A Survey and a Layered Taxonomy of Software-Defined Networking

Yosr Jarraya, *Member, IEEE*, Taous Madi, and Mourad Debbabi, *Member, IEEE*

***Abstract*—Software-defined networking (SDN) has recently gained unprecedented attention from industry and research communities, and it seems unlikely that this will be attenuated in the near future. The ideas brought by SDN, although often described as a "revolutionary paradigm shift" in networking, are not completely new since they have their foundations in programmable networks and control–data plane separation projects. SDN promises simplified network management by enabling network automation, fostering innovation through programmability, and decreasing CAPEX and OPEX by reducing costs and power consumption. In this paper, we aim at analyzing and categorizing a number of relevant research works toward realizing SDN promises. We first provide an overview on SDN roots and then describe the architecture underlying SDN and its main components. Thereafter, we present existing SDN-related taxonomies and propose a taxonomy that classifies the reviewed research works and brings relevant research directions into focus. We dedicate the second part of this paper to studying and comparing the current SDN-related research initiatives and describe the main issues that may arise due to the adoption of SDN. Furthermore, we review several domains where the use of SDN shows promising results. We also summarize some foreseeable future research challenges.**

***Index Terms*—Software-defined networking, OpenFlow, programmable networks, controller, management, virtualization, flow.**

I. Introduction

FOR a long time, networking technologies have evolved at a lower pace compared to other communication technologies. Network equipments such as switches and routers have been traditionally developed by manufacturers. Each vendor designs his own firmware and other software to operate their own hardware in a proprietary and closed way. This slowed the progress of innovations in networking technologies and caused an increase in management and operation costs whenever new services, technologies or hardware were to be deployed within existing networks. The architecture of today's networks consists of three core logical planes: Control plane, data plane, and management plane. So far, networks hardware have been developed with tightly coupled control and data planes. Thus, traditional networks are known to be "inside the box" paradigm. This significantly increases the complexity and cost of network administration and management. Being aware of these limitations, networking research communities and industrial market leaders have collaborated in order to rethink the design of traditional networks. Thus, proposals for a new networking paradigm, namely programmable networks [1], have emerged (e.g., active networks [2] and Open Signalling (OpenSig) [3]).

Recently, Software-Defined Networking (SDN) has gained popularity in both academia and industry. SDN is not a revolutionary proposal but it is a reshaping of earlier proposals investigated several years ago, mainly programmable networks and control–data plane separation projects [4]. It is the outcome of a long-term process triggered by the desire to bring network "out of the box". The principal endeavors of SDN are to separate the control plane from the data plane and to centralize network's intelligence and state. Some of the SDN predecessors that advocate control–data plane separation are Routing Control Platform (RCP) [5], 4D [6], [7], Secure Architecture for the Networked Enterprise (SANE) [8], and lately Ethane [9], [10]. SDN philosophy is based on dissociating the control from the network forwarding elements (switches and routers), logically centralizing network intelligence and state (at the controller), and abstracting the underlying network infrastructure from the applications [11]. SDN is very often linked to the OpenFlow protocol. The latter is a building block for SDN as it enables creating a global view of the network and offers a consistent, system-wide programming interface to centrally program network devices. OpenFlow is an open protocol that was born in academia at Stanford University after the Clean Slate Project.[1] In [12], OpenFlow was proposed for the first time to enable researchers to run experimental protocols [13] in the campus networks they use every day. Currently, the Open Networking Foundation (ONF), a non-profit industry consortium, is in charge of actively supporting the advancements of SDN and the standardization of OpenFlow, which is currently published under version 1.4.0 [14].

The main objective of this paper is to survey the literature on SDN over the period 2008–2013 to provide a deep and comprehensive understanding of this paradigm, its related technologies, its domains of application, as well as the main issues that need to be solved towards sustaining its success. Despite SDN's juvenility, we have identified a large number of scientific publications not counting miscellaneous blogs, magazine articles, and online forums, etc. To the best of our knowledge, this paper is the first comprehensive survey on the SDN paradigm. While reviewing the literature, we found few papers surveying specifc aspects of SDN [15]–[17]. For

Manuscript received August 7, 2013; revised January 18, 2014; accepted April 3, 2014. Date of publication April 24, 2014; date of current version November 18, 2014.

The authors are with the Concordia Institute for Information Systems Engineering, Concordia University, Montreal, QC H3G 2W1, Canada (e-mail: y_jarray@encs.concordia.ca).

Digital Object Identifier 10.1109/COMST.2014.2320094

[1] http://cleanslate.stanford.edu/

instance, Bozakov and Sander [15] focus on the OpenFlow protocol and provide implementation scenarios using OpenFlow and NOX controller. Sezer *et al.* [17] briefly present a survey on SDN concepts and issues while considering a limited number of surveyed works. Lara *et al.* [16] present an OpenFlow-oriented survey and concentrate on a two-layer architecture of SDN: control and data layers. They review and compare OpenFlow specifications, from the earliest versions till version 1.3.0. and then present works on OpenFlow capabilities, applications, and deployments all around the word. Although important research issues have been identified, there is no mentioning about other relevant aspects such as distributed controllers, northbound APIs, and SDN programming languages.

In the present paper, we aim at providing a more comprehensive and up-to-date overview of SDN by targeting more than one aspect while analyzing most relevant research works and identifying foreseeable future research directions. The main contributions of this paper are as follows:

- Provide a comprehensive tutorial on SDN and OpenFlow by studying their roots, their architecture and their principal components.
- Propose a taxonomy that allows classifying the reviewed research works, bringing relevant research directions into focus, and easing the understandability of the related domains.
- Elaborate a survey on the most relevant research proposals supporting the adoption and the advancement of SDN.
- Identify new issues raised from the adoption of SDN that still need to be addressed by future research efforts.

The paper is structured as follows; Section II is a preliminary section where the roots of SDN are briefly presented. Section III is dedicated to unveiling SDN concepts, components, and architecture. Section IV discusses existing SDN taxonomies and elaborates on a novel taxonomy for SDN. Section V is the survey of the research works on SDN organized according to the proposed taxonomy. The latter identifies issues brought by SDN paradigm and the currently proposed solutions. Section VI describes open issues that still need to be addressed in this domain. The paper ends with a conclusion in Section VII.

II. Software-Defined Networking Roots

SDN finds its roots in programmable networks and control–data plane separation paradigms. In the following, we give a brief overview of these two research directions and then highlight how SDN differs from them.

The key principle of programmable networks is to allow more flexible and dynamically customizable network. To materialize this concept, two separate schools of thoughts have emerged: OpenSig [3] from the community of telecommunications and active networks [2] from the community of IP networks. Active networking emerged from DARPA [2] in mid 1990s. Its fundamental idea was to allow customized programs to be carried by packets and then executed by the network equipments. After executing these programs, the behavior of switches/routers would be subject to change with different levels of granularity. Various suggestions on the levels of programmability exist in the literature [1]. Active networks introduced a high-level dynamism for the deployment of new services at run-time. At the same time, OpenSig community has proposed to control networks through a set of well-defined network programming interfaces and distributed programming environments (middleware toolkits such as CORBA) [3]. In that case, physical network devices are manipulated like distributed computing objects. This would allow service providers to construct and manage new network services (e.g., routing, mobility management, etc.) with QoS support.

Both active networking and OpenSig introduced many performance, isolation, complexity, and security concerns. First, they require that each packet (or subset of packets) is processed separately by network nodes, which raises performance issues. They require executing code at the infrastructure level, which needs most, if not all, routers to be fundamentally upgraded, and raises security and complexity problems. This was not accepted by major network devices vendors, and consequently hampered research and industrial developments in these directions. A brief survey on approaches to programmable networks can be found in [18], where SDN is considered as a separate proposal towards programmable networks besides three other paradigms, namely approaches based on: 1) Improved hardware routers such as active networks, OpenSig, Juniper Network Operating System SDK (Junos SDK), 2) Software routers such as Click and XORP, 3) Virtualization such as network virtualization, overlay network, and virtual routers. The survey on programmable networks presented in [1] is a more comprehensive but less recent. SDN resembles past research on programmable networks, particularly active networking. However, while SDN has an emphasis on programmable control plane, active networking focuses on programmable data planes [19].

After programmable networks, projects towards control–data plane separation have emerged supported by efforts towards standard open interface between control and data planes such as the Forwarding and Control Element Separation (ForCES) framework [20] and by efforts to enable a logically centralized control of the network such as the Path Computation Element Protocol (PCEP) [21] and RCP [5]. Although focusing on control–data plane separation, these efforts rely on existing routing protocols. Thus, these proposals do neither support a wide range of functionalities (e.g., dropping, flooding, or modifying packets) nor do they allow for a wider range of header fields matching [19]. These restrictions impose significant limitations on the range of applications supported by programmable controllers [19]. Furthermore, most of the aforementioned proposals failed to face backwards compatibility challenges and constraints, which inhibited immediate deployment.

To broaden the vision of control and data plane separation, researchers explored clean-slate architectures for logically centralized control such as 4D [6], SANE [8], Ethane [9]. Clean Slate 4D project [6] was one of the first advocating the redesign of control and management functions from the ground up based on sound principles. SANE [8] is a single protection layer consisting of a logically centralized server that enforces several security policies (access control, firewall, network address translation, etc.) within the enterprise network. And more recently, Ethane [9] is an extension of SANE that is based on the principle of incremental deployment in enterprise networks. In

Ethane, two components can be distinguished; The first component is a controller that knows the global network topology and contains the global network policy used to determine the fate of all packets. It also performs route computation for the permitted flows. The second component is a set of simple and dump Ethane switches. These switches consist of a simple flow table and use a secure channel to communicate with the controller for exchanging information and receiving forwarding rules. This principle of packet processing constitutes the basis of SDN's proposal.

The success and fast progress of SDN are widely due to the success of OpenFlow and the new vision of a network operating system. Unlike previous proposals, OpenFlow specification relies on backwards compatibility with hardware capabilities of commodity switches. Thus, enabling OpenFlow's initial set of capabilities on switches did not need a major upgrade of the hardware, which encouraged immediate deployment. In later versions of the OpenFlow switch specification (i.e., starting from 1.1.0), an OpenFlow-enabled switch supports a number of tables containing multiple packet-handling rules, where each rule matches a subset of the traffic and performs a set of actions on it. This potentially prepares the floor for a large set of controller's applications with sophisticated functionalities. Furthermore, the deployment of OpenFlow testbeds by researchers not only on a single campus network but also over a wide-area backbone network demonstrated the capabilities of this technology. Finally, a network operating system, as envisioned by SDN, abstracts the state from the logic that controls the behavior of the network [19], which enables a flexible programmable control plane.

In the following sections, we present a tutorial and a comprehensive survey on SDN where we highlight the challenges that have to be faced to provide better chances for SDN paradigm.

III. SDN: Global Architecture and Meronomy

In this section, we present the architecture of SDN and describe its principal components. According to the ONF, SDN is an emerging architecture that decouples the network control and forwarding functions. This enables the "network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services".[2] In such an architecture, the infrastructure devices become simply forwarding engines that process incoming packets based on a set of rules generated on the fly by a (or a set of) controller at the control layer according to some predefined program logic. The controller generally runs on a remote commodity server and communicates over a secure connection with the forwarding elements using a set of standardized commands. ONF presents in [11] a high-level architecture for SDN that is vertically split into three main functional layers:

- *Infrastructure Layer*: Also known as the data plane [11], it consists mainly of Forwarding Elements (FEs) including physical and virtual switches accessible via an open interface and allows packet switching and forwarding.

[2]ONF, https://www.opennetworking.org/sdn-resources/sdn-definition

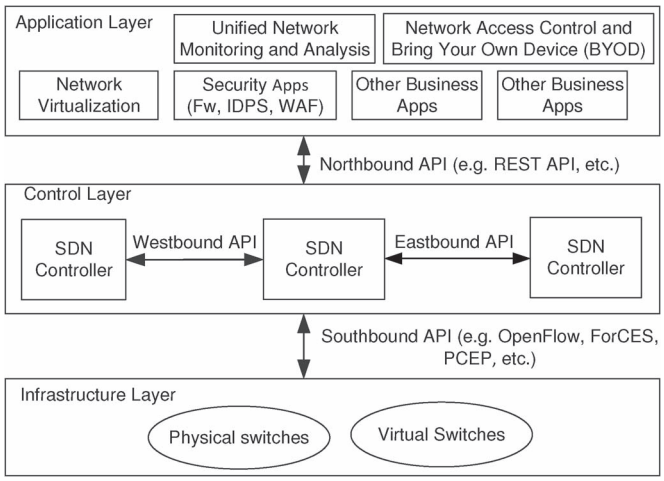

Fig. 1. SDN architecture [11], [23].

