp. 81–86.

[291] D. Palma *et al.*, "The QueuePusher: Enabling queue management in OpenFlow," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, pp. 125–126.

[292] Z. A. Qazi *et al.*, "SIMPLE-fying middlebox policy enforcement using SDN," in *Proc. Conf. Appl. Technol. Architect. Protocols Comput. Commun.*, 2013, pp. 27–38.

[293] P. Skoldstrom and B. C. Sanchez, "Virtual aggregation using SDN," in *Proc. 2nd Eur. Workshop Softw. Defined Netw.*, 2013, pp. 56–61.

[294] J. Schulz-Zander, N. Sarrar, and S. Schmid, "AeroFlux: A near-sighted controller architecture for software-defined wireless networks," presented at the Open Netw. Summit, Santa Clara, CA, USA, 2014.

[295] J. Schulz-Zander, N. Sarrar, and S. Schmid, "Towards a scalable and near-sighted control plane architecture for WiFi SDNs," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 217–218.

[296] H. Ali-Ahmad *et al.*, "CROWD: An SDN approach for densenets," in *Proc. 2nd Eur. Workshop Softw. Defined Netw.*, Oct. 2013, pp. 25–31.

[297] J. Vestin *et al.*, "CloudMAC: Towards software defined WLANs," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 16, no. 4, pp. 42–45, Feb. 2013.

[298] A. Dawson, M. K. Marina, and F. J. Garcia, "On the benefits of RAN virtualisation in C-RAN based mobile networks," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, 6 pp.

[299] Y. Yamasaki, Y. Miyamoto, J. Yamato, H. Goto, and H. Sone, "Flexible access management system for campus VLAN based on OpenFlow," in *Proc. IEEE/IPSJ 11th Int. Symp. Appl. Internet*, 2011, pp. 347–351.

[300] J. Schulz-Zander *et al.*, "Programmatic orchestration of WiFi networks," in *Proc. USENIX Annu. Tech. Conf.*, Jun. 2014, pp. 347–358.

[301] M. Yang *et al.*, "OpenRAN: A software-defined ran architecture via virtualization," in *Proc. ACM SIGCOMM Conf.*, 2013, pp. 549–550.

[302] K.-K. Yap *et al.*, "OpenRoads: Empowering research in mobile networks," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 125–126, Jan. 2010.

[303] V. Chandrasekhar, J. Andrews, and A. Gatherer, "Femtocell networks: A survey," *IEEE Commun. Mag.*, vol. 46, no. 9, pp. 59–67, Sep. 2008.

[304] Y. Yu, C. Qian, and X. Li, "Distributed and collaborative traffic monitoring in software defined networks," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 85–90.

[305] S. Shirali-Shahreza and Y. Ganjali, "FleXam: Flexible sampling extension for monitoring and security applications in OpenFlow," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 167–168.

[306] C. Yu *et al.*, "FlowSense: Monitoring network utilization with zero measurement cost," in *Proc. 14th Int. Conf. Passive Active Meas.*, 2013, pp. 31–41.

[307] L. Jose, M. Yu, and J. Rexford, "Online measurement of large traffic aggregates on commodity switches," in *Proc. 11th USENIX Conf. Hot Topics Manage. Internet Cloud Enterprise Netw. Services*, 2011, p. 13.

[308] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers, "OpenNetMon: Network monitoring in openflow software-defined networks," in *Proc. IEEE Netw. Oper. Manage. Symp.*, 2014, DOI: 10.1109/NOMS.2014.6838228.

[309] J. Suh, T. Kwon, C. Dixon, W. Felter, and J. Carter, "OpenSample: A low-latency, sampling-based measurement platform for commodity SDN," in *Proc. IEEE 34th Int. Conf. Distrib. Comput. Syst.*, Jun. 2014, pp. 228–237.

[310] sFlow.org Forum, 2012. [Online]. Available: http://www. sflow.org/

[311] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with OpenSketch," in *Proc. 10th USENIX Conf. Netw. Syst. Design Implement.*, 2013, pp. 29–42.

[312] C. Argyropoulos, D. Kalogeras, G. Androulidakis, and V. Maglaris, "PaFloMon—A slice aware passive flow monitoring framework for OpenFlow enabled experimental facilities," in *Proc. Eur. Workshop Softw. Defined Netw.*, 2012, pp. 97–102.

[313] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "PayLess: A low cost network monitoring framework for software defined networks," in *Proc. 14th IEEE/IFIP Netw. Oper. Manage. Symp.*, 2014, DOI: 10.1109/NOMS.2014.6838227.

[314] G. Wang, T. E. Ng, and A. Shaikh, "Programming your network at run-time for big data applications," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 103–108.

[315] T. Benson, A. Akella, A. Shaikh, and S. Sahu, "Cloudnaas: A cloud networking platform for enterprise applications," in *Proc. 2nd ACM Symp. Cloud Comput.*, 2011, pp. 8:1–8:13.

[316] A. Das *et al.*, "Transparent and flexible network management for big data processing in the cloud," in *Proc. 5th USENIX Conf. Hot Topics Cloud Comput.*, San Jose, CA, USA, 2013, pp. 1–6.

[317] A. Arefin, V. K. Singh, G. Jiang, Y. Zhang, and C. Lumezanu, "Diagnosing data center behavior flow by flow," in *Proc. IEEE 33rd Int. Conf. Distrib. Comput. Syst.*, Jul. 2013, DOI: 10.1109/ICDCS.2013.18.

[318] E. Keller, S. Ghorbani, M. Caesar, and J. Rexford, "Live migration of an entire network (and its hosts)," in *Proc. 11th ACM Workshop Hot Topics Netw.*, 2012, pp. 109–114.

[319] R. Raghavendra, J. Lobo, and K.-W. Lee, "Dynamic graph query primitives for SDN-based cloudnetwork management," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 97–102.

[320] M. Ghobadi, "TCP adaptation framework in data centers," Ph.D. dissertation, Grad.

Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, 2013.

[321] R. Hand, M. Ton, and E. Keller, "Active security," in *Proc. 12th ACM Workshop Hot Topics Netw.*, Nov. 2013, 17 pp.

[322] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "AVANT-GUARD: Scalable and vigilant switch flow management in software-defined networks," in *Proc. ACM Conf. Comput. Commun. Security*, 2013, pp. 413–424.

[323] S. Shin and G. Gu, "CloudWatcher: Network security monitoring using OpenFlow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?)," in *Proc. 20th IEEE Int. Conf. Netw. Protocols*, 2012, DOI: 10.1109/ICNP.2012.6459946.

[324] E. Tantar, M. Palattella, T. Avanesov, M. Kantor, and T. Engel, "Cognition: A tool for reinforcing security in software defined networks," in *EVOLVE—A Bridge between Probability, Set Oriented Numerics, Evolutionary Computation V*, vol. 288, A.-A. Tantar *et al.*, Ed. Berlin, Germany: Springer-Verlag, 2014, pp. 61–78.

[325] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS flooding attack detection using NOX/OpenFlow," in *Proc. IEEE 35th Conf. Local Comput. Netw.*, Oct. 2010, pp. 408–415.

[326] G. Stabler, A. Rosen, S. Goasguen, and K.-C. Wang, "Elastic IP and security groups implementation using OpenFlow," in *Proc. 6th Int. Workshop Virtualization Technol. Distrib. Comput. Date*, 2012, pp. 53–60.

[327] J. Matias, J. Garay, A. Mendiola, N. Toledo, and E. Jacob, "FlowNAC: Flow-based network access control," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, 6 pp.

[328] K. Wang, Y. Qi, B. Yang, Y. Xue, and J. Li, "LiveSec: Towards effective security management in large-scale production networks," in *Proc. 32nd Int. Conf. Distrib. Comput. Syst. Workshops*, Jun. 2012, pp. 451–460.

[329] A. Sapio *et al.*, "MAPPER: A mobile application personal policy enforcement router for enterprise networks," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, 2 pp.

[330] Y. Wang, Y. Zhang, V. Singh, C. Lumezanu, and G. Jiang, "NetFuse: Short-circuiting traffic surges in the cloud," in *Proc. IEEE Int. Conf. Commun.*, 2013, DOI: 10.1109/ICC.2013.6655095.

[331] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "OpenFlow random host mutation: Transparent moving target defense using software defined networking," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 127–132.

[332] J. R. Ballard, I. Rae, and A. Akella, "Extensible and scalable network monitoring using OpenSAFE," in *Proc. Internet Netw. Manage. Conf. Res. Enterprise Netw.*, 2010, p. 8.

[333] A. Zaalouk, R. Khondoker, R. Marx, and K. Bayarou, "OrchSec: An orchestrator-based architecture for enhancing network-security using network monitoring and SDN control functions," in *Proc. IEEE Netw. Oper. Manage. Symp.*, May 2014, DOI: 10.1109/NOMS.2014.6838409.

[334] Flow-RT group, "sFlow-RT," 2014. [Online]. Available: http://www. inmon.com/products/sFlow-RT.php

[335] D. Kotani, K. Suzuki, and H. Shimonishi, "A design and implementation of OpenFlow controller handling ip multicast with fast tree switching," in *Proc. IEEE/IPSJ 12th Int. Symp. Appl. Internet*, 2012, pp. 60–67.

[336] K. Giotis, G. Androulidakis, and V. Maglaris, "Leveraging SDN for efficient anomaly detection and mitigation on legacy networks," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, 6 pp.

[337] G. Yao, J. Bi, and P. Xiao, "Source address validation solution with OpenFlow/NOX architecture," in *Proc. 19th IEEE Int. Conf. Netw. Protocols*, 2011, pp. 7–12.

[338] R. Wang, D. Butnariu, and J. Rexford, "OpenFlow-based server load balancing gone wild," in *Proc. 11th USENIX Conf. Hot Topics Manage. Internet Cloud Enterprise Netw. Services*, 2011, p. 12.

[339] X.-N. Nguyen, D. Saucez, C. Barakat, and T. Turletti, "Optimizing rules placement in OpenFlow networks: Trading routing for better efficiency," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 127–132.

[340] A. Schwabe and H. Karl, "Using MAC addresses as efficient routing labels in data centers," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 115–120.

[341] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in SDN-OpenFlow networks," *Comput. Netw.*, vol. 71, pp. 1–30, Oct. 2014.

[342] H. Ballani, P. Francis, T. Cao, and J. Wang, "Making routers last longer with Viaggre," in *Proc. 6th USENIX Symp. Netw. Syst. Design Implement.*, 2009, pp. 453–466.

[343] D. Meyer, L. Zhang, and K. Fall, "Report from the IAB Workshop on Routing and Addressing," Internet Engineering Task Force, RFC 4984 (Informational), Sep. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4984.txt

[344] M. Jarschel, F. Wamser, T. Hohn, T. Zinner, and P. Tran-Gia, "SDN-based application-aware networking on the example of YouTube video streaming," in *Proc. 2nd Eur. Workshop Softw. Defined Netw.*, Oct. 2013, pp. 87–92.

[345] T. G. Edwards and W. Belkin, "Using SDN to facilitate precisely timed actions on real-time data streams," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 55–60.

[346] H. Kumar, H. H. Gharakheili, and V. Sivaraman, "User control of quality of experience in home networks using SDN," in *Proc. IEEE Int. Conf. Adv. Netw. Telecommun. Syst.*, 2013, DOI: 10.1109/ANTS.2013.6802847.

[347] L. Li, Z. Mao, and J. Rexford, "Toward software-defined cellular networks," in *Proc. Eur. Workshop Softw. Defined Netw.*, 2012, pp. 7–12.

[348] X. Jin, L. Erran Li, L. Vanbever, and J. Rexford, "SoftCell: Scalable and flexible cellular core network architecture," in *Proc. 9th Int. Conf. Emerging Netw. Exp. Technol.*, 2013, pp. 163–174.

[349] P. Dely, A. Kassler, and N. Bayer, "OpenFlow for wireless mesh networks," in *Proc. 20th Int. Conf. Comput. Commun. Netw.*, 2011, pp. 1–6.

[350] M. J. Yang, S. Y. Lim, H. J. Park, and N. H. Park, "Solving the data overload: Device-to-device bearer control architecture for cellular data offloading," *IEEE Veh. Technol. Mag.*, vol. 8, no. 1, pp. 31–39, Mar. 2013.

[351] K.-K. Yap *et al.*, "Blueprint for introducing innovation into wireless mobile networks," in *Proc. 2nd ACM SIGCOMM Workshop Virtualized Infrastructure Syst. Architect.*, 2010, pp. 25–32.

[352] M. Bansal, J. Mehlman, S. Katti, and P. Levis, "Openradio: A programmable wireless dataplane," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 109–114.

[353] S. Sundaresan *et al.*, "Broadband internet performance: A view from the gateway," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 134–145, Aug. 2011.

[354] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting traffic anomaly detection using software defined networking," in *Proc. 14th Int. Conf. Recent Adv. Intrusion Detection*, 2011, pp. 161–180.

[355] P. Wette and H. Karl, "Which flows are hiding behind my wildcard rule?: Adding packet sampling to OpenFlow," in *Proc. ACM SIGCOMM Conf.*, 2013, pp. 541–542.

[356] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, "Theory and practice of bloom filters for distributed systems," *IEEE Commun. Surv. Tut.*, vol. 14, no. 1, pp. 131–155, 2012.

[357] J. Kempf *et al.*, "Scalable fault management for OpenFlow," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2012, pp. 6606–6610.

[358] G. Bianchi, M. Bonola, G. Picierro, S. Pontarelli, and M. Monaci, "StreaMon: A software-defined monitoring platform," in *Proc. 26th ITC*, Sep. 2014, pp. 1–6.

[359] D. Kreutz, F. M. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 55–60.

[360] K. Kant, "Data center evolution: A tutorial on state of the art, issues, challenges," *Comput. Netw.*, vol. 53, no. 17, pp. 2939–2965, 2009.

[361] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68–73, Dec. 2008.

[362] M. Bari *et al.*, "Data center network virtualization: A survey," *IEEE Commun. Surv. Tut.*, vol. 15, no. 2, pp. 909–928, 2013.

[363] A. Krishnamurthy, S. P. Chandrabose, and A. Gember-Jacobson, "Pratyaastha: An efficient elastic distributed SDN control plane," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 133–138.

[364] P. Calyam *et al.*, "Leveraging OpenFlow for resource placement of virtual desktop cloud applications," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage.*, 2013, pp. 311–319.

[365] J. Parraga, "Avior," 2013. [Online]. Available: http://openflow.marist. edu/avior

[366] GlobalNOC, "OESS—Open Exchange Software Suite," 2013. [Online]. Available: http://globalnoc.iu.edu/sdn/oess.html.

[367] C. Duckett, "Software defined networking: HP has an App store for that," 2013. [Online]. Available: http://www.zdnet.com/software-defined-networking-hp-has-an-app-store-for-that-7000021365/

[368] Hewlett-Packard Company (HP), "SDN app store," 2013. [Online]. Available: http://h17007.www1.hp.com/us/en/networking/solutions/technology/sdn/devcenter/#sdnAppstore

[369] B. H. Sigelman *et al.*, "Dapper, a large-scale distributed systems tracing infrastructure," Google, Inc., Tech. Rep., 2010.

[370] L. Layman *et al.*, "Debugging revisited: Toward understanding the debugging needs of contemporary software developers," in

Proc. ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas., Oct. 2013, pp. 383–392.

[371] U. Erlingsson, M. Peinado, S. Peter, M. Budiu, and G. Mainar-Ruiz, "Fay: Extensible distributed tracing from kernels to clusters," *ACM Trans. Comput. Syst.*, vol. 30, no. 4, pp. 13:1–13:35, Nov. 2012.