- *Control Layer*: Also known as the control plane [11], it consists of a set of software-based SDN controllers providing a consolidated control functionality through open APIs to supervise the network forwarding behavior through an open interface. Three communication interfaces allow the controllers to interact: southbound, northbound and east/westbound interfaces. These interfaces will be briefly presented next.
- *Application Layer*: It mainly consists of the end-user business applications that consume the SDN communications and network services [22]. Examples of such business applications include network visualization and security business applications [23].

Fig. 1 illustrates this architecture while detailing some key parts of it, such as the control layer, the application layer, as well as the communication interfaces among the three layers.

A SDN controller interacts with these three layers through three open interfaces:

- *Southbound*: This communication interface allows the controller to interact with the forwarding elements in the infrastructure layer. OpenFlow, a protocol maintained by ONF [14], is according to ONF a foundational element for building SDN solutions and can be viewed as a promising implementation of such an interaction. At the time of writing this paper, the latest OpenFlow version is 1.4 [14]. The evolution of OpenFlow switch specification will be summarized in later sections. Most non-OpenFlow-based SDN solutions from various vendors employ proprietary protocols such as Cisco's Open Network Environment Platform Kit (onePK) [24] and Juniper's contrail [25]. Other alternatives to OpenFlow exist, for instance, the Forwarding and Control Element Separation (ForCES) framework [20]. The latter defines an architectural framework with associated protocols to standardize information exchange between the control and forwarding layers. It has existed for several years as an IETF proposal but it has never achieved the level of adoption of OpenFlow. A comparison between OpenFlow and ForCES can be found in [26].
- *Northbound*: This communication interface enables the programmability of the controllers by exposing universal

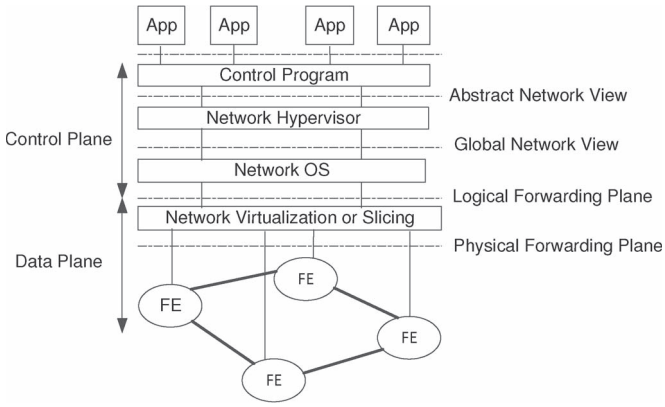

Fig. 2. SDN abstraction layers [29], [30].

network abstraction data models and other functionalities within the controllers for use by applications at the application layer. It is more considered as a software API than a protocol that allows programming and managing the network. At the time of writing this paper, there is no standardization effort yet from the ONF side who is encouraging innovative proposals from various controllers' developers. According to the ONF, different levels of abstractions as latitudes and different use cases as longitudes have to be characterized, which may lead to more than a single northbound interface to serve all use cases and environments. Among the various proposals, various vendors are offering a REpresentational State Transfer (REST)-based APIs [27] to provide a programmable interface to their controller to be used by the business applications.

- *East/Westbound*: This interface is an envisioned communication interface, which is not currently supported by an accepted standard. It is mainly meant for enabling communication between groups or federations of controllers to synchronize state for high availability [28].

From another perspective, several logical layers defined by abstraction were presented for control [29] and data [30] layers. These layers of abstractions simplify understanding of the SDN vision, decrease network programming complexity and facilitate reasoning about such networks. Fig. 2 compiles these logical layers and can be described in a bottom-up approach as follows:

- *Physical Forwarding Plane*: This refers to the set of physical network forwarding elements [30].
- *Network Virtualization* (or slicing): This refers to an abstraction layer that aims at providing great flexibility to achieve operational goals, while being independent from the underlying physical infrastructure. It is responsible for configuring the physical forwarding elements so that the network implements the desired behavior as specified by the logical forwarding plane [30]. At this layer, there exist proposals to slice network flows such as FlowVisor [13], [30], [31].
- *Logical Forwarding Plane*: It is a logical abstraction of the physical forwarding plane that provides an end-to-end forwarding model. It allows abstracting from the physical infrastructure. This abstraction is realized by the network virtualization layer [30].
- *Network Operating System*: A Network Operating System (NOS) may be though of as a software that abstracts the installation of state in network switches from the logic and applications that control the behavior of the network [4]. It provides the ability to observe and control a network by offering a programmatic interface (NOS API) as well as an an execution environment for programmatic control of the network [32]. NOS needs to communicate with the forwarding elements in two-ways: receives information in order to build the global state view and pushes the needed configurations in order to control the forwarding mechanisms of these elements [29]. The concept of a single network operating system has been extended to distributed network operating system to accommodate large-scale networks, such as ONOS,[3] where open source software are used to maintain consistency across distributed state and to provide a network topology database to the applications [4].
- *Global Network View*: It consists of an annotated network graph provided through an API [29].
- *Network Hypervisor*: Its main function is to map the abstract network view into the global network view and vice-versa [33].
- *Abstract Network View*: It provides to the applications a minimal amount of information required to specify management policies. It exposes an "abstract" view of the network to the applications rather than a topologically faithful view [29].

In the following, we provide the details on the role and implementations of each layer in the architecture.

A. Forwarding Elements

In order to be useful in an SDN architecture, forwarding elements, mainly switches, have to support a southbound API, particularly OpenFlow. OpenFlow switches come in two flavors: Software-based (e.g., Open vSwitch (OVS) [34]–[36]) and OpenFlow-enabled hardware-based implementations (e.g., NetFPGA [37]). Software switches are typically well-designed and comprise complete features. However, even mature implementations suffer from being often quite slow. Table I provides a list of software switches supporting OpenFlow. Hardware-based OpenFlow switches are typically implemented as Application-Specific Integrated Circuits (ASICs); either using merchant silicon from vendors or using a custom ASIC. They provide line rate forwarding for large number of ports but lack the flexibility and feature completeness of software implementations [38]. There are various commercial vendors that support OpenFlow in their hardware switches including but not limited to HP, NEC, Pronto, Juniper, Cisco, Dell, Intel, etc.

An OpenFlow-enabled switch can be subdivided into three main elements [12], namely, a hardware layer (or datapath), a software layer (or control path), and the OpenFlow protocol:

- The datapath consists of one or more flow tables and a group table, which perform packet lookups and forwarding.

[3]Open Network Operating System (ONOS) http://www.sdncentral.com/projects/onos-open-network-operating-system/

TABLE I
OPENFLOW STACKS AND SWITCH IMPLEMENTATIONS

OpenFlow Switch	Description	Open Source	Language	Origin	OpenFlow Version
Open vSwitch [34]	OpenFlow Stack: Soft switch and control stack for hardware switching	yes	C/Python	Multiple contributors	v 1.0
OpenFlow Reference Implementation [39]	OpenFlow Stack that follows the specification	yes	C	Stanford University/ Nicira Networks	v 0.8
Pica8 [40]	hardware-independent software for hardware switching	not yet	C	Pica8	v 1.2
Indigo [41]	For OpenFlow on physical and hypervisor switches based on the Stanford reference implementation	yes	C/Lua	Big Switch Networks	v 1.0
Pantou/OpenWRT [42]	Enable Commercial OpenWRT wireless devices with OpenFlow	yes	C	-	v 1.0

A flow table consists of flow entries each associated with an (or a set of) action that tells the switch how to process the flow. Flow tables are typically populated by the controller. A group table consists of a set of group entries. It allows to express additional methods of flow forwarding.

- The control path is a channel that connects the switch to the controller for signaling and for programming purposes. Commands and packets are exchanged through this channel using the OpenFlow protocol.
- The OpenFlow protocol [14] provides the means of communication between the controller and the switches. Exchanged messages may include information on received packets, sent packets, statistics collection, actions to be performed on specific flows, etc.

The first release of OpenFlow was published by Stanford University in 2008. Since 2011, the OpenFlow switch specification has been maintained and improved by ONF starting from version 1.0 [43] onward. The latter version is currently widely adopted by OpenFlow vendors. In that version, forwarding is based on a single flow table and matching focuses only on layer 2 information and IPv4 addresses. The support of multiple flow tables and MPLS tags has been introduced in version 1.1, while IPv6 support has been included in version 1.2.

In version 1.3 [44], the support for multiple parallel channels between switches and controllers has been added. The latest available OpenFlow switch specification published in 2013 is version 1.4 [14]. The main included improvements are the retrofitting of various parts of the protocol with the TLV structures introduced in version 1.2 for extensible matching fields and a flow monitoring framework allowing a controller to monitor in real-time the changes to any subset of the flow tables done by other controllers. In the rest of this paper, we describe the OpenFlow switch specification version 1.4 [14], if the version is not explicitly mentioned.

A flow table entry in an OpenFlow-enabled switch is constituted of several fields that can be classified as follows:

- Match fields to match packets based on a 15-tuple packet's header, the ingress port, and optionally packet's metadata. Fig. 3 illustrates the packet header fields grouped according to the OSI layers L1-4.
- Priority of the flow entry, which prioritizes the matching precedence of the flow entry.
- An action set that specifies actions to be performed on packets matching the header field. The three basic actions are: forward the packet to a port or a set of ports, forward the flow's packets to the controller and drop the flow's packets.
- Counters to keep track of flow statistics (the number of packets and bytes for each flow, and the time since the last packet has matched the flow).
- Timeouts specifying the maximum amount of time or idle time before the flow is expired by the switch.

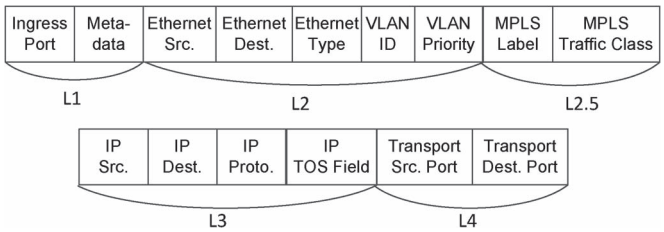

Fig. 3. Flow identification in OpenFlow.

OpenFlow messages can be categorized into three main types [14]: controller-to-switch, asynchronous, and symmetric. Messages initiated by the controller and used to manage or inspect the state of the switches are the controller-to-switch messages. A switch may initiate asynchronous messages in order to update the controller on network events and changes to the switch's state. Finally, symmetric messages are initiated, without solicitation, by either the switch or the controller and they are used, for instance, to test the liveliness of a controller-switch connection. Once an ingress packet arrives to the OpenFlow switch, the latter performs lookup in the flow tables based on pipeline processing [14]. A flow table entry is uniquely identified by its matching fields and its priority. A packet matches a given flow table entry if the values in the packet match those specified in the entry's fields. A flow table entry field with a value of ANY (field omitted or wildcard field) matches all possible values in the header. Only the highest priority flow entry that matches the packet must be selected. In the case the packet matches multiple flow entries with the same highest priority, the selected flow entry is explicitly undefined [14]. In order to remediate to such a scenario, OpenFlow specification [14] provides a mechanism that enables the switch to optionally verify whether the added new flow entry overlaps with an existing entry. Thus, a packet can be matched exactly to a flow (microflow), matched to a flow with wildcard fields (macroflow) or does not match any flow. In the case of a match found, the set of actions will be performed as defined in the matching flow table entry. In the case of no mach, the switch forwards the packet (or just its header) to the controller to request a decision. After consulting the associated

TABLE II
SDN CONTROLLERS

Controller	Open Source	Language	Multi-threaded	GUI	Origin
NOX [32]	yes	C++/Python	no	yes	Nicira Networks
NOX-MT [46]	yes	C++	yes	no	Nicira Networks and Big Switch Networks
POX [47]	yes	Python	-	yes	Nicira Networks
Maestro [48]	yes	Java	yes	no	Rice University
Beacon [49]	yes	Java	yes	yes	Stanford University
SNAC [50]	no	C++/Python	no	yes	Nicira Networks
RISE [51]	yes	C and Ruby	non-guaranteed	no	NEC
Floodlight [52]	yes	Java	-	yes	Big Switch Networks
McNettle [53]	yes	Nettle/Haskell	no	no	Yale University
MUL [54]	yes	C	yes	yes	KulCloud
RYU [55]	yes	Python	-	-	NTT OSRG and VA Linux
OpenDaylight [56]	yes	Java	yes	yes	Multiple contributors

policy located at the management plane, the controller responds by a new flow entry to be added to the switch's flow table. The latter entry is used by the switch to handle the queued packet as well as the subsequent packets in the same flow.

In order to dynamically and remotely configure OpenFlow switches, a protocol, namely the OpenFlow Configuration and Management Protocol (OF-CONFIG) [45], is also being maintained by ONF. The latter enables the configuration of essential artifacts so that an OpenFlow controller can communicate with the network switches via OpenFlow. It operates on a slower time-scale than OpenFlow as it is used for instance to enable/disableaport on a switch, to set the IP address of the controller, etc.