[372] S. Tomaselli and O. Landsiedel, "Towards lightweight logging and replay of embedded, distributed systems," in *Proc. Workshop Architecting Safety Collaborative Mobile Systems/32nd Int. Conf. Comput. Safety Reliab. Security*, M. Roy, Ed., Toulouse, France, Sep. 2013, 7 pp.

[373] J. Tan, S. Kavulya, R. Gandhi, and P. Narasimhan, "Visual, log-based causal tracing for performance debugging of map reduce systems," in *Proc. IEEE 30th Int. Conf. Distrib. Comput. Syst.*, Jun. 2010, pp. 795–806.

[374] R. Fonseca, G. Porter, R. H. Katz, S. Shenker, and I. Stoica, "X-trace: A pervasive network tracing framework," in *Proc. 4th USENIX Conf. Netw. Syst. Design Implement.*, 2007, p. 20.

[375] V. Trivedi, "Software development: Debugging and testing," in *How to Speak Technology*. New York, NY, USA: Apress, 2014, pp. 89–95.

[376] A. Anand and A. Akella, "Netreplay: A new network primitive," *SIGMETRICS Perform. Eval. Rev.*, vol. 37, no. 3, pp. 14–19, Jan. 2010.

[377] Y. Zhuang *et al.*, "Netcheck: Network diagnoses from blackbox traces," in *Proc. 11th USENIX Symp. Netw. Syst. Design Implement.*, Apr. 2014, pp. 115–128.

[378] N. Handigol, B. Heller, V. Jeyakumar, D. Maziéres, and N. McKeown, "Where is the debugger for my software-defined network?" in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 55–60.

[379] A. Wundsam, D. Levin, S. Seetharaman, and A. Feldmann, "OFRewind: Enabling record and replay troubleshooting for networks," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, 2011, p. 29.

[380] M. Canini, D. Venzano, P. Perešíni, D. Kostić, and J. Rexford, "A NICE way to test OpenFlow applications," in *Proc. 9th USENIX Conf. Netw. Syst. Design Implement.*, Apr. 2012, pp. 127–140.

[381] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, "OFLOPS: An open framework for OpenFlow switch evaluation," in *Proc. 13th Int. Conf. Passive Active Meas.*, 2012, pp. 85–95.

[382] E. Al-Shaer and S. Al-Haj, "FlowChecker: Configuration analysis and verification of federated OpenFlow infrastructures," in *Proc. 3rd ACM Workshop Assurable Usable Security Config.*, 2010, pp. 37–44.

[383] A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey, "VeriFlow: Verifying network-wide invariants in real time," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 49–54.

[384] M. Kuzniar, M. Canini, and D. Kostic, "OFTEN testing OpenFlow networks," in *Proc. 1st Eur. Workshop Softw. Defined Netw.*, 2012, pp. 54–60.

[385] G. Altekar and I. Stoica, "Focus replay debugging effort on the control plane," Electr. Eng. Comput. Sci., Univ. California Berkeley, Berkeley, CA, USA, Tech. Rep., May 2010.

[386] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, and N. McKeown, "I know what your packet did last hop: Using packet histories to troubleshoot networks," in *Proc. 11th USENIX Symp. Netw. Syst. Design Implement.*, Apr. 2014, pp. 71–85.

[387] Y. Zhang, N. Beheshti, and R. Manghirmalani, "NetRevert: Rollback recovery in SDN," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 231–232.

[388] J. S. Plank, M. Beck, G. Kingsley, and K. Li, "Libckpt: Transparent checkpointing under unix," in *Proc. USENIX Tech. Conf.*, 1995, p. 18.

[389] J. S. Plank, Y. Chen, K. Li, M. Beck, and G. Kingsley, "Memory exclusion: Optimizing the performance of checkpointing systems," *Softw., Practice Exp.*, vol. 29, no. 2, pp. 125–142, Feb. 1999.

[390] N. Ruchansky and D. Proserpio, "A (not) nice way to verify the OpenFlow switch specification: Formal modelling of the OpenFlow switch using alloy," in *Proc. ACM SIGCOMM Conf.*, 2013, pp. 527–528.

[391] H. Zeng *et al.*, "Libra: Divide and conquer to verify forwarding tables in huge networks," in *Proc. 11th USENIX Symp. Netw. Syst. Design Implement.*, Apr. 2014, pp. 87–99.

[392] R. Sherwood and K.-K. Yap, "Cbench controller benchmarker," 2011. [Online]. Available: http://www.openflow.org/wk/index.php/Oflops

[393] M. Jarschel, F. Lehrieder, Z. Magyari, and R. Pries, "A flexible OpenFlow-controller benchmark," in *Proc. Eur. Workshop Softw. Defined Netw.*, 2012, pp. 48–53.

[394] Veryx Technologies, "PktBlaster SDN controller test." [Online]. Available: http://sdn.veryxtech.com/

[395] C. Rotsos, G. Antichi, M. Bruyere, P. Owezarski, and A. W. Moore, "An open testing framework for next-generation OpenFlow switches," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, pp. 127–128.

[396] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao, "FLOWGUARD: Building robust firewalls for software-defined networks," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 97–102.

[397] T. Ball *et al.*, "Vericon: Towards verifying controller programs in software-defined networks," *SIGPLAN Notes*, vol. 49, no. 6, pp. 282–293, Jun. 2014.

[398] P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: Static checking for networks," in *Proc. 9th USENIX Conf. Netw. Syst. Design Implement.*, 2012, p. 9.

[399] H. Mai *et al.*, "Debugging the data plane with anteater," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 290–301, Aug. 2011.

[400] P. Kazemian *et al.*, "Real time network policy checking using header space analysis," in *Proc. 10th USENIX Conf. Netw. Syst. Design Implement.*, 2013, pp. 99–112.

[401] R. Beckett *et al.*, "An assertion language for debugging SDN applications," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 91–96.

[402] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," in *Proc. 8th Int. Conf. Emerging Netw. Exp. Technol.*, 2012, pp. 253–264.

[403] V. Antonenko and R. Smelyanskiy, "Global network modelling based on Mininet approach," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 145–146.

[404] J. Teixeira *et al.*, "Datacenter in a box: Test your SDN cloud-datacenter controller at home," in *Proc. 2nd Eur. Workshop Softw. Defined Netw.*, Oct. 2013, pp. 99–104.

[405] P. Wette *et al.*, "Maxinet: Distributed emulation of software-defined networks," in *Proc. Netw. Conf.*, Jun. 2014, pp. 1–9.

[406] A. Roy, M. Bari, M. Zhani, R. Ahmed, and R. Boutaba, "Design and management of DOT: A distributed OpenFlow testbed," in *Proc. Netw. Oper. Manage. Symp.*, May 2014, pp. 1–9.

[407] A. Carter *et al.*, "CityFlow: OpenFlow city experiment—Linking infrastructure and applications," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, pp. 129–130.

[408] ns-3 project, "OpenFlow switch support," 2013. [Online]. Available: http://www.nsnam.org/docs/release/3.13/models/html/openflow-switch.html

[409] J. Sommers *et al.*, "Efficient network-wide flow record generation," in *Proc. IEEE INFOCOM*, 2011, pp. 2363–2371.

[410] ucb-sts, "STS—SDN troubleshooting simulator," 2013. [Online]. Available: http://ucb-sts.github.io/sts/

[411] H. Zhang *et al.*, "Enabling layer 2 pathlet tracing through context encoding in software-defined networking," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 169–174.

[412] K. Agarwal, E. Rozner, C. Dixon, and J. Carter, "SDN traceroute: Tracing SDN forwarding without changing network behavior," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 145–150.

[413] S. K. Fayaz and V. Sekar, "Testing stateful and dynamic data planes with FlowTest," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 79–84.

[414] M. Shahbaz *et al.*, "Architecture for an open source network tester," in *Proc. ACM/IEEE Symp. Architect. Netw. Commun. Syst.*, Oct. 2013, pp. 123–124.

[415] N. Laurent, S. Vissicchio, and M. Canini, "SDLoad: An extensible framework for SDN workload generation," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 215–216.

[416] M. Gupta, J. Sommers, and P. Barford, "Fast, accurate simulation for SDN prototyping," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 31–36.

[417] R. R. Fontes, A. L. C. Oliveira, T. R. Pinheiro, P. N. Sampaio, and R. A. Figueira, "Authoring of OpenFlow networks with visual network description (SDN version)," in *Proc. Summer Comput. Simul. Conf.*, Monterey, CA, USA, 2014, pp. 22:1–22:6.

[418] A. R. Curtis *et al.*, "DevoFlow: Scaling flow management for high-performance networks," *Comput. Commun. Rev.*, vol. 41, no. 4, pp. 254–265, Aug. 2011.

[419] C. J. Casey, A. Sutton, and A. Sprintson, "tinyNBI: Distilling an API from essential OpenFlow abstractions," 2014. [Online]. Available: http://arxiv.org/abs/1403.6644

[420] L. Ogrodowczyk *et al.*, "Hardware abstraction layer for non-OpenFlow capable devices," in *Proc. 30th Trans Eur. Res. Edu. Netw. Conf.*, 2014, 8 pp.

[421] A. Vidal, C. E. Rothenberg, and F. L. Verdi, "The libfluid OpenFlow driver implementation," in *Proc. 32nd Brazilian Symp. Comp. Netw. (SBRC)*, May 2014, pp. 1029–1036.

[422] M. Appelman and M. D. Boer, "Performance analysis of open-flow hardware," Univ. Amsterdam, Amsterdam, The Netherlands, Tech. Rep., Feb. 2012.

[423] K. Kannan and S. Banerjee, "Compact TCAM: Flow entry compaction in TCAM for power aware SDN," in *Distributed Computing and Networking*, vol. 7730, D. Frey, M. Raynal, S. Sarkar, R. Shyamasundar, and P. Sinha, Eds. Berlin, Germany: Springer-Verlag, 2013, pp. 439–444.

[424] J. Liao, "SDN system performance," Jun. 2012. [Online]. Available: http://pica8.org/blogs/?p=201

[425] B. Agrawal and T. Sherwood, "Modeling TCAM power for next generation network devices," in *Proc. IEEE Int. Symp. Performance Anal. Syst. Softw.*, 2006, pp. 120–129.

[426] B. Owens, "OpenFlow switching performance: Not all TCAM is created equal," Feb. 2013. [Online]. Available: http://packetpushers. net/openflow-switching-performance-not-all-tcam-is-created-equal/

[427] B. Salisbury, "TCAMs and OpenFlow—What every SDN practitioner must know," Jul. 2012. [Online]. Available: http://www.sdncentral.com/technology/sdn-openflow-tcam-need-to-know/2012/07/

[428] W. Braun and M. Menth, "Wildcard compression of inter-domain routing tables for OpenFlow-based software-defined networking," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, 6 pp.

[429] K. Agarwal, C. Dixon, E. Rozner, and J. Carter, "Shadow MACs: Scalable label-switching for commodity Ethernet," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 157–162.

[430] R. Rudell and A. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 6, no. 5, pp. 727–750, Sep. 1987.

[431] R. Bifulco and M. Dusi, "Reactive logic in software-defined networking: Accounting for the limitations of the switches," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, 6 pp.

[432] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter, "Past: Scalable Ethernet for data centers," in *Proc. 8th Int. Conf. Emerging Netw. Exp. Technol.*, 2012, pp. 49–60.

[433] M. Kobayashi *et al.*, "Maturing of OpenFlow and software-defined networking through deployments," *Comput. Netw.*, vol. 61, *Special Issue on Future Internet Testbeds—Part I*, pp. 151–175, 2014.

[434] J. C. Mogul and P. Congdon, "Hey, you darned counters! Get off my asic!" in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 25–30.

[435] P. Bosshart *et al.*, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in *Proc. ACM SIGCOMM Conf.*, 2013, pp. 99–110.

[436] O. Ferkouss *et al.*, "A 100 gig network processor platform for openflow," in *Proc. 7th Int. Conf. Netw. Service Manage.*, 2011, pp. 1–4.

[437] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown, "Implementing an OpenFlow switch on the NetFPGA platform," in *Proc. 4th ACM/IEEE Symp. Architect. Netw. Commun. Syst.*, 2008, DOI: 10.1145/1477942.1477944.

[438] G. Memon *et al.*, "FlashFlow: A GPU-based fully programmable OpenFlow switch," Univ. Oregon, Eugene, OR, USA, Tech. Rep., 2013.

[439] Y. Luo, P. Cascon, E. Murray, and J. Ortega, "Accelerating Open-Flow switching with network processors," in *Proc. 5th ACM/IEEE Symp. Architect. Netw. Commun. Syst.*, 2009, pp. 70–71.

[440] A. Rostami, T. Jungel, A. Koepsel, H. Woesner, and A. Wolisz, "Oran: OpenFlow routers for academic networks," in *Proc. IEEE 13th Int. Conf. High Performance Switching Routing*, 2012, pp. 216–222.

[441] G. Pongracz, L. Molnar, and Z. Kis, "Removing roadblocks from SDN: OpenFlow software switch performance on Intel DPDK," in *Proc. 2nd Eur. Workshop Softw. Defined Netw.*, Oct. 2013, pp. 62–67.

[442] B. Stephens, "Designing scalable networks for future large datacenters," Ph.D. dissertation, Dept. Comp., Rice Univ., Houston, TX, USA, May 2012.

[443] Y. Li, D. Zhang, K. Huang, D. He, and W. Long, "A memory-efficient parallel routing lookup model with fast updates," *Comput. Commun.*, vol. 38, pp. 60–71, Feb. 2014.

[444] N. Katta, J. Rexford, and D. Walker, "Infinite CacheFlow in software-defined networks," Princeton Schl. Eng. Appl. Sci., Princeton, NJ, USA, Tech. Rep., Oct. 2013.

[445] H. Mekky, F. Hao, S. Mukherjee, Z.-L. Zhang, and T. Lakshman, "Application-aware data plane processing in SDN," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 13–18.

[446] M. Moshref, A. Bhargava, A. Gupta, M. Yu, and R. Govindan, "Flow-level state transition as a new switch primitive for SDN," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 61–66.

[447] B. Yan, Y. Xu, H. Xing, K. Xi, and H. J. Chao, "CAB: A reactive wildcard rule caching system for software-defined networks," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 163–168.

[448] S. McGillicuddy, "XPliant Ethernet chip sets new standard for programmability," 2014. [Online]. Available: http://goo.gl/xE8K9B

[449] Intel Processors, "Software defined networking and software-based services with Intel Processors," 2012. [Online]. Available: http://www.intel.com/content/dam/doc/white-paper/communications-ia-software-defined-networking-paper.pdf

[450] Intel Corporation, "Intel data plane development kit," 2014. [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/guides/intel-dpdk-getting-started-guide.pdf

[451] A. Sivaraman, K. Winstein, S. Subramanian, and H. Balakrishnan, "No silver bullet: Extending SDN to the data plane," in *Proc. 12th ACM Workshop Hot Topics Netw.*, Nov. 2013, DOI: 10.1145/2535771.2535796.

[452] S. Zhou, W. Jiang, and V. Prasanna, "A programmable and scalable OpenFlow switch using heterogeneous SoC platforms," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 239–240.

[453] S. Hauger *et al.*, "Packet processing at 100 Gbps and beyond—Challenges and perspectives," in *Proc. ITG Symp. Photon. Netw.*, May 2009, pp. 1–10.