B. Controllers

The controller is the core of SDN networks as it is the main part of the NOS. It lies between network devices at the one end and the applications at the other end. An SDN controller takes the responsibility of establishing every flow in the network by installing flow entries on switch devices. One can distinguish two flow setup modes: Proactive vs. Reactive. In proactive settings, flow rules are pre-installed in the flow tables. Thus, the flow setup occurs before the first packet of a flow arrives at the OpenFlow switch. The main advantages of a proactive flow setup is a negligible setup delay and a reduction in the frequency of contacting the controller. However, it may overflow flow tables of the switches. With respect to a reactive flow setup, a flow rule is set by the controller only if no entry exists in the flow tables and this is performed as soon as the first packet of a flow reaches the OpenFlow switch. Thus, only the first packet triggers a communication between the switch and the controller. These flow entries expire after a pre-defined timeout of inactivity and should be wiped out. Although a reactive flow setup suffers from a large round trip time, it provides a certain degree of flexibility to make flow-by-flow decisions while taking into account QoS requirements and traffic load conditions. To respond to a flow setup request, the controller first checks this flow against policies on the application layer and decides on the actions that need to be taken. Then, it computes a path for this flow and installs new flow entries in each switch belonging to this path, including the initiator of the request.

With respect to the flow entries installed by the controller, there is a design choice over the controlled flow granularity, which raises a trade-off between flexibility and scalability based on the requirements of network management. Although a fine grained traffic control, called micro-flows, offers flexibility, it can be infeasible to implement especially in the case of large networks. As opposed to micro-flows, macro-flows can be built by aggregating several micro-flows simply by replacing exact bit pattern with wildcard. Applying a coarse grained traffic control, called macro-flow, allows gaining in terms of scalability at the cost of flexibility.

In order to get an overview on the traffic in the switches, statistics are communicated between the controller and the switches. There are two ways for moving the statistics from the switch to the controller: Push-based vs. pull-based flow monitoring. In a push-based approach, statistics are sent by each switch to the controller to inform about specific events such as setting up a new flow or removing a flow table entry due to idling or hard timeouts. This mechanism does not inform the controller about the behavior of a flow before the entry times out, which is not useful for flow scheduling. In a pull-based approach, the controller collects the counters for a set of flows matching a given flow specification. It can optionally request a report aggregated over all flows matching a wildcard specification. While this can save switch-to-controller bandwidth, it disables the controller from learning much about the behavior of specific flows. A pull-based approach requires tuning the delay between controller's requests as this may impact the scalability and reliability of operations based on statistics gathering.

In what follows, we present a list of the most prominent existing OpenFlow controllers. Note that most of these reviewed SDN controllers (except RYU) currently support OpenFlow version 1.0. The controllers are summarized in Table II and compared based on the availability of the source code, the implementation language, whether multi-threading is supported, the availability of a graphical user interface, and finally their originators.

- **NOX** [32] is the first, publicly available, OpenFlow single threaded controller. Several derivatives of NOX exist; A multi-threaded successor of NOX, namely NOX-MT has been proposed in [46]. QNOX [57] is a QoS-aware version of NOX based on Generalized OpenFlow, which is an extension of OpenFlow supporting multiple layer networking in the spirit of GMPLS. FortNox [58] is another extension of NOX, which implements a conflict analyzer to detect and re-conciliate conflicting flow rules caused by dynamic OpenFlow applications insertions. Finally, POX [47] controller is a pure Python controller, redesigned to improve the performance of the original NOX controller.

TABLE III
SDN PROGRAMMING LANGUAGES

Framework	Level of Abstraction	Query Language	Runtime System	Implementation Language	Programming Type	Policies Type
Frenetic [66], [67]	high	yes	yes	Python	FRP	Active
NetCore [68]	high	yes	yes	Python	FRP	Active
Nettle [61]	low	no	no	Haskell	FRP	Active
FML [69]	high	no	no	Python/C++	LP	Passive
Procera [70]	high	no	no	Haskell	FRP	Active

- **Maestro** [48], [59], [60] takes advantage of multicore technology to perform parallelism at low-level while keeping a simple programming model for the application's programmers. It achieves performance through distributing tasks evenly over available working threads. Moreover, Maestro processes a batch of flow requests at once, which would increase its efficiency. It has been shown that on an eight-core server, Maestro throughput may achieve a near linear scalability for processing flow setup requests.
- **Beacon** [49] is built at Stanford University. It is a multi-threaded, cross-platform, modular controller that optionally embeds the Jetty enterprise web server and a custom extensible user interface framework. Code bundles in Beacon can be started, stopped, refreshed, and installed at runtime, without interrupting other non-dependent bundles.
- **SNAC** [50] uses a web-based policy manager to monitor the network. A flexible policy definition language and a user friendly interface are incorporated to configure devices and monitor events.
- **Floodlight** [52] is a simple and performent Java-based OpenFlow Controller that was forked from Beacon. It has been tested using both physical and virtual OpenFlow-compatible switches. It is now supported and enhanced by a large community including Intel, Cisco, HP, and IBM.
- **McNettle** [53] is an SDN controller programmed with Nettle [61], a Domain-Specific Language (DSL) embedded in Haskell, that allows programming OpenFlow networks in a declarative style. Nettle is based on the principles of Functional Reactive Programming (FRP) that allows programming dynamic controllers. McNettle operates on shared-memory multicore servers to achieve global visibility, high throughput, and low latency.
- **RISE** [51], for Research Infrastructure for large-Scale network Experiments, is an OpenFlow controller based on Trema.[4] The latter is an OpenFlow stack framework based on Ruby and C. Trema provides an integrated testing and debugging environment and includes a development environment with an integrated tool chain.
- **MUL** [54] is a C-based multi-threaded OpenFlow SDN controller that supports a multi-level northbound interface for hooking up applications.
- **RYU** [55] is a component-based SDN framework. It is open sourced and fully developed in python. It allows layer 2 segregation of tenants without using VLAN. It supports OpenFlow v1.0, v1.2, v1.3, and the Nicira Extensions.
- **OpenDaylight** [56] is an open source project and a software SDN controller implementation contained within its own Java virtual machine. As such, it can be deployed on any hardware and operating system platform that supports Java. It supports the OSGi framework [62] for local controller programmability and bidirectional REST [27] for remote programmability as northbound APIs. Companies such as ConteXtream, IBM, NEC, Cisco, Plexxi, and Ericsson are actively contributing to OpenDaylight.

[4]http://trema.github.com/trema/

C. Programming SDN Applications

SDN applications interact with the controllers through the northbound interface to request the network state and/or to request and manipulate the services provided by the network. While the southbound interface between the controller software and the forwarding elements is reasonably well-defined through standardization efforts of the underlying protocols such as OpenFlow and ForCES, there is no standard yet for the interactions between controllers and SDN applications. This may stem from the fact that the northbound interface is more a set of software-defined APIs than a protocol exposing the universal network abstraction data models and the functionality within the controller [23]. Programming using these APIs allows SDN applications to easily interface and reconfigure the network and its components or pull specific data based on their particular needs [63]. From the one hand, northbound APIs can enable basic network functions including path computation, routing, traffic steering, and security. From the other hand, they also allow orchestration systems such as the OpenStack Quantum [64] to manage network services in a cloud.

SDN programming frameworks consist generally of a programming language and eventually the appropriate tools for compiling and validating the OpenFlow rules generated by the application program as well as for querying the network state (see Table III). SDN programming languages can be compared according to three main design criteria: the level of abstraction of the programming language, the class of language it belongs to, and the type of programmed policies:

- *Level of Abstraction: Low-Level vs. High-Level*. Low-level programming languages allow developers to deal with details related to OpenFlow, whereas high-level programming languages translate information provided by the OpenFlow protocol into a high-level semantics. Translating the information provided by the OpenFlow protocol into a high-level semantics allows programmers to focus on network management goals instead of details of low-level rules.
- *Programming: Logic vs. Functional Reactive*. Most of the existing network management languages adopt the declarative programming paradigm, which means that only the

logic of the computation is described (what the program should accomplish), while the control flow (how to accomplish it) is delegated to the implementation. Nevertheless, there exist two different programming fashions to express network policies: Logic Programming (LP) and Functional Reactive Programming (FRP). In logic programming, a program is constituted of a set of logical sentences. It applies particularly to areas of artificial intelligence. Functional reactive programming [65] is a paradigm that provides an expressive and a mathematically sound approach for programming reactive systems in a declarative manner. The most important feature of FRP is that it allows to capture both continuous time-varying behaviors and event-based reactivity. It is consequently used in areas such as robotics and multimedia.

- *Policy Logic: Passive vs. Active*. A programming language can be devised to develop either passive or active policies. A passive policy can only observe the network state, while an active policy is programmed to reactively affect the network-wide state as a response to certain network events. An example of a reactive policy is to limit the network access to a device/a user based on a maximum bandwidth usage.

In the following we examine the most relevant programming frameworks proposed for developing SDN applications.

1) Frenetic [66], [67]: It is a high-level network programming language constituted of two levels of abstraction. The first is a low-level abstraction that consists of a runtime system that translates high-level policies and queries into low-level flow rules and then issues the needed OpenFlow commands to install these rules on the switches. The second is a high-level abstraction that is used to define a declarative network query language that resembles the Structured Query Language (SQL) and a FRP-based network policy management library. The query language provides means for reading the state of the network, merging different queries, and expressing high-level predicates for classifying, filtering, transforming, and aggregating the packets' streams traversing the network. To govern packet forwarding, the FRP-based policy management library offers high-level packet-processing operators that manipulate packets as discrete streams only. This library allows reasoning about a unified architecture based on "see every packet" abstraction and describing network programs without the burden of low-level details. Frenetic language offers operators that allows combining policies in a modular way, which facilitates building new tools out of simpler reusable parts. Frenetic has been used to implement many services such as load balancing, network topology discovery, fault tolerance routing and it is designed to cooperate with the controller NOX.

Frenetic defines only parallel composition, which gives each application the illusion of operating on its own copy of each packet. Monsanto *et al.* [71] defines an extension to Frenetic language with a sequential composition operator so that one module acts on the packets produced by another module. Furthermore, an abstract packet model was introduced to allow programmers extending packets with virtual fields used to associate packets with high-level meta-data and topology abstraction. This abstraction allows to limit the scopes of network view and module' actions, which achieves information hiding and protection, respectively.

2) NetCore [68]: It is a successor of Frenetic that enriches the policy management library of Frenetic and proposes algorithms for compiling monitoring policies and managing controller-switch interactions. NetCore has a formal semantics and its algorithms have been proved correct. NetCore defines a core calculus for high-level network programming that manipulates two components: Predicates, which match sets of packets, and policies, which specify locations where to forward these packets. Set-theoretic operations are defined to build more complex predicates and policies from simple ones. Contrarily to Frenetic, NetCore compiler uses wildcard rules to generate switch classifiers (sets of packet-forwarding rules), which increases the efficiency of packets processing on the switches.

3) Nettle [61]: It is another FRP-based approach for programming OpenFlow networks that is embedded in Haskell [72], a strongly typed language. It defines signal functions that transform messages issued from switches into commands generated by the controller. Nettle allows to manipulate continuous quantities (values) that reflect abstract properties of a network, such as the volume of messages on a network link. It provides a declarative mechanism for describing time-sensitive and time-varying behaviors such as dynamic load balancing. Compared to Frenetic, Nettle is considered as a low-level programming language, which makes it more appropriate for programming controllers. However, it can be used as a basis for developing higher level DSL for different tasks such as traffic engineering and access control. Moreover, Nettle has a sequential operator for creating compound commands but lacks a support for composing modules affecting overlapping portions of the flow space, as it is proposed by Frenetic.

4) Procera [70]: It is an FRP-based high-level language embedded in Haskell. It offers a declarative, expressive, extensible, and compositional framework for network operators to express realistic network policies that react to dynamic changes in network conditions. These changes can be originated from OpenFlow switches or even from external events such as user authentication, time of the day, measurements of bandwidth, server load, etc. For example, access to a network can be denied when a temporal bandwidth usage condition occurs.

5) Flow-Based Management Language (FML) [69]: It is a declarative language based on non-recursive Datalog, a declarative logic programming language. A FML policy file consists of a set of declarative statements and may include additionally external references to, for instance, SQL queries. While the combination of policies statements written by different authors is made easy, conflicts are susceptible to be created. Therefore, a conflict resolution mechanism is defined as a layer on top of the core semantics of FML. For each new application of FML, developers can define a set of keywords that they need to implement. FML is written in C++ and Python and operates within NOX. Although FML provides a high-level abstraction, contrarily to Procera, it lacks expressiveness for describing dynamic policies, where forwarding decisions change over time. Moreover, FML policies are passive, which means they can only observe the network state without modifying it.