[454] G. Bianchi *et al.*, "MAClets: Active MAC protocols over hard-coded devices," in *Proc. 8th Int. Conf. Emerging Netw. Exp. Technol.*, 2012, pp. 229–240.

[455] I. Tinnirello *et al.*, "Wireless MAC processors: Programming MAC protocols on commodity hardware," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 1269–1277.

[456] P. Bosshart *et al.*, "Programming protocol-independent packet processors," 2013. [Online]. Available: http://arxiv.org/abs/1312.1719

[457] S. Schmid and J. Suomela, "Exploiting locality in distributed SDN control," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 121–126.

[458] P. Berde, "ONOS: Towards an open, distributed SDN OS," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, DOI: 10.1145/2620728.2620744.

[459] D. M. Volpano, X. Sun, and G. G. Xie, "Towards systematic detection and resolution of network control conflicts," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 67–72.

[460] S. Azodolmolky, P. Wieder, and R. Yahyapour, "Performance evaluation of a scalable software-defined networking deployment," in *Proc. 2nd Eur. Workshop Softw. Defined Netw.*, Oct. 2013, pp. 68–74.

[461] S. H. Park, B. Lee, J. Shin, and S. Yang, "A high-performance IO engine for SDN controllers," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, 2 pp.

[462] Y. Zhang, S. Natarajan, X. Huang, N. Beheshti, and R. Manghirmalani, "A compressive method for maintaining forwarding states in SDN controller," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 139–144.

[463] S. H. Park *et al.*, "RAON: Recursive abstraction of OpenFlow networks," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, 2 pp.

[464] A. AuYoung *et al.*, "Corybantic: Towards the modular composition of SDN control programs," in *Proc. 12th ACM Workshop Hot Topics Netw.*, Nov. 2013, DOI: 10.1145/2535771.2535795.

[465] P. Sun *et al.*, "A network-state management service," in *Proc. ACM Conf. SIGCOMM*, 2014, pp. 563–574.

[466] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized? State distribution trade-offs in software defined networks," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, DOI: 10.1145/2342441.2342443.

[467] D. Peleg, *Distributed Computing: A Locality-Sensitive Approach*. Philadelphia, PA, USA: SIAM, 2000.

[468] F. J. Ros and P. M. Ruiz, "Five nines of southbound reliability in software-defined networks," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 31–36.

[469] V. Daniel Philip and Y. Gourhant, "Cross-control: A scalable multi-topology fault restoration mechanism using logically centralized controllers," in *Proc. IEEE 15th Int. Conf. High Performance Switching Routing*, Jul. 2014, pp. 57–63.

[470] M. Borokhovich, L. Schiff, and S. Schmid, "Provable data plane connectivity with local fast failover: Introducing OpenFlow graph algorithms," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 121–126.

[471] D. Kreutz, A. Casimiro, and M. Pasin, "A trustworthy and resilient event broker for monitoring cloud infrastructures," in *Proc. 12th IFIP WG 6.1 DAIS*, 2012, pp. 87–95.

[472] K. Tesink, "Definitions of managed objects for the synchronous optical network/synchronous digital hierarchy (SONET/SDH) interface type," Internet Engineering Task Force, Internet Draft, Sep. 2003. [Online]. Available: http://tools.ietf.org/html/rfc3592

[473] R. Prasanna, "BIP: Billing information protocol," Internet Engineering Task Force, Internet Draft, Dec. 2002. [Online]. Available: http://tools.ietf.org/html/draft-prasanna-bip-00

[474] G. Swallow, A. Fulignoli, M. Vigoureux, S. Boutros, and D. Ward, "MPLS fault management operations, administration, maintenance (OAM)," Internet Engineering Task Force, RFC 6427 (Proposed Standard), Nov. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6427.txt

[475] D. Whiting, R. Housley, and N. Ferguson, "Counter with CBC-MAC (CCM)," Internet Engineering Task Force, RFC 3610 (Informational), Sep. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3610.txt

[476] M. Desai and T. Nandagopal, "Coping with link failures in centralized control plane architectures," in *Proc. 2nd Int. Conf. Commun. Syst. Netw.*, 2010, DOI: 10.1109/COMSNETS.2010.5431977.

[477] H. Kim *et al.*, "Coronet: Fault tolerance for software defined networks," in *Proc. 20th IEEE Int. Conf. Network Protocols*, Oct. 2012, DOI: 10.1109/ICNP.2012.6459938.

[478] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "OpenFlow: Meeting carrier-grade recovery requirements," *Comput. Commun.*, vol. 36, no. 6, pp. 656–665, Mar. 2013.

[479] A. Panda, C. Scott, A. Ghodsi, T. Koponen, and S. Shenker, "Cap for networks," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 91–96.

[480] M. Kuźniar, P. Perešíni, N. Vasić, M. Canini, and D. Kostić, "Automatic failure recovery for software-defined networks," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 159–160.

[481] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 7–12.

[482] R. Ramos, M. Martinello, and C. Esteve Rothenberg, "SlickFlow: Resilient source routing in data center networks unlocked by OpenFlow," in *Proc. IEEE 38th Conf. Local Comput. Netw.*, Oct. 2013, pp. 606–613.

[483] J. T. Araújo, R. Landa, R. G. Clegg, and G. Pavlou, "Software-defined network support for transport resilience," in *Proc. IEEE Netw. Oper. Manage. Symp.*, 2014, DOI: 10.1109/NOMS.2014.6838243.

[484] E. Brewer, "Pushing the cap: Strategies for consistency and availability," *Computer*, vol. 45, no. 2, pp. 23–29, Feb. 2012.

[485] D. Katz and D. Ward, "Bidirectional forwarding detection (BFD)," Internet Engineering Task Force, RFC 5880 (Proposed Standard), Jun. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5880.txt

[486] N. L. M. van Adrichem, B. J. van Asten, and F. A. Kuipers, "Fast recovery in software-defined networks," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, 6 pp.

[487] N. M. Sahri and K. Okamura, "Fast failover mechanism for software defined networking: Openflow based," in *Proc. 9th Int. Conf. Future Internet Technol.*, 2014, pp. 16:1–16:2.

[488] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. 10th ACM SIGCOMM Conf. Internet Meas.*, 2010, pp. 267–280.

[489] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with difane," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 351–362, Aug. 2010.

[490] M. F. Bari *et al.*, "Dynamic controller provisioning in software defined networks," in *Proc. 9th Int. Conf. Netw. Service Manage.*, 2013, pp. 18–25.

[491] V. Yazici, M. O. Sunay, and A. O. Ercan, "Controlling a software-defined network via distributed controllers," in *Proc. Conf. Implement. Future Media Internet Towards New Horizons*, Oct. 2012, pp. 16–22.

[492] M. Jarschel *et al.*, "Modeling and performance evaluation of an OpenFlow architecture," in *Proc. 23rd Int. Teletraffic Congr.*, Sep. 2011, pp. 1–7.

[493] A. Bianco, R. Birke, L. Giraudo, and M. Palacin, "OpenFlow switching: Data plane performance," in *Proc. IEEE Int. Conf. Commun.*, May 2010, DOI: 10.1109/ICC.2010.5502016.

[494] R. Pries, M. Jarschel, and S. Goll, "On the usability of OpenFlow in data center environments," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2012, pp. 5533–5537.

[495] J. Hwang, K. K. Ramakrishnan, and T. Wood, "NetVM: High performance and flexible networking using virtualization on commodity platforms," in *Proc. 11th USENIX Symp. Netw. Syst. Design Implement.*, Apr. 2014, pp. 445–458.

[496] Y. Dong, Z. Yu, and G. Rose, "SR-IOV networking in Xen: Architecture, design and implementation," in *Proc. 1st Conf. I/O Virtualization*, 2008, p. 10.

[497] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 7–12.

[498] S. Azodolmolky *et al.*, "An analytical model for software defined networking: A network calculus-based approach," in *Proc. IEEE GlobeCom*, Oct. 2013, DOI: 10.1109/GLOCOM.2013.6831269.

[499] M. Marchetti, M. Colajanni, M. Messori, L. Aniello, and Y. Vigfusson, "Cyber attacks on financial critical infrastructures," in *Collaborative Financial Infrastructure Protection*, R. Baldoni and G. Chockler, Eds. Berlin, Germany: Springer-Verlag, 2012, pp. 53–82.

[500] S. Amin and A. Giacomoni, "Smart grid, safe grid," *IEEE Power Energy Mag.*, vol. 10, no. 1, pp. 33–40, Jan.-Feb. 2012.

[501] A. Nicholson, S. Webber, S. Dyer, T. Patel, and H. Janicke, "SCADA security in the light of cyber-warfare," *Comput. Security*, vol. 31, no. 4, pp. 418–436, 2012.

[502] K.-K. R. Choo, "The cyber threat landscape: Challenges and future research directions," *Comput. Security*, vol. 30, no. 8, pp. 719–731, 2011.

[503] D. Kushner. (2013, Mar.). The real story of Stuxnet. *IEEE Spectrum.* [Online]. *50(3)*, pp. 48–53. Available: http://spectrum.ieee.org/telecom/security/the-real-story-of-stuxnet

[504] R. Perez-Pena, "Universities face a rising barrage of cyberattacks," *New York Times*, Jul. 2013. [Online]. Available: http://www.nytimes.com/2013/07/17/education/barrage-of-cyberattacks-challenges-campus-culture.html

[505] C. Tankard, "Advanced persistent threats and how to monitor and deter them," *Netw. Security*, vol. 2011, no. 8, pp. 16–19, 2011.

[506] S. Sorensen, "Security implications of software-defined networks," Fierce Telecom, 2012. [Online]. Available: http://www.fiercetelecom.com/story/security-implications-software-defined-networks/2012-05-14

[507] S. M. Kerner, "Is SDN secure?" Enterprise Networking Planet, Mar. 2013. [Online]. Available: http://www.enterprisenetworkingplanet.com/netsecur/is-sdn-secure.html

[508] A. Agapi *et al.*, "Routers for the cloud: Can the internet achieve 5-nines availability?" *IEEE Internet Comput.*, vol. 15, no. 5, pp. 72–77, Sep./Oct. 2011.

[509] R. Kloti, "OpenFlow: A security analysis," M.S. thesis, Dept. Inf. Tech. Elec. Eng., Swiss Fed. Inst. Technol. Zurich (ETH), Zurich, Switzerland, 2013.

[510] M. Wasserman and S. Hartman, "Security analysis of the Open Networking Foundation (ONF) OpenFlow switch specification," Internet Engineering Task Force, Apr. 2013. [Online]. Available: https://datatracker.ietf.org/doc/draft-mrw-sdnsec-openflow-analysis/

[511] S. Shin and G. Gu, "Attacking software-defined networks: A first feasibility study," in *Proc. 2nd Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 1–2.

[512] K. Benton, L. J. Camp, and C. Small, "OpenFlow vulnerability assessment," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 151–152.

[513] S. Scott-Hayward, G. O'Callaghan, and S. Sezer, "SDN security: A survey," in *Proc. IEEE SDN Future Netw. Services*, Nov. 2013, DOI: 10.1109/SDN4FNS.2013.6702553.

[514] S. Sezer *et al.*, "Are we ready for SDN? Implementation challenges for software-defined networks," *IEEE Commun. Mag.*, vol. 51, no. 7, pp. 36–43, Jul. 2013.

[515] S. Hernan, S. Lambert, T. Ostwald, and A. Shostack, "Uncover security design flaws using the STRIDE approach," *MSDN Mag.*, Nov. 2006. [Online]. Available: http://msdn.microsoft.com/en-us/magazine/cc163519.aspx

[516] B. Chandrasekaran and T. Benson, "Tolerating SDN application failures with LegoSDN," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 235–236.

[517] W. J. Bolosky, D. Bradshaw, R. B. Haagens, N. P. Kusters, and P. Li, "Paxos replicated state machines as the basis of a high-performance data store," in *Proc. Symp. Netw. Syst. Design Implement.*, 2011, pp. 141–154.

[518] P. Sousa, A. Bessani, M. Correia, N. Neves, and P. Verissimo, "Highly available intrusion-tolerant services with proactive-reactive recovery," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 4, pp. 452–465, Apr. 2010.

[519] R. Chua, "SDN security: Oxymoron? New interview with Phil Porras of SRI International," 2013. [Online]. Available: http://www.sdncentral. com/technology/sdn-security-oxymoron-phil-porras-sri/2013/02/

[520] J. Korniak, "The GMPLS controlled optical networks as industry communication platform," *IEEE Trans. Ind. Inf.*, vol. 7, no. 4, pp. 671–678, Nov. 2011.

[521] P. Fonseca, R. Bennesby, E. Mota, and A. Passito, "A replication component for resilient OpenFlow-based networking," in *Proc. IEEE Netw. Oper. Manage. Symp.*, Apr. 2012, pp. 933–939.

[522] U. Toseef, A. Zaalouk, T. Rothe, M. Broadbent, and K. Pentikousis, "C-BAS: Certificate-based AAA for SDN experimental facilities," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, 6 pp.

[523] F. Klaedtke, G. O. Karame, R. Bifulco, and H. Cui, "Access control for SDN controllers," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 219–220.

[524] D. Kreutz, A. Bessani, E. Feitosa, and H. Cunha, "Towards secure and dependable authentication and authorization infrastructures," in *Proc. Pacific Rim Int. Symp. Dependable Comput.*, Nov. 2014, pp. 43–52.

[525] D. Kreutz and E. Feitosa, "Identity providers-as-a-service built as cloud-of-clouds: Challenges and opportunities," in *Position Papers of the 2014 Federated Conference on Computer Science and Information Systems*, vol. 3, M. P. M. Ganzha and L. Maciaszek, Eds. Warsaw, Poland: PTI, 2014, pp. 101–108.

[526] P. Verssimo, N. Neves, and M. Correia, "Intrusion-tolerant architectures: Concepts and design," in *Architecting Dependable Systems*, vol. 2677, R. de Lemos, C. Gacek, and A. Romanovsky, Eds. Berlin, Germany: Springer-Verlag, 2003, pp. 3–36.

[527] C. E. Rothenberg *et al.*, "Revisiting routing control platforms with the eyes and muscles of software-defined networking," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 13–18.

[528] J. P. Stringer, Q. Fu, C. Lorier, R. Nelson, and C. E. Rothenberg, "Cardigan: Deploying a distributed routing fabric," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 169–170.

[529] C. E. Rothenberg *et al.*, "Hybrid networking towards a software defined era," in *Network Innovation Through OpenFlow and SDN: Principles and Design Book*. London, U.K.: Taylor & Francis/CRC Press, 2014.

[530] D. Levin, M. Canini, S. Schmid, F. Schaffert, and A. Feldmann, "Panopticon: Reaping the benefits of incremental SDN deployment in enterprise networks," in *Proc. USENIX Annu. Tech. Conf.*, Jun. 2014, pp. 333–345.

[531] H. Lu *et al.*, "Hybnet: Network manager for a hybrid network infrastructure," in *Proc. Ind. Track 13th ACM/IFIP/USENIX Int. Middleware Conf.*, 2013, pp. 6:1–6:6.

[532] A. Csoma *et al.*, "Multi-layered service orches-tration in a multi-domain network environment," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, 2 pp.

[533] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek, "The click modular router," *SIGOPS Oper. Syst. Rev.*, vol. 33, no. 5, pp. 217–231, Dec. 1999.

[534] S. Salsano *et al.*, "OSHI—Open source hybrid IP/SDN networking (and its emulation on Mininet and on distributed SDN testbeds)," Apr. 2014. [Online]. Available: http://arxiv.org/abs/1404.4806

[535] S. Vissicchio, L. Vanbever, and O. Bonaventure, "Opportunities and research challenges of hybrid software defined networks," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 70–75, Apr. 2014.