IV. SDN TAXONOMY

The first step in understanding SDN is to elaborate a classification using a taxonomy that simplifies and eases the understanding of the related domains. In the following, we elaborate a taxonomy of the main issues raised by the SDN networking paradigm and the solutions designed to address them. Our proposed taxonomy provides a hierarchical view and classifies the identified issues and solutions per layer: infrastructure, control, and application. We also consider inter-layers, mainly application/control, control/infrastructure, and application/control/infrastructure.

While reviewing the literature, we found only two taxonomies where each focuses on a single aspect of SDN: The first is a taxonomy based on switch-level SDN deployment provided by Gartner in a non public report [73], that we will not detail here, and the second focuses abstractions for the control plane [18]. The latter abstractions are meant for ensuring compatibility with low-level hardware/software and enabling making decisions based on the entire network. The three proposed control plane abstractions in [18], [74] are as follows:

- *Forwarding Abstraction*: A flexible forwarding model that should support any needed forwarding behavior and should hide details of the underlying hardware. This corresponds to the aforementioned logical forwarding plane.
- *Distributed State Abstraction*: This abstraction aims at abstracting away complex distributed mechanisms (used today in many networks) and separating state management from protocol design and implementation. It allows providing a single coherent global view of the network through an annotated network graph accessible for control via an API. An implementation of such an abstraction is a Network Operating System (NOS).
- *Specification (or Configuration) Abstraction*: This layer allows specifying the behavior of desired control requirements (such as access control, isolation, QoS, etc.) on the abstract network model and corresponds to the abstract network view as presented earlier.

Each one of these existing taxonomies focuses on a single specific aspect and we believe that none of them serves our purpose. Thus, we present in the following a hierarchical taxonomy that comprises three-levels: the SDN layer (or layers) of concern, the identified issues according to the SDN layer (or layers), and the proposed solutions in the literature to address these issues. In the following, we elaborate on our proposed taxonomy.

A. Infrastructure Layer

At this layer, the main issues identified in the literature are the performance and scalability of the forwarding devices as well as the correctness of the flow entries.

1) Performance and Scalability of the Forwarding Devices: To tackle performance and scalability issues at this layer, three main solution classes can be identified and they are described as follows:

- *Switches Resources Utilization*: Resources on switches such as CPU power, packet buffer size, flow tables size, and bandwidth of the control datapath are scarce and may create performance and scalability issues at the infrastructure layer. Works tackling this class of problems propose either optimizing the utilization of these resources, or modifying the switches hardware and architecture.
- *Lookup Procedure*: The implementation of the switch lookup procedure may have an important impact on the performance at the switch-level. A trade-off exists between using hardware and/or software tables since the first type of tables are expensive resources and the implementation of the second type of table may add lookup latencies. Works tackling this class of problems propose to deal with this trade-off.

2) Correctness of Flow Entries: Several factors may lead to problems of inconsistencies and conflicts within OpenFlow configurations at the infrastructure level. Among these factors, the distributed state of the OpenFlow rules across various flow tables and the involvement of multiple independent OpenFlow rules writers (administrators, protocols, etc.). Several approaches have been proposed to tackle this issue and the solutions can be classified as follows:

- *Run-Time Formal Verification*: In this thread, the verification is performed at run-time, which allows to capture the bugs before damages occur. This class of solutions is based on formal methods such as model checking.
- *Offline Formal Verification*: In this case, the formal verification is performed offline and the check is only run periodically.

B. Control Layer

As far as network control is concerned, the identified critical issues are performance, scalability, and reliability of the controller and the security of the control layer.

1) Performance, Scalability, and Reliability: The control layer can be a bottleneck of the SDN networks if relying on a single controller to manage medium to large networks. Among envisioned solutions, we can find the following categories:

- *Control Partitioning: Horizontal or Vertical*. In large SDN networks, partitioning the network into multiple controlled domains should be envisaged. We can distinguish two main types of control plane distribution [75]:
  - Horizontally distributed controllers: Multiple controllers are organized in a flat control plane where each one governs a subset of the network switches. This deployment comes in two flavors: with state replication or without state replication.
  - Vertically distributed controllers: It is a hierarchical control plane where the controllers' functionalities are organized vertically. In this deployment model, control tasks are distributed to different controllers depending on criteria such as network view and locality requirements. Thus, local events are handled to the controller that are lower in the hierarchy and more global events are handled at higher level.
- *Distributed Controllers Placement*: Distributed controllers may solve the performance and scalability issue, however,

they raise a new issue, which is determining the number of the needed controllers and their placement within the controlled domain. Research works in this direction aim at finding an optimized solution for this problem.

2) Security of the Controller: The scalability issue of the controller enables targeted flooding attacks, which leads to control plane saturation. Possible solutions to this problem is *Adding Intelligence to the Infrastructure*. The latter relies on adding programmability to the infrastructure layer, which prevents the congestion of the control plane.

C. Application Layer

At this layer, we can distinguish two main research directions studied in the literature: developing SDN applications to manage specific network functionalities and developing SDN applications for managing specific environments (called use cases).

1) SDN Applications: At this layer, SDN applications interact with the controllers to achieve a specific network function in order to fulfill the network operators needs. We can categorize these applications according to the related network functionality or domain they achieve including security, quality of service (QoS), traffic engineering (TE), universal access control lists (U-ACL) management, and load balancing (LB).

2) SDN Use Cases: From the other side, several SDN applications are developed in order to serve a specific use case in a given environment. Among possible SDN uses cases, we focus on the application of SDN in cloud computing, Information Content Networking (ICN), mobile networks, network virtualization, and Network Function Virtualization (NFV).

D. Control/Infrastructure Layers

In this part of the taxonomy, we focus on issues that may span control and infrastructure layers and the connection between them.

1) Performance and Scalability: In the SDN design vision of keeping data plane simple and delegating the control task to a logically centralized controller, the switches-to-controller connection tends to be highly solicited. This adds latency to the processing of the first packets of a flow in the switches' buffers but can lead to irreversible damage to the network such as loss of packets and palatalization of the whole network. In order to tackle this issue, we mainly found a proposal on *Control Load Devolving*. The latter is based on the delegation of some of the control load to the infrastructure layer to alleviate the frequency by which the switches contact the controller.

2) Network Correctness: The controller is in charge of instructing the switches in the data plane on how to process the incoming flows. As this dynamic insertion of forwarding rules may cause potential violation of network properties, verifying network correctness at run-time is essential in keeping the network operational. Among the proposed solutions we cite the use of *Algorithm for Run-time Verification* to deal with checking correctness of the network while inserting new forwarding rules. The verification involves checking network-wide policies and invariants such as absence of loops and reachability properties.

E. Application/Control Layers

Various SDN applications are developed by different network administrators to manage network functionalities by programming the controller's capabilities. Two main issues were examined in this context: Policy correctness and the northbound interface security threats represented by adversarial SDN applications.

1) Policy Correctness: Conflicts between OpenFlow rules may occur due to multiple requests made by several SDN applications. Different solutions are proposed for conflicts detection and resolution that can be classified as either approaches for *Run-time Formal Verification* using well-established formal methods or Custom *Algorithm for Run-time Verification*.

2) Northbound Interface Security: Multiple SDN applications may request a set of OpenFlow rule insertions, which may lead to the possible creation of security breaches in the ongoing OpenFlow network configuration. Among the envisaged solutions is the use of a *Role-based Authorization* model to assign a level of authorization to each SDN application.

F. Application/Control/Infrastructure Layers

The decision taken by the SDN application deployed at the application layer influence the OpenFlow rules configured at the infrastructure layer. This influence is directed via the control layer. In this part of the taxonomy, we focus on issues that concern all of the three SDN layers.

1) Policy Updates Correctness: Modification in policies programmed by the SDN applications may result in inconsistent modification of the OpenFlow network configurations. These changes in configurations are common source of network instability. To prevent such a critical problem, works propose solutions to verify and/or ensure consistent updates. Thus we enumerate two classes of solutions

- *Formal Verification of Updates*: Formal verification approaches, such as model-checking, are mainly used to verify that updates are consistent (i.e., updates preserve well-defined behaviors when transitioning between configurations).
- *Update Mechanism/Protocol*: This class of solutions proposes a mechanism or protocol that ensures that updates are performed without introducing inconsistent transient configurations.

2) Network Correctness: While the network correctness at the Control/Infrastructure layers is more about the newly inserted OpenFlow rules and the existing ones at the infrastructure layer, network correctness at Application/Control/Infrastructure concerns the policy specified by the applications and the existing OpenFlow rules. Among the proposed solutions is *Offline Testing*, which uses testing techniques to check generic correctness properties such as no forwarding loops or no black holes and application-specific properties over SDN networks taking into account the three layers.

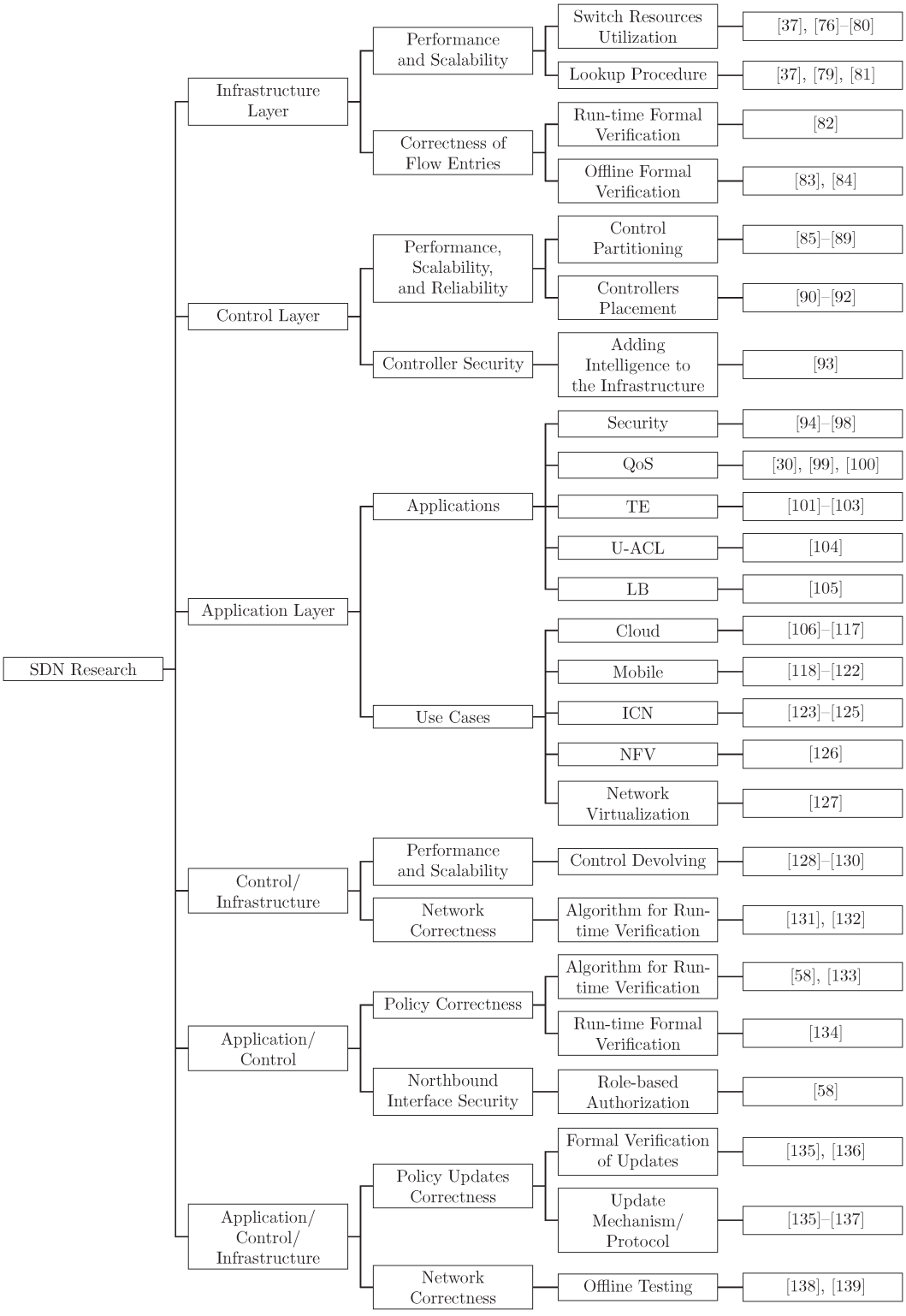

Fig. 4. Overview of the surveyed research works classified according to the proposed taxonomy.