[536] P. Bernier, "NTT recognized with IBC award for SDN-based HDTV service," Sep. 2013. [Online]. Available: http://www.sdnzone.com/topics/software-defined-network/articles/353466-ntt-recognized-with-ibc-award-sdn-based-hdtv.htm

[537] NTT DATA, "Infrastructure services," 2014. [Online]. Available: http://www.nttdata.com/global/en/services/infrastructure/solution.html

[538] M. Wagner, "NTT taps SDN to enhance cloud flexibility," Mar. 2014. [Online]. Available: http://www.lightreading.com/ntt-taps-sdn-to-enhance-cloud-flexibility/d/d-id/708133

[539] AT&T Inc., "AT&T introduces the 'user-defined network cloud': A vision for the network of the future," Feb. 2014. [Online]. Available: http://www.att.com/gen/press-room?pid=25274&cdvn=news&newsarticleid=37439&mapcode=

[540] E. Haleplidis *et al.*, "ForCES applicability to SDN-enhanced NFV," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, 6 pp.

[541] Open Networking Foundation (ONF), "OpenFlow-enabled SDN and network functions virtualization," Feb. 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/solution-briefs/sb-sdn-nvf-solution.pdf

[542] I. Cerrato *et al.*, "User-specific network service functions in an SDN-enabled network node," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, 2 pp.

[543] M. Xia, M. Shirazipour, Y. Zhang, H. Green, and A. Takacs, "SDN and optical flow steering for network function virtualization," presented at the Open Networking Summit, Santa Clara, CA, USA, 2014.

[544] E. Haleplidis, J. Hadi Salim, S. Denazis, and O. Koufopavlou, "Towards a network abstraction model for SDN," *J. Netw. Syst. Manage.*, 2014, DOI: 10.1007/s10922-014-9319-3.

[545] A. Gember-Jacobson *et al.*, "OpenNF: Enabling innovation in network function control," in *Proc. ACM Conf. SIGCOMM*, 2014, pp. 163–174.

[546] I. Cerrato, M. Annarumma, and F. Risso, "Supporting fine-grained network functions through Intel DPDK," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, 6 pp.

[547] J. Ruckert, J. Blendin, N. Leymann, G. Schyguda, and D. Hausheer, "Demo: Software-defined network service chaining," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, 2 pp.

[548] J. Blendin, J. Ruckert, N. Leymann, G. Schyguda, and D. Hausheer, "Software-defined network service chaining," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, 6 pp.

[549] P. Skoldstrom *et al.*, "Towards unified programmability of cloud and carrier infrastructure," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, 6 pp.

[550] W. John *et al.*, "Research directions in network service chaining," in *Proc. IEEE SDN Future Netw. Services*, Nov. 2013, DOI: 10.1109/SDN4FNS.2013.6702549.

[551] B. Naudts *et al.*, "Techno-economic analysis of software defined networking as architecture for the virtualization of a mobile network," in *Proc. Eur. Workshop Softw. Defined Netw.*, Oct. 2012, pp. 67–72.

[552] Open Networking Foundation (ONF), "Operator network monetization through OpenFlow-enabled SDN," Apr. 2013. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/solution-briefs/sb-network-monetization.pdf

[553] P. Skoldstrom and W. John, "Implementation and evaluation of a carrier-grade OpenFlow virtualization scheme," in *Proc. 2nd Eur. Workshop Softw. Defined Netw.*, Oct. 2013, pp. 75–80.

[554] H. H. Gharakheili and V. Sivaraman, "Virtualizing national broad-band access infrastructure," in *Proc. 9th Int. Conf. Emerging Netw. Exp. Technol.*, 2013, pp. 27–30.

[555] Pacnet Australia, "Pacnet offers first Pan-Asia network-as-a-service architecture," Nov. 2013. [Online]. Available: http://www.cmo.com.au/mediareleases/17701/pacnet-offers-first-pan-asia-network-as-a-service/

[556] R. Bennesby, P. Fonseca, E. Mota, and A. Passito, "An inter-AS routing component for software-defined networks," in *Proc. IEEE Netw. Oper. Manage. Symp.*, Apr. 2012, pp. 138–145.

[557] N. D. Corporation, "NTT DATA advance in SDN business provides highly-flexible control of network by software," Jun. 2012. [Online]. Available: http://www.nttdata.com/global/en/news-center/pressrelease/2012/060801.html

[558] S. Das, A. Sharafat, G. Parulkar, and N. McKeown, "MPLS with a simple OPEN control plane," in *Proc. Nat. Fiber Opt. Eng. Conf. Opt. Fiber Commun. Conf. Expo.*, 2011, pp. 1–3.

[559] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian, "Fabric: A retrospective on evolving SDN," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 85–90.

[560] M. Martinello, M. Ribeiro, R. de Oliveira, and R. de Angelis Vitoi, "Keyflow: A prototype for evolving SDN toward core network fabrics," *IEEE Network*, vol. 28, no. 2, pp. 12–19, Mar. 2014.

[561] N. Feamster *et al.*, "SDX: A software-defined internet exchange," in *Proc. IETF*, Orlando, FL, USA, Mar. 2013. [Online]. Available: http://www.ietf.org/proceedings/86/slides/slides-86-sdnrg-6

[562] A. Devlic, W. John, and P. Skoldstrom, "A use-case based analysis of network management functions in the ONF SDN model," in *Proc. Eur. Workshop Softw. Defined Netw.*, Oct. 2012, pp. 85–90.

[563] M. Shirazipour, W. John, J. Kempf, H. Green, and M. Tatipamula, "Realizing packet-optical integration with SDN and OpenFlow 1.1 extensions," in *Proc. IEEE Int. Conf. Commun.*, 2012, pp. 6633–6637.

[564] W. John *et al.*, "SplitArchitecture: SDN for the carrier domain," *IEEE Commun. Mag.*, vol. 52, no. 10, pp. 146–152, Oct. 2014.

[565] W. John *et al.*, "Split architecture for large scale wide area networks," Feb. 2014.

[566] C.-Y. Hong *et al.*, "Achieving high utilization with software-driven WAN," in *Proc. ACM SIGCOMM Conf.*, 2013, pp. 15–26.

[567] D. Staessens, S. Sharma, D. Colle, M. Pickavet, and P. Demeester, "Software defined networking: Meeting carrier grade requirements," in *Proc. 18th IEEE Workshop

Local Metropolitan Area Netw., Oct. 2011, DOI: 10.1109/LANMAN.2011.6076935.

[568] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "A demonstration of automatic bootstrapping of resilient OpenFlow networks," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage.*, 2013, pp. 1066–1067.

[569] R. Niranjan Mysore *et al.*, "PortLand: A scalable fault-tolerant layer 2 data center network fabric," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 39–50, Aug. 2009.

[570] A. Greenberg *et al.*, "VL2: A scalable and flexible data center network," in *Proc. ACM SIGCOMM Conf. Data Commun.*, 2009, pp. 51–62.

[571] A. Sadasivarao *et al.*, "Bursting data between data centers: Case for transport SDN," in *Proc. IEEE 21st Annu. Symp. High-Performance Interconnects*, 2013, pp. 87–90.

[572] J. C. Tanner, "Taking SDN to transport and beyond," 2013. [Online]. Available: http://www.telecomasia.net/content/taking-sdn-transport-and-beyond

[573] S. Elby, "Carrier vision of SDN, 2012. [Online]. Available: http://www.brighttalk.com/webcast/6985/58527

[574] B. Anwer, T. Benson, N. Feamster, D. Levin, and J. Rexford, "A slick control plane for network middleboxes," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 147–148.

[575] C. Gerlach and H.-M. Foisel, "OIF carrier WG requirements on transport networks in SDN architectures," Optical Internetworking Forum (OIF), Tech. Rep., Sep. 2013.