V. SDN Issues and Research Directions

In this section, we present a survey on the most relevant research initiatives studying problematic issues raised by SDN and providing proposals towards supporting the adoption of SDN concepts in today's networks. The reviewed works are organized using our taxonomy: either belonging to a specific functional layer or concerning a cross-layer issue. We identified a set of most critical concerns that may either catalyze the successful growth and adoption of SDN or refrain its advance. These concerns are scalability, performance, reliability, correctness, and security. Fig. 4 provides an overview of the reviewed research work classified using our taxonomy.

TABLE IV
SURVEY ON INITIATIVES ADDRESSING PERFORMANCE AND SCALABILITY AT THE INFRASTRUCTURE LAYER

Reference	Addressed Issue	Proposed Solution	Required Modification
Kannan and Banerjee [76]	TCAM Flow tables size and power dissipation	Compact TCAM by replacing flow entries with shorter Flow-ID	Packets headers to carry the Flow-ID and a Flow-ID table at the controller site
Narayanan et al. [77]	Flow tables size, bus speed between the hardware pipeline and the on-chip CPU, and CPU power	Use multi-core programmable ASICs and a new switch architecture (Split SDN Data Plane) splitting the forwarding fast path into TCAM-based and software-based paths and use a high-speed bus	Modification to switches hardware and architecture
Lu et al. [78]	Packet buffer size, CPU power, and bandwidth between CPU and ASIC	Equipping switches with powerful CPUs and gigabytes DRAM and setup a high-bandwidth internal link between the CPU and the ASIC	Modification to switches hardware and architecture
Tanyingyong et al. [81]	Software-based lookup procedure	Use a standard commodity Network Interface Card (NIC) and achieve fast decision path by caching flow table entries on the NIC	Modification to switches hardware and architecture
Luo et al. [79]	CPU power and Lookup procedure	Implemented a network processor-based acceleration card and an OpenFlow switching algorithm	Modification to switches hardware and architecture
Naous et al. [37]	Flow tables size, CPU power, and Lookup procedure	Implementation of a full line-rate OpenFlow switch on NetFPGA	Modification to switches hardware and architecture
Kang et al. [80]	Flow tables size	Algorithm for rules placement that distributes forwarding policies and supports dynamic incremental update of policies	Rule placement algorithm within the controller to satisfy switch table-size constraints

A. Infrastructure Layer

1) Performance and Scalability: Despite the undeniable advantages brought by SDN since its inception, it has introduced several concerns including scalability and performance. These concerns stem from various aspects including the implementation of the OpenFlow switches. The most relevant switch-level problems that limit the support of the SDN/OpenFlow paradigm are as follows:

- ***Flow Tables Size***. The CAM is a special type of memory that is content addressable. CAM is much faster than RAM as it allows parallel lookup search. A TCAM, for ternary CAM, can match 3-valued inputs: '0', '1' and 'X' where 'X' denotes the "don't care" condition (usually referred to as wildcard condition). Thus, a TCAM entry typically requires a greater number of entries if stored in CAM. With the emergence of SDN, the volume of flow entries is expected to grow several orders higher than in traditional networks. This is due to the fact that OpenFlow switches rely on a fine grained management of flows (microflows) to maintain complete visibility in a large OpenFlow network. However, TCAM entries are a relatively expensive resource in terms of ASIC area and power consumption.
- ***Lookup Procedure***. Two types of flow tables exist: hash table and linear table. Hash table is used to store microflows where the hash of the flow is used as an index for fast lookups. The hashes of the exact flows are typically stored in Static RAM (SRAM) on the switch. One drawback of this type of memory is that it is usually off-chip, which causes lookup latencies. Linear tables are typically used for storing macroflows and are usually implemented in TCAM, which is most efficient to store flow entries with wildcards. TCAM is often located on the switching chip, which decreases lookup delays. In ordinary switches, lookup mechanism is the main operation that is performed, whereas in OpenFlow-enabled switches, other operations are considered, especially the "insert" operation. This can lead to a higher power dissipation and a longer access latency [76] than in regular switches.
- ***CPU Power***. For a purely software-based OpenFlow switch, every flow is handled by the system CPU and thus, performance will be determined by the switches' CPU power. Furthermore, CPU is needed in order to encapsulate the packet to be transmitted to the controller for a reactive flow setup through the secure channel. However, in traditional networks, the CPU on a switch was not intended to handle per-flow operations, thus, limiting the supported rate of OpenFlow operations. Furthermore, the limited power of a switch CPU can restrict the bandwidth between the switch and the controller, which will be discussed in the cross-layer issues.
- ***Bandwidth Between CPU and ASIC***. The control datapath between the ASIC and the CPU is typically a slow path as it is not frequently used in traditional switch operation.
- ***Packet Buffer Size*** The switch packet buffer is characterized by a limited size, which may lead to packet drops and cause throughput degradation.

Various research works [37], [76]–[81] addressed one or more of these issues to improve performance and scalability of SDN data plane, and specifically of OpenFlow Switches. These works are summarized in Table IV.

TABLE V
SURVEY ON INITIATIVES ADDRESSING CORRECTNESS ISSUES

Reference	Layer	Goal	Approach	Type of verification
Al-Shaer and Al-Haj [82] (FlowChecker)	Infrastructure	Flow tables in OpenFlow switch networks against properties expressed in temporal logic	Symbolic model-checking over BDDs	Run-time verification
McGeer [83]	Infrastructure	Flow tables in OpenFlow switch networks against network properties	Formal verification: Solving satisfiability problems over networks of logic functions	Offline verification
Skowyra et al. [84]	Infrastructure	OpenFlow learning switch networks between mobile end-hosts against network correctness, network convergence, and mobility-related properties	Formal verification of the design using Verificare tool againts a library of requirements using model-checkers	Offline design verification
Khurshid et al. [131] (VeriFlow)	Control-Infrastructure	Checks for network-wide invariant violations using existing flow table rules at each forwarding rule is inserted by the controller	Algorithm: searching for overlapping rules using equivalence classes of packets	Run-time verification
Kazemian et al. [132] (NetPlumber)	Control-Infrastructure	Checks for network-wide policies and invariants on every event using existing flow table rules	Algorithm: Header space analysis of forwarding rules	Run-time verification
Porras et al. [58] (FortNox)	Application-Control	Conflicts detection and resolution between newly OpenFlow rules inserted by applications vs. existing OpenFlow rules	Algorithm: Alias set rule reduction for detecting rule contradictions	Run-time verification
Wang et al. [133]	Application-Control	Conflicts detection and resolution between flow table rulesets and SDN firewall applications rules	Algorithm: Header space analysis for detecting and solving conflicts	Run-time verification
Son et al. [134] (FLOVER)	Application-Control	Compliance of updated flow table rulesets with respect to non-bypass security properties	Formal verification using an SMT solver	Run-time verification
Reitblatt et al. [135], [136] (Kinetic)	Application-Control-Infrastructure	Invariance of trace properties over updated policies	A mechanism for consistent update and a formal verification using a model-checker	Offline verification
McGeer [137]	Application-Control-Infrastructure	Consistent updates of the programmed policies	A protocol for OpenFlow rule updates to ensure per-packet rule-set consistency	-
Canini et al. [138] (NICE)	Application-Control-Infrastructure	Testing the behavior of controller and its applications integrated wit an abstract model of the switches	State-space search using symbolic execution and custom explicit state model-checking	Offline Testing
Kuzniar et al. [139] (OFTEN)	Application-Control-Infrastructure	Testing the behavior of controller with its applications integrated with real OpenFlow switches	Based on NICE [138] synchronized with state of real OpenFlow switches	Offline testing

2) Correctness of Flow Entries: More than half of network errors are due to misconfiguration bugs [140]. Misconfiguration has a direct impact on the security and the efficiency of the network because of forwarding loops, content delivery failure, isolation guarantee failure, access control violation, etc. Skowyra *et al.* [84] propose an approach based on formal methods to model and verify OpenFlow learning switches network with respect to properties such as network correctness, network convergence, and mobility-related properties. The verified properties are expressed in LTL and PCTL* and both SPIN and PRISM model-checkers are used. McGeer [83] discusses the complexity of verifying OpenFlow networks. Therein, a network of OpenFlow switches is considered as an acyclic network of high-dimensional Boolean functions. Such verification is shown to be NP-complete by a reduction from SAT. Furthermore, restricting the OpenFlow rule set to prefix rules makes the verification complexity polynomial. FlowChecker [82] is another tool to analyze, validate, and enforce end-to-end OpenFlow configuration in federated OpenFlow infrastructures. Various types of misconfiguration are investigated: intra-switch misconfiguration within a single FlowTable as well as inter-switch or inter-federated inconsistencies in a path of OpenFlow switches across the same or different OpenFlow infrastructures. For federated OpenFlow infrastructures, FlowChecker is run as a master controller communicating with various controllers to identify and resolve inconsistencies using symbolic model-checking over Binary Decision Diagrams (BDD) to encode OpenFlow configurations. These works are compared in Table V.

B. Control Layer

1) Performance, Scalability, and Reliability: Concerns about performance and scalability have been considered as major in SDN since its inception. The most determinant factors that impact the performance and scalability of the control

plane are the number of new flows installs per second that the controller can handle and the delay of a flow setup. Benchmarks on NOX [32] showed that it could handle at least 30.000 new flow installs per second while maintaining a sub 10-ms flow setup delay [142]. Nevertheless, recent experimental studies suggest that these numbers are insufficient to overcome scalability issues. For example, it has been shown in [143] that the median flow arrival rate in a cluster of 1500 servers is about 100.000 flows per second. This level of performance, despite its suitability to some deployment environments such as enterprises, leads to raise legitimate questions on scaling implications. In large networks, increasing the number of switches results in augmenting the number of OpenFlow messages. Furthermore, networks with large diameters may result in an additional flow setup delay. At the control layer, the partitioning of the control, the number and the placement of controllers in the network, and the design and implementation choices of the controlling software are various proposals to address these issues.

The deployment of SDN controller may have a high impact on the reliability of the control plane. In contrast to traditional networks where one has to deal with network links and nodes failures only, SDN controller and the switches-to-controllers links may also fail. In a network managed by a single controller, the failure of the latter may collapse the entire network. Moreover, in case of failure in OpenFlow SDN systems, the number of forwarding rules that need to be modified to recover can be very large as the number of hosts grows. Thus, ensuring the reliability of the controlling entity is vital for SDN-based networks.

As for the design and implementation choices, some works suggest to take benefit from the multi-core technology and propose multi-threaded controllers to improve their performance. Nox-MT [46], Maestro [48] and Beacon [49] are examples of such controllers. Tootoonchian *et al.* [46] show through experiments that multi-threaded controllers exhibit better performance than single-threaded ones and may boost the performance by an order of magnitude.

In the following, we focus first on various frameworks proposing specific architectures for partitioning the control plane and then discuss works proposing solutions to determine the number of needed controllers and their placement in order to tackle performance issues.

Control Partitioning: Several proposals [85]–[89] suggest an architectural-based solution that employs multiple controllers deployed according to a specific configuration. Hyperflow [85] is a distributed event-based control plane for OpenFlow. It keeps network control logically centralized but uses multiple physically distributed NOX controllers. These controllers share the same consistent network-wide view and run as if they are controlling the whole network. For the sake of performance, HyperFlow instructs each controller to locally serve a subset of the data plane requests by redirecting OpenFlow messages to the intended target. HyperFlow is implemented as an application over NOX [32] and it is in charge of proactively and transparently pushing the network state to other controllers using a publish/subscribe messaging system. Based on the latter system, HyperFlow is resilient to network partitions and components failures and allows interconnecting independently managed OpenFlow networks while minimizing the cross-region control traffic.

Onix [86] is another distributed control platform, where one or multiple instances may run on one or more clustered servers in the network. Onix controllers operate on a global view of the network state where the state of each controller is stored in a Network Information Base (NIB) data structure. To replicate the NIB over instances, two choices of data stores are offered with different degrees of durability and consistency: a replicated transactional database for state applications that favor durability and strong consistency and a memory-based one-hop Distributed Hash table (DHT) for volatile state that is more tolerant to inconsistencies. Onix supports at least two control scenarios. The first is horizontally distributed Onix instances where each one is managing a partition of the workload. The second is a federated and hierarchical structuring of Onix clusters where the network managed by a cluster of Onix nodes is aggregated so that it appears as a single node in a separate cluster's NIB. In this setting, a global Onix instance performs a domain-wide traffic engineering. Control applications on Onix handle four types of network failures: forwarding element failures, link failures, Onix instance failures, and failures in connectivity between network elements and Onix instances as well as between Onix instances themselves.

The work in [87] proposes a deployment of horizontally distributed multiple controllers in a cluster of servers, each installed on a distinct server. At any point in time, a single master controller that has the smallest system's load is elected and is periodically monitored by all of the other controllers for possible failure. In case of failure, master reelection is performed. The master controller dynamically maps switches to controllers using IP aliasing. This allows dynamic addition and removal of controllers to the cluster and switch migration between controllers and thus deals with the failure of a controller and a switch-to-controller link.

These aforementioned frameworks allow reducing the limitations of a centralized controller deployment but they overlook an important aspect, which is the fact that not all applications require a network-wide state. In SDN, it belongs to the controller to maintain a consistent global state of the network. Observation granularity refers to the set of information enclosed in the network view. Controllers generally provide some visibility of the network under control including the switch-level topology such as links, switches, hosts, and middelboxes. This view is referred to as a partial network-wide view, whereas a view that accounts for the network traffic is considered as a full network-wide view. Note that a partial network-wide view changes at a low pace and consequently it can be scalably maintained. Based on this observation, Kandoo [88] proposes a two-layered hierarchy; A bottom controllers layer, closer to the data plane, that have neither interconnection nor knowledge of the network-wide state. They run only local control applications (i.e., functions using the state of a single switch), handle most of the frequent events, and effectively shield the top layer. The top layer runs a logically centralized controller (root controller) that maintains the network-wide state and thus runs applications requiring access to this global view. It is also used for

TABLE VI
SDN CONTROL FRAMEWORKS

Proposal	Partitioning	Cross-Controllers Coordination	Observation Gran.	Implementation	Failure Recovery
HyperFlow [85]	Horizontal with state replication	Publish/subscribe messaging	Full	Application within NOX [32]	Network partitions and component failure
Onix [86]	Horizontal with state replication or Vertical	Replicated NIB in a replicated transactional database and a DHT	Full	Multiple Onix controllers (cluster) running on a set of physical servers	switch, link, Onix instance, switch-to-Onix link, and Onix-to-Onix link
Yazıcı et al. [87]	Horizontal with state replication	JGroups notifications and messaging	Full	Multiple controllers (cluster with a master controller) each on a distinct server	controller and switch-to-controller link
Kandoo [88]	Vertical (2 levels): a centralized controller (may be horizontal with state replication) at the top layer and a horizontal deployment with no state replication at the the bottom layer	Depending on the top layer	Partial (bottom layer) Full (top layer)	Bottom layer controller can be implemented on a commodity middlebox as switch proxy or directly on OpenFlow switches	None
SiBF [89]	Horizontal with no state replication	KeySpace, Distributed NoSQL database integrated with each RM	–	Each RMs is a standalone application running in one of the rack servers	Server and network failures
CPRecovery [141]	Centralized	Primary to secondary controller communication (for state replication)	Full	Application within NOX [32]	Controller failure
FlowVisor [13], [30], [31]	Horizontal with no state replication	None	Partial (Network-slice view for each controller)	A software slicing layer between the forwarding and control planes housed outside the switch	None

coordinating between local controllers, if needed. The root top layer controller can be a single controller or horizontally distributed controller such as Onix [86] or HyperFlow [85]. The work in [141] proposes a solution based on the primary-backup technique to increase control plane resiliency to failure in the case of a centralized architecture. A component, namely CPRecovery, is designed to provide a seamless transition between a failure state and a backup, consistent with the latest network's failure-free state. In this solution, one or more backup secondary controllers are maintained consistent with respect to the last consistent state of the primary controller. In case of failure of the primary controller, one of these secondary controllers is elected to shoulder smoothly the control tasks from the last valid state. This solution adds to the OpenFlow protocol, which provides the means to configure one or more backup controllers, a coordination mechanism between the primary controller and the backup ones. Replicating the controller allows to overcome the controller failure issue but does not solve the scalability concern.

FlowVisor [13], [30], [31] slices the network in order to allow multiple network's tenants to share the same physical infrastructure. It acts as a transparent proxy between OpenFlow switches and various guest network operating systems. Table VI summarizes and compares these aforementioned works on control partitionning.

The work in [144] addresses the problem of overloaded controllers due to statically configured mapping between switches and controllers in a distributed SDN controllers environment. This is the case for instance for HyperFlow [85] and Onix [86]. A controller may become overloaded if the switches statically attached to it suddenly observe a large number of flows, while other controllers remain underutilized. To solve this problem Dixit *et al.* [144] propose an elastic distributed controller architecture, namely ElastiCon, where new controller instances may be added or removed from a pool and a proposed protocol dynamically assigns switches to controllers according to traffic conditions change over time. The proposed protocol is designed to minimally affect the normal operation of the network while guaranteeing consistency and reliability.

Controllers Placement: As explained above, distributed controller deployments allow to alleviate performance, scalability, and reliability problems. However, they introduce other new concerns that need to be investigated and addressed. The most relevant ones are the number and placement of the controllers and the global state inconsistency.

In [90], the problems of deciding on the number of controllers to use and their placement within the topology are discussed. The efficiency of various proposed placement solutions is compared based on the switch-to-controller latency, which is one of the most important performance metrics, especially

in wide area networks. They concluded that there is no general placement rules that apply to every network. Rather, the number of controllers to be used and their placement depend on the network topology on the one side, and the trade-off between the availability and the performance metrics fixed by the network operator on the other side. Hu *et al.* [91] propose algorithms to automate the placement decisions given a physical network, the number of controllers, and a pre-defined objective function to optimize. The main objective of these algorithms is to maximize resiliency of SDN to failures. In [92], a study has been conducted on how the physically distributed control plane state impacts the performance of the logically centralized control applications. Two trade-offs have been studied through experiments on load balancing applications: the trade-off between the application performance and the state distribution overhead, and the trade-off between the robustness to inconsistency and the complexity of the control applications. The study concluded that inconsistency has a significant impact on control applications, which means that the application logic should be aware of the state distribution, like in Onix, for a better performance.

Even though performance and scalability issues have been always coined to SDN, [145] arguments that these concerns are neither caused by nor fundamentally unique to SDN and they can be addressed without losing the benefits of SDN.

2) Security: Shin *et al.* [93] propose AVANT-GUARD, a new framework that tackles SDN security issues by adding intelligence to the OpenFlow data plane. AVANT-GUARD enables the control plane and the SDN network to be more resilient and scalable against control plane saturation attacks such as TCP-SYN flood. It articulates around two modules. The first is a connection migration module that allows increasing the Open-Flow network resilience and scalability by preventing from saturation attacks (spoofed and non-spoofed flooding attacks). This is achieved by inspecting the TCP sessions at the data plane before notifying the controller. The second module, called the actuating trigger, enables the controller to push into the data plane rules for detecting and responding to the observed threats. This is realized through a more efficient collection of network statistics and by automatically setting specific flow rules under some predefined conditions. To implement AVANT-GUARD, OpenFlow switch specification needs to be extended with new adequate commands to handle modules operations.

C. Application Layer

1) SDN Applications: The SDN paradigm allows multiple network administrators to govern various network flows through their own services and applications. In the following, we focus on some specific services, namely security, QoS, traffic engineering, universal ACL management, and load balancing. For each service, we show how SDN overcomes issues existing in traditional networks.

Security: Thanks to its capacity to provide a global network state observation, a logically centralized control intelligence, and a standardized programmability of network devices, SDN constitutes a great opportunity to design and deploy novel effective solutions or to improve on existing ones in order to address network security issues. For instance, in traditional networks, Anomaly Detection Systems (ADS) are generally deployed in the network core of the service provider in order to protect home and office networks from security threats. Mehdi *et al.* [94] propose to bring ADS from network core to home networks using SDN. They devise a framework composed of four anomaly detection algorithms implemented as application for the controller NOX. They showed that these algorithms allow applying a higher accurate security policing when deployed in SDN home networks compared to a network core deployment.

Braga *et al.* [95] propose a security application to detect Distributed Denial of Service (DDoS) attacks for NOX controllers. In their approach, the controller retrieves the information needed about the traffic flow features that are specific to DDoS attacks including average packets per flow, average bytes per flow, and growth of single flows. Then, this information is processed by the Self-Organizing Map (SOM) mechanism, an artificial neural network, to identify abnormal traffic by classifying it either as normal traffic or an attack-related traffic. It has been proved that the technique achieves a high rate of detection and a low rate of false alarms. Jafarian *et al.* [96] define a countermeasure against network scanning based on SDN. They propose a technique for IP address mutation based on a central management authority. This would prevent attackers from making the right hypotheses about IP addresses assignment in the network. The strength of the technique lies in the fact that monitoring in an SDN network is centralized, which allows efficiently coordinating IP mutation across OpenFlow switches with minimal overhead.

FleXam [98] is an OpenFlow extension that implements probabilistic and deterministic per-flow sampling techniques. This extension allows the controller accessing useful packet-level information while keeping a minimum overhead. The pulled information can be used to implement efficient network security OpenFlow controller applications, that need packet-level information to perform properly, such as port scan detection, worm detection and botnet detection. Giotis *et al.* [97] propose a real-time, modular and scalable OpenFlow-based anomaly detection architecture. The solution is constituted of three main modules. The collector module gathers data based on sFLow [146], a flow monitoring mechanism utilizing packet sampling. The output of the collector module is periodically fed to the anomaly detection module in order to identify potential attacks. As soon as an anomaly is detected, specific network metrics are inspected, correlated and exposed to the mitigation module. The latter issues and installs appropriate flow entries to neutralize the identified attacks by blocking the malicious traffic. It has been shown that the proposed solution successfully detects DDoS attacks, worm propagation, and port scan attacks.

Quality of Service: Quality of service provisioning has been for a long time a chronic issue in traditional networks and a source of operating expenses and risk that is based on delivering differentiated types of quality to network end-users. This is partly due to the closed interface of networking equipments, the completely distributed hop-by-hop routing architecture of

the Internet and to the lack of a synthetic picture of the overall network and resources. SDN can solve several network QoS challenges by offering a complete network visibility and opening networking devices to programmability. Only few papers focus on such an issue. Egilmez *et al.* [99] propose dynamic QoS routing mechanism called OpenQoS that is specifically designed for delivering multimedia traffic with QoS. OpenQoS was implemented for the Floodlight controller. In addition to preliminary research initiatives [30], [100], the ONF is actively enhancing the QoS support in OpenFlow. Historically, only experimental QoS support has been introduced in OpenFlow version 0.8. In OpenFlow 1.0 [43], packets can be forwarded to queues of output ports by the optional enqueue action that is renamed to set_queue in version 1.3 [44]. Therein, the behavior of the queue is determined outside the scope of OpenFlow. Complex QoS support was introduced in OpenFlow 1.3 with meter tables that consist of entries defining per-flow meters. These meters enable OpenFlow to implement various simple QoS operations, such as rate-limiting. These meters can be combined with the optional set_queue action, which associates a packet to a per-port queue in order to implement complex QoS frameworks, such as DiffServ [44].

Traffic Engineering: Traffic engineering or traffic management is a method for dynamically analyzing, regulating, and predicting the behavior of data flowing in networks with the aim of performance optimization to meet Service-Level Agreements (SLAs). Traffic engineering in traditional networks is still a challenging task since it is based on the deployment of excessively expensive infrastructures. SDN offers both the complete visibility of the network-wide view and the ability to externally program network devices by dynamically provisioning forwarding tables. This allows choosing the most efficient paths based on application requirements and on real-time information. Centralized traffic engineering in WAN using SDN is already adopted by Google [101]. As a first step towards achieving traffic engineering through SDN, authors in [102] explored an SDN-based integrated network control architecture for configuring and optimizing the network to fit better to big data applications requirements, using dynamically configurable optical circuits. However, flow-level traffic engineering for big data applications has been postponed for future work. In [103], capabilities of SDN/OpenFlow for WAN traffic engineering have been outlined and demonstrated through a network application.

Universal ACL Management: Network policy updates require device-level configuration across heterogeneous elements (switches, routers, firewalls) by human operators. This is time-consuming, error-prone, and costly. SDN allows network administrators to perceive network devices as a unique abstract switch and to create universal access policies that will be further spread over a complex network topology with a single command. In this direction, authors in [104] motivate the objective of building machinery for global policy transformation by moving, merging, and/or splitting rules across multiple switches, while preserving the overall forwarding behavior of a network.

Load Balancing: To cope with the increasing traffic load of online services such as social networks, services are replicated over multiple servers. Then, a load balancer is typically used to split clients' requests among these servers. However, load balancers are expensive hardware with a rigid policy set, and they constitute a single point of failure. Wang *et al.* [105] propose a costless and more flexible OpenFlow-based alternative. In this solution, traffic is allocated to servers by switches based on the flow tables installed by the controller. They devise an algorithm that figures out the concise placement of wildcards in flow table entries to achieve the adequate forwarding granularity for a scalable solution.

2) SDN Use Cases: Many works proposed to adopt SDN to different application domains such as cloud computing, mobile networks, and information content networking. In the following, we review the most prominent ones.