[576] L. Velasco, A. Asensio, J. Berral, A. Castro, and V. Lopez, "Towards a carrier SDN: An example for elastic inter-datacenter connectivity," in *Proc. 39th Eur. Conf. Exhibit. Opt. Commun.*, 2013, DOI: 10.1049/cp.2013.1289.

[577] A. Alba *et al.*, "Efficient and agile storage management in software defined environments," *IBM J. Res. Develop.*, vol. 58, no. 2, pp. 1–12, Mar. 2014.

[578] W. Arnold *et al.*, "Workload orchestration and optimization for software defined environments," *IBM J. Res. Develop.*, vol. 58, no. 2, pp. 1–12, Mar. 2014.

[579] C. Dixon *et al.*, "Software defined networking to support the software defined environment," *IBM J. Res. Develop.*, vol. 58, no. 2, pp. 1–14, Mar. 2014.

[580] IBM Systems and Technology Group, "IBM software defined network for virtual environments," IBM Corporation, Tech. Rep., Jan. 2014.

[581] IBM Systems, "Manage all workloads with an efficient, scalable software defined environment (SDE)," 2014. [Online]. Available: http://www-03.ibm.com/systems/infrastructure/us/en/software-defined-environment/

ABOUT THE AUTHORS

Diego Kreutz (Member, IEEE) received the Computer Science degree, the M.Sc. degree in informatics, in 2009 and the M.Sc. degree in production engineering from the Federal University of Santa Maria, Santa Maria, Brazil, in 2005. Currently, he is working toward the Ph.D. degree in informatics engineering at the Faculty of Sciences of University of Lisbon, Lisbon, Portugal.

Over the past 11 years he has worked as an Assistant Professor in the Lutheran University of Brazil, Canoas, RS, Brazil, and in the Federal University of Pampa, Rio Grande do Sul, Brazil, and as a researcher member of the Software/Hardware Integration Lab (LISHA), Federal University of Santa Catarina, Florianópolis, Brazil. Out of the academia, he also has experience as an independent technical consultant on network operations and management for small and medium enterprises and government institutions. He is involved in research projects related to intrusion tolerance, security, and future networks including the TRONE and SecFuNet international projects. His main research interests are in network control platforms, software-defined networks, intrusion tolerance, system security and dependability, high-performance computing, and cloud computing.

Fernando M. V. Ramos (Member, IEEE) received the Ph.D. degree in computer science and engineering from the University of Cambridge, Cambridge, U.K., in 2012. He received his Master of Science degree in telecommunications from Queen Mary University of London, London, U.K. (with distinction, and best student award), in 2003, and the "Licenciatura" degree in electronics and telecommunications engineering (5 year undergraduate course) from the University of Aveiro, Aveiro, Portugal, in 2001.

He is an Assistant Professor at the University of Lisbon, Lisbon, Portugal. His previous academic positions include those of Teaching Assistant (Supervisor) at the University of Cambridge; at the Higher Institute of Engineering of Lisbon, Lisbon, Portugal; and at the University of Aveiro, Aveiro, Portugal. Over the past 12 years he has taught 20+ courses: from physics (electromagnetism) to electrical engineering (digital electronics, electric circuits, telecommunication systems, and foundations) to computer science (operating and distributed systems, computer networks, algorithms, programming languages). Periods outside academia include working as a Researcher at Portugal Telecom and at Telefonica Research Barcelona. His current research interests include software-defined networking, network virtualization, and cloud computing, with security and dependability as an orthogonal concern.

Paulo Esteves Veríssimo (Fellow, IEEE) is currently a Professor and FNR PEARL Chair at the Faculty of Science, Technology and Communication (FSTC), University of Luxembourg (UL), Luxembourg, Luxembourg; and head of the CritiX research group at UL's Interdisciplinary Centre for Security, Reliability and Trust. He is currently interested in secure and dependable distributed architectures, middleware and algorithms for: resilience of large-scale systems and critical infrastructures, privacy and integrity of highly sensitive data, and adaptability and safety of real-time networked embedded systems. He is the author of over 170 peer-refereed publications and coauthor of five books.

Prof. Veríssimo is a Fellow of the Association for Computing Machinery (ACM). He is an Associate Editor of the *International Journal on Critical Infrastructure Protection*. He is Chair of the International Federation for Information Processing (IFIP) Working Group (WG) 10.4 on Dependable Computing and Fault-Tolerance and Vice-Chair of the Steering Committee of the IEEE/IFIP Dependable Systems and Networks (DSN) Conference.

Christian Esteve Rothenberg (Member, IEEE) received the Telecommunication Engineering degree from the Universidad Politécnica de Madrid (ETSIT-UPM), Madrid, Spain, the M.Sc. (Dipl. Ing.) degree in electrical engineering and information technology from the Darmstadt University of Technology (TUD), Darmstadt, Germany, 2006, and the Ph.D. degree in computer engineering from the University of Campinas (UNICAMP), Campinas, Brazil, in 2010.

He is an Assistant Professor in the Faculty of Electrical and Computer Engineering, UNICAMP. From 2010 to 2013, he was a Senior Research Scientist in the areas of IP systems and networking at CPqD Research and Development Center in Telecommunications, Campinas, Brazil, where he was technical lead of R&D activities in the field of OpenFlow/SDN such as the RouteFlow project, the OpenFlow 1.3 Ericsson/CPqD softswitch, or the Open Networking Foundation (ONF) Driver competition. He holds two international patents and has over 50 publications, including scientific journals and top-tier networking conferences such as SIGCOMM and INFOCOM. Since April 2013, he has been an ONF Research Associate.

Siamak Azodolmolky (Senior Member, IEEE) received the Computer Engineering degree from Tehran University, Tehran, Iran, in 1994, the M.Sc. degree in computer architecture from Azad University in 1998, the M.Sc. degree, with distinction, from Carnegie Mellon University, Pittsburgh, PA, USA, in 2006, and the Ph.D. degree from the Universitat Politécnica de Catalunya (UPC), Barcelona, Spain, in 2011.

He was employed by Data Processing Iran Co. (IBM in Iran) as a Software Developer, Systems Engineer, and as a Senior R&D Engineer during 1992–2001. He joined Athens Information Technology (AIT) as a Research Scientist and Software Developer in 2007, while pursuing his Ph.D. degree. In August 2010, he joined the High Performance Networks research group of the School of Computer Science and Electronic Engineering (CSEE), University of Essex, Colchester, Essex, U.K., as a Senior Research Officer. He has been the technical investigator of various national and European Union (EU)-funded projects. Software-defined networking (SDN) has been one of his research interests since 2010, in which he has been investigating the extension of OpenFlow toward its application in core transport (optical) networks. He has published more than 50 scientific papers in international conferences, journals, and books. One of his recent books is *Software Defined Networking with OpenFlow* (Birmingham, U.K.: Packt Publishing, 2013). Currently, he is with Gesellschaft für Wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG), Göttigen, Germany, as a Senior Researcher and has led SDN related activities since September 2012.

Dr. Azodolmolky is a professional member of the Association for Computing Machinery (ACM).

Steve Uhlig (Member, IEEE) received the Ph.D. degree in applied sciences from the University of Louvain, Place de l'Université, Belgium, in 2004.

From 2004 to 2006, he was a Postdoctoral Fellow of the Belgian National Fund for Scientific Research (F.N.R.S.). His thesis won the annual IBM Belgium/F.N.R.S. Computer Science Prize 2005. Between 2004 and 2006, he was a Visiting Scientist at Intel Research, Cambridge, U.K., and at the Applied Mathematics Department, University of Adelaide, Adelaide, S.A., Australia. Between 2006 and 2008, he was with Delft University of Technology, Delft, The Netherlands. He was a Senior Research Scientist with the Technische Universität Berlin/Deutsche Telekom Laboratories, Berlin, Germany. Starting in January 2012, he became a Professor of Networks and Head of the Networks Research group at the Queen Mary University of London, London, U.K.