Cloud Computing: Cloud data centers are required to be large in scale, cost-effective, easy to operate and offer a reactive on-demand network configuration management. However, cloud data center based on traditional networks are still facing problems to meet these requirements. Indeed, they are very expensive to setup and their resources are underused [143], which increases their operating costs. Moreover, current network APIs have very limited expressiveness and network switches are designed to operate autonomously with static configurations and minimum administrative intervention. This prohibits transferring the information between the required network functionality and the capabilities of commodity network devices.

Various research works have revisited these concerns using SDN and OpenFlow. For instance, Matias *et al.* [106] explore the virtualization of the physical network using OpenFlow in order to enable the presence of multiple cloud operators sharing a common infrastructure. This provides a virtualization framework that manages resources in a similar way to FlowVisor [13], [30], [31]. Lei *et al.* [107] propose an OpenFlow virtualized network architecture for large-scale multi-tenant data centers and use the REST API to support the on-demand network management and configuration. Rotsos *et al.* [108] present an event-driven OpenFlow controller library built on top of Mirage platform [147] that allows cloud applications to exercise flow-level control over the forwarding process. Direct exposure of network capabilities to the applications effectively distributes control among all entities sharing the network infrastructure, which contradicts the current data center model where the service provider is firmly in charge of the control hierarchy.

The emergence of SDN provides an opportunity to leverage the cloud networking feature through the programmable interfaces and the flow-based access control. A joint orchestration of computing and storage with networking resources is seen as one of the most important applications for a software-defined network. Cloud orchestration requires the interworking of an SDN controller and a cloud computing controller, such as OpenStack [109]. The latter is an open source infrastructure-as-a-service cloud platform that provides Network-as-a-Service cloud functionality through the recently added component, Quantum. This service endows tenants with the capability of creating and controlling their own virtual networks. Quantum has an extendable, API-driven and pluggable architecture with networks, subnets, IP addresses and ports manipulation and

access control features. OpenStack supports the SDN controller Ryu and communicates with it using the Quantum Rest API [55]. Meridian [110], is another SDN-based controller framework that has been proposed for cloud networking. It has been implemented on top of the Floodlight controller [52] and it articulates around three logical layers: The network abstraction and APIs layer exposes to the network control applications the information needed to interact with the network. The network orchestration layer performs logical-to-physical translation of commands issued through the abstraction layer and provides the global network view and state. Finally, the third layer consists of interfaces that allow to interact with the various underlying network devices.

As per security in the cloud, Stabler *et al.* [111] propose an implementation of an elastic IP and security group service, similar to the Amazon EC2 services, using the OpenFlow protocol and the OpenNebula system. The implementation relies on the integration of the OpenFlow controller NOX with the EC2 server. Flow rules are inserted in the controller using the EC2 API and then used by Open vSwitches on the underlying hypervisor to manage network traffic. This implementation enables customizable network services operated by the end users through API calls. Koorevaar [112] proposes an approach for leveraging SDN architecture to automatically enforce security policy using a mechanism called Elastic Enforcement Layer tags (EEL-tags) by adding meta-data to the packet in order to route it to the next middlebox instance. This is done by enabling the hypervisor to add an application Id Tag into the flow of packets emitted by the VM, which allows identifying the security policies that actually refer to a chain of middleboxes to be traversed by the secured traffic.

NICE [117] is a defense intrusion detection framework for virtual network systems. It is based on two components: NICE-A, a network intrusion detection engine installed in each cloud server, and a centralized control center which is mainly constituted of an attack analyzer and an OpenFlow network controller. After detecting suspicious or abnormal traffic, NICE-A sends alerts to the central control center through a secure channel. Afterward, the attack analyzer evaluates the received alerts based on an attack graph, and decides of the countermeasure to apply. Then, the network controller reconfigures the OpenFlow switches in a way to establish the newly defined countermeasure. In [116], authors propose SnortFlow a comprehensive intrusion prevention system for cloud virtual network environments. SnortFlow combines the benefit of Snort, the multi-mode packet analysis tool, with the power of the OpenFlow programmable infrastructure. This solution is mainly based on the SnortFlow server component, which actively collects alert data from Snort agents running on cloud servers, evaluates the network security status and issues actions to be pushed to the OpenFlow controller in order to reconfigure the network tasks accordingly.

As far as VM mobility is concerned, both live and offline VM migration provide an important level of flexibility, workload balancing, availability, fault tolerance and bring down enterprises OPEX costs. However, VM migration techniques are still facing several challenges. For example, they are limited to local networks mainly because of the hierarchical addressing used by layer 3 routing protocols. Recent works have made use of SDN and OpenFlow controllers to overcome these limitations. CrossRoads [113] and VICTOR [114], for instance, are OpenFlow-based systems proposed for seamless VM migration across data centers. VICTOR supposes that data centers are managed by a unique controller while CrossRoad suggests that each data center is governed by its own controller. In [115], authors take another direction by defining a Infrastructure-as-a-Service (IaaS) software middleware solution based on OpenFlow to achieve interoperability between data centers with different network topologies.

Information Content Networking: Information Content Networking (ICN) or Named Data Networking (NDN) is a recently emerging network paradigm that proposes an alternative for the current IP-based networks. It focuses on the content itself rather than on hosts or connection channels that are carrying this content. ICN is considered as one of the major characteristics of the future Internet [148] as it introduces features such as content-based services and efficient network caching. Among current ICN research projects we can cite CONVERGENCE [149], SAIL [150] and PURSUIT [151].

Deploying ICN on traditional networks, however, turns out to be a challenging issue since running equipments need to be updated or replaced with ICN-enabled devices. Moreover, ICN aims at shifting the delivery fashion form host-to-user to content-to-user. Consequently, there is a need for a clean separation between the task of content controlling (information demand and supply) and the task of forwarding. In this context, SDN seems to be a key enabling technology for deploying ICN since it offers the programmability feature and the separation between the control and the forwarding plane. Combining SDN and ICN has already gained considerable attention among the research community, and many papers have been proposed to discuss issues related to supporting ICN using SDN concept [123]–[125]. Melazzi *et al.* [123] propose CONET, a framework for deploying ICN functionalities over SDN that leverage OpenFlow to better fit to ICN requirements. Syrivelis *et al.* [124] tackle technical issues related to combining ICN and SDN. Therein, a mapping of the notion of "flow" from SDN to ICN is proposed. According to [124], combining SDN to ICN would raise interesting business strategic questions in addition to the technical ones, since it will attract not only Internet and networking players but also a wide variety of industries. Chanda *et al.* [125] describes a content centric network architecture and propose a mechanism to observe and extract content metadata at the network layer used to optimize the network behavior. However, some modifications to the OpenFlow protocol are necessary to support the proposed mechanisms.

Mobile Networks: In recent years, mobile environments have become commonplace and mobile traffic has exponentially increased and it is even more expected to increase. Consequently, mobile environments are becoming a prevalent part of the Internet. Moreover, mobile users are permanently requiring new services with high-quality and efficient content-delivery independently of their location. A lot of mobility management solutions exist in the literature, but in order to get sound validations over these propositions, researchers need a long time to simulate wireless channels and to emulate

traffic and movement of users in a realistic way. Moreover, realizing back-hauling between heterogeneous technologies is hard to achieve. As the first goal of SDN is to enhance and to speed up the development of new solutions, many works have already studied its applicability to wireless and heterogeneous networks. For example, in [118], authors explore the use of SDN in heterogeneous (infrastructure and infrastructureless-based) networks. They discuss some application scenarios and research challenges in the context of "Heterogeneous SDN" (H-SDN), the case of SDN in heterogeneous networks. Yap *et al.* [119] propose OpenRoad or OpenFlow wireless to support a flexible wireless infrastructure. OpenRoad is an adaptation of OpenFlow to the framework of wireless networks. An OpenFlow-based architecture for mesh networks is proposed in [120].

In [121], authors discuss how SDN could enable better solutions for major challenges in cellular networks (i.e., Long Term Evolution (LTE)) and how it could be of a great usefulness for operators by giving them a greater control over their equipment, by simplifying network management and by introducing value-added services. Therein, extensions to controller platforms, network switches, and base stations are presented to enable software-defined cellular networks. Bansal *et al.* [122] propose OpenRadio, a platform that is quite close to OpenFlow in the sense that it aims at achieving a programmable data plane in cellular networks. OpenRadio focuses on decoupling wireless protocols' definitions from the hardware, then, it proposes a software abstraction layer that exposes a modular and declarative interface to program these protocols remotely in the base stations. Designing a control plane for cellular SDN infrastructure still needs to be addressed. In summary, SDN and OpenFlow are becoming a key enabling technology for mobile networks and research in this field is still in its infancy. Many challenges need to be addressed including security and interoperability between different SDN domains, more precisely between mobile and fixed domains.

Network Virtualization: A Virtual Network (VN) or a network virtualization environment is a subset of the underlying physical network. It consists of a set of virtual nodes connected through virtual links. The main goal of network virtualization is to realize the coexistence of heterogeneous network architectures, with eventually conflicting purposes, in isolation from each other within the same substrate. Sharing the same infrastructure and supporting logical network topologies that differ from the physical network are two common concepts to programmable networks and network virtualization [1], [152]. Network virtualization enables a flexible and independent implementation and management for each VN. Furthermore, it facilitates introducing customized network protocols and management policies. It also provides means to implement performance, QoS, and isolation, which allows minimizing the impact of network security threats [152]. Various surveys [152]–[154] have been recently published in the context of network virtualization.

SDN is not a new network virtualization technique but is more an enabling tool or technique that can be used in order to create virtualized networks [154]. Among relevant works on enabling network virtualization using SDN technology, FlowN [127] is a virtualization solution that provides programmable control over a network of switches where each tenant has the illusion of having its own address space, topology, and controller. The approach leverages database technology to efficiently store and manipulate mappings between virtual networks and physical switches.

Network Function Virtualization: Network Function Virtualization (NFV) [126] is an industry specification group that was formed by several network service providers (AT&T, BT, Deutsche Telekom, Orange, Telecom Italia, Telefonica and Verizon) under the European Telecommunications Standards Institute (ETSI). NFV aims at leveraging the standard IT virtualization technology in a way that decouples network functions from proprietary hardware appliances. It involves the implementation of mobile and fixed network functions such as firewalling, signaling, intrusion detection, and DNS, in software. The implemented virtual appliances are hence designated to be executed on different, yet standardized, environments provided by different network vendors for different network operators. Applying NFV is susceptible to bring several benefits to the telecommunication industry:

- It allows reducing development time and costs for deploying new services in order to meet the emerging business requirements, which in turn, lowers the associated risks.
- It allows services to be scaled up and down based on the actual requirements by a simple remote software provisioning.
- It opens the market to many different players to create new virtual appliances without significant risks, which encourages innovation.

Though being complementary and sharing two main objectives, namely, openness and innovation, NFV and SDN are two independent paradigms. That is to say, NFV can be implemented without separating the control plane from the data plane as suggested by SDN. However, an NFV infrastructure with SDN support is perfectly conceivable. Even better, it is expected that such an alignment would engender a greater value to NFV, since SDN enhances compatibility, eases maintenance procedures, and provides support for standardization.

D. Control/Infrastructure Layers

Examining the interconnection between the data and the control Layers, two important dimensions have been investigated in the literature: Performance and scalability of the southbound interface and its correctness.

1) Performance and Scalability: One of the most important design goals of OpenFlow was to keep the data plane simple and to delegate the control task to a logically centralized controller. As a result, switches have to consult the controller frequently for instructions on how to handle incoming packets of new flows. This tends to congest switch-controller connections, which in turn adds latency to the processing of the first packets of a flow in the switch buffer. Solutions to devolve a part of the control load to be processed by the data plane have been proposed. For instance, Devoflow [128] design goal is to keep flows in the data plane as much as possible by redistributing as many decisions as possible to the switches

and maintaining a useful amount of visibility on the flows for a central control. This can be done by minimizing the need for frequent invocation of the OpenFlow controller for flow setup and statistics gathering. Mainly, only detected significant (elephant or long-lived) flows are managed by the controller while short-lived ones are handled in the datapath. Despite the benefit of reducing switch-controller network bandwidth consumption, this approach requires a modification of the OpenFlow model. DIFANE [129] proposal is to split pre-installed OpenFlow wildcard rules among multiple switches, called authority switches, in a way that ensures all decisions can be made in the data plane. Thus, the controller has only to generate the flow entries and then partition them over the switches. Mahout [130] is a new traffic management system proposed to eliminate the need of per-flow monitoring in the switches using a combination of elephant flows detection at end-hosts and in-band signaling to the controller. The approach incurs low overhead and requires few switch resources, however implies the modification of end-hosts. In the aim of evaluating OpenFlow systems' performance, the work in [155] describes a queuing theory-based performance model of a preliminary OpenFlow architecture (constituted of one OpenFlow switch connected to a controller). In this model, the OpenFlow switch and controller are abstracted as a forwarding queue system and a feedback queue system, respectively. The work concluded that the packet sojourn time depends mainly on the processing speed of the controller and the probability of new flow arrivals.

Although addressing a critical issue in SDN architecture, these solutions have the drawback of requiring a modification (either to the OpenFlow protocol, the switches, or the end-hosts) and comes at the cost of flow visibility in the control plane.

2) Network Correctness: VeriFlow [131] is a layer between the controller and the data plane that is proposed for runtime verification of the forwarding rules to check network invariants violations. It acts as a proxy application by intercepting and monitoring rules insertion and deletion messages between the two entities and checking their effect on the network. In case of invariant violation, VeriFlow executes the associated action that is pre-configured by the network operator. The major drawback of VeriFlow is the added latency to the connection as it runs in real-time. This work is summarized in Table V. NetPlumber [132] is a real-time policy checking tool based on Header Space Analysis (HSA) [156]. It creates a dependency graph of all forwarding rules in the network, then uses it to verify the overall network policy. NetPlumber allows preventing errors and reporting violations as soon as they occur. It can be used both in SDN and conventional networks.

E. Application/Control Layers

1) Policy Correctness: Wang *et al.* [133] focus on SDN-firewall applications and proposes an HSA-based approach for detecting and resolving conflicts between firewall rules and the flow table rulesets in order to avoid bypass threats. In [134], a model checking-based approach is proposed to verify that the dynamically inserted flow entries do not violate the overall security properties. Porras *et al.* [58] focus on the problem of detecting and re-conciliating potentially conflicting flow rules caused by dynamic OpenFlow applications insertions. To address this issues, an extension to NOX controller, called FortNox, has been proposed with an integrated conflict analyzer component that detects flow rules conflicts over the flow rule insertion requests made by several security OpenFlow applications. To do so, FORTNOX translates all the current flow rules into a representation called Alias Reduced Roles (ARR) and stores them in an aggregate flow table. To detect a conflict between a candidate flow ruleset (cRules) and the existing ruleset (fRules), cRules are converted to the ARR form and then compared to the fRules. Conflicts are resolved based on the priority attached to each ruleset.

These works are compared with other approaches in Table V.

2) Northbound Interface Security: FortNox [58] implements a role-based authorization model for SDN applications based on three roles among applications that produce flow rule insertion requests: Human administrator, security applications, and non-security related applications. To these roles different priorities are assigned to the flow rule insertion requests: Human administrator with the highest priority, security applications with medium priority, and non-security related applications with the lowest priority. Roles are implemented through a digital signature scheme, in which FortNOX is preconfigured with the public keys of various rule insertion sources. The role-based source authentication component validate the digital signatures and assigns the appropriate priority to each candidate flow rule.

F. Application/Control/Infrastructure Layers

1) Policy Updates Correctness: SDN allows frequent modifications to the network configuration. However, these changes may introduce inconsistencies leading to network failures. To avoid such scenarios, mechanisms to prevent transient anomalies during changes are of paramount importance. A number of research works [135]–[137] propose safe update protocols and abstractions for OpenFlow networks. Reitblatt *et al.* [135] propose abstractions for network updates with strong semantic guarantees. These are implementable as abstract update operations that ensure per-packet and per-flow consistencies. Each operation identifies a set of packets that, when updating from an old policy to a new one across multiple switches, every packet (per-packet consistency) uses either the old or the new policy, not some combination of the two. Per-flow consistency is a generalization of per-packet consistency where it guarantees all packets in the same flow to be processed by the same policy. A formal model and proofs of the per-packet abstraction updates are investigated in [136]. Therein, Kinetic, a run-time system implementing these update abstractions on top of the NOX OpenFlow controller, is presented and its performance is evaluated. Kinetic is realized under the Frenetic [66] project and provides consistent writes, which are the dual abstractions of consistent reads presented in Frenetic. McGeer [137] propose OpenFlow Safe Update Protocol to ensure per-packet rule-set consistency and demonstrate formally its correctness.

2) Network Correctness: As any software, controllers and its applications may contain not only implementation errors, but

also critical design and logical flaws. The impact of these flaws could be extremely damaging if not detected and corrected before system deployment. Nice [138] is a proactive approach for testing the behavior of SDN application and controllers where a simplified OpenFlow Switch model is used. NICE mainly combines symbolic execution, explicit state model-checking, and search strategies. It exhaustively explores systems state using a model-checker in order to find out invalid states. It is designed for offline-verification. OFTEN [139] is an OpenFlow integration testing framework for SDN networks, based on black-box testing. Instead of the NICE simplified OpenFlow switch model, OFTEN builds upon NICE [138] by enabling testing an SDN system consisting of one or more controllers and potentially heterogeneous collection of real switches to check that it operates without violating correctness properties. These works are summarized in Table V.

VI. SDN Open Issues

A part from the discussed issues to which some researcher have proposed potential solutions, other open research problems are still not well investigated and need to be addressed by future research efforts to provide more chances to the adoption of SDN. In this section, we review some of the most important open research issues.

A. SDN Security Applications

Managing and orchestrating security in an SDN-based network requires the development and creation of security applications that interact with the controller northbound API to accomplish the desired security functions. For instance, network security applications, such as intrusion and anomaly detection and prevention require packets related information at different levels of details and at different paces. Particularly, access to payload information is crucial for many network security applications. Additionally, this information need to be obtained at a considerably reduced latencies in order to respond appropriately to abnormal traffic or degenerate conditions.

In its current version, OpenFlow [14] handles mostly layer 2/3 network traffic information and the entire packet may be sent to the controller but only in some special cases (because of no available buffers in the switch or the first packet of a given unknown flow). Thus, applications that need to have access and to manipulate data packet payload cannot benefit from the current OpenFlow implementation as both deep packet inspection and aggressive polling of the data plane can rapidly cause degradation of the latter's performance. There are some research efforts that have been proposing initial solutions for such a problem [93], [97], [98], however, major efforts need to be spent in this area in order to propose solutions with good trade-offs between performance, usability and security.

B. Security of SDN

After reviewing the literature, we can safely affirm that security and dependability of the SDN itself is still an open issue. While SDN brings significant promises to networking, it introduces many legitimate questions about the potential security risks that SDN itself might present to a network. Various works (e.g., [157]–[159]) have investigated vulnerabilities and threat vectors related to the deployment of SDN with OpenFlow. By decoupling the control plane from the data plane, the attack surface for SDNs is augmented, when compared to traditional networks. According to Kreutz *et al.* [157], new attack surface areas are introduced by SDN deployment. Three identified vectors out of seven are specific to SDN, which are controllers software, control-to-data layers communications, and control-to-application layers communications. The remaining identified four threats, already present in traditional networks, may have a potentially augmented impact.

Among the well-known vulnerabilities of the SDN platform, controllers are susceptible to DoS attacks, which can have a devastating impact on the whole network. By setting up a large number of new and unknown flows, an attacker can overwhelm the controller by a large number of OpenFlow requests from the switch to decide on how to handle these flows. A saturated controller would no more be able to make decisions about the rest of the traffic. To address such issue, Kreutz *et al.* [157] propose the replication of the controller with the applications, the use of diverse controller products, and a mechanism to dynamically associate switches with more than one controller. However, in a scenario where the switch has to store packets in its input queue awaiting for the flow table entry to be returned, the DoS can be also observed on the level of the switch node. Several mechanisms and good practices from several communities are proposed in [157] to address various threats. While these recommendations are valuable for improving the security of SDN, no concrete solutions were provided. These recommendations need to be followed by specific solutions that should be carefully studied and experimented so that they do not add other problems such as performance and scalability problems, do not decrease the expectations from SDN (e.g., flexibility), and do not introduce new security threats. Among the few works proposing concrete solutions to secure control platform in SDN, Shin *et al.* [93] designed and implemented AVANT-GUARD to defeat TCP-SYN flooding attacks and network scanning. The proposed framework is shown to successfully prevent control plane saturation attacks (DOS) and flow-rule-flooding problem in the data plane. However, the proposed approach is not designed to prevent application layer DoS attacks or attacks based on other protocols such as UDP or ICMP. Also, more works need to be done to address more sophisticated attacks.

Another important identified threat to SDN is the communication between the applications and the controller API. SDN networks are programmed using policies that might be frequently and easily modified using business applications. These applications systematically acquire the privilege of manipulating the entire network behavior through the controller. As the controller does not apply any verification on the semantics of the implemented policies, buggy and malicious applications may be at the origin of various sever threats such as circumventing flow rules imposed by security applications or may cause harm to the whole network due to the abuse of privilege. Porras *et al.* [58] are pioneers in practically addressing this

issue by proposing a security enforcement kernel (FortNOX) that implements a role-based authentication technique and sets priorities between applications in order to restrict their privileges. In this context, Flowvisor [13], [30], [31] attempts to enhance the SDN security by achieving the network inter-slices isolation, which may decrease the impact of rogue applications on only a single slice of the network.

Other identified security threats are related to the OpenFlow switch specification. For instance, Benton *et al.* [158] analyze OpenFlow vulnerabilities of version 1.0.0. However, the OpenFlow specification has been extended and new features have been improved along the released versions. Koli [159] studied OpenFlow vulnerabilities using STRIDE methodology and showed that even though various improvements have been performed on the OpenFlow switch specification from version 1.0.0 to 1.3.1, they address only a subset of the potential security flaws. In the IETF Internet draft [160], some security properties of OpenFlow specification version 1.3.0 [44] are discussed. It has been noticed that security of OpenFlow is underspecified, which may lead to differences between multiple implementations and consequently to operational complexity, interoperability issues or unexpected security vulnerabilities. While analyzing the latest OpenFlow version 1.4.0 [14], we found that the vulnerabilities identified in [160] were not eliminated.

To summarize, the security research community needs to attach a considerable attention to security issues in SDN in order to reduce risks while preserving the undeniable benefits of deploying SDN. Still major efforts need to be spent in this area in order to propose solutions with good trade-offs between performance, usability and security.

C. Compatibility and Interoperability

OpenFlow switches run embedded software that are mainly needed to process control messages sent by the controller and configure the flow tables accordingly. This piece of software needs to be compliant with the OpenFlow specification. However, specifications may be ambiguous and may have several interpretations, which may give implementation freedom to vendors. This could lead to implementations that exhibit compatibility and interoperability concerns. In real-life SDN deployment scenarios, it is likely that the infrastructure is constituted of OpenFlow switches from multiple vendors. Thus, this type of problems can easily occur at the forwarding infrastructure level. SOFT (Systematic OpenFlow Testing) [161] is an exhaustive approach and tool for automated switch interoperability testing using symbolic execution and the constraint solver STP (having as input formulas over the theory of bit-vectors and arrays that captures most expressions from languages like C,C++,Java, Verilog etc.). The approach allows to leverage multiple OpenFlow implementations at the development stage.

As SDN enables the development of independent network components, it becomes an urgent necessity to ensure that SDN networks and components, when integrated together, perform correctly at each layer of the SDN stack (control and data plane). Attempting to investigate these concerns, Kuzniar *et al.* [139] proposed OFTEN for testing integrated OpenFlow SDN systems consisting of one or more controllers and potentially heterogeneous set of real switches. OFTEN checks that the system does not violate preexisting list of correctness properties. For instance, some packets were lost by the tested load balancing application during reconfiguration phase due to incompatibility of OpenFlow switch specification earlier to version 0.8.9 with the later ones, which was not taken into account in the tested application.

Finally, as multiple controllers may be used to control the same or various domains, it is important to ensure compatibility among controllers to enable cooperation. This communication is needed for enabling various fundamental services such as inter-domain routing to enable communication between hosts in different domains. This compatibility can be improved by standardizing inter-controller communication through the east-westbound interface. To summarize, research in this direction has not received enough attention. A lot of work is needed to provide appropriate tools and techniques to resolve incompatibility and interoperability problems.

D. SDN Applications Creation and Orchestration

SDN brings two potential benefits for improving computer networks: facilitating innovation in network technologies on the one hand, and making the creation, deployment, and composition of a variety of network services an easy task on the other hand. While the first opportunity has been well-grasped by many efforts, the second one has received less attention. Indeed, few works [70], [162], [163] have proposed frameworks for orchestrating policies expressed in different contexts (QoS, traffic engineering, access control, etc.) in order to harmoniously manage networks and avoid possible conflicts.

Although these are some promising research contributions to network services creation and orchestration, there is a clear need for more research and implementation efforts in this direction. Indeed, it is vital to SDN to provide a framework for the creation, the deployment, and the coordination of not only security-related applications but also all types of applications that achieve and improve on today's networks services. Additionally, these frameworks should enable the development of applications independently from the used controller.

VII. Conclusion

SDN has recently gained an unprecedented attention from academia and industry. SDN was born in academia [9], [10], [12]. Several important organizations such as Google [101] and VMware are running SDN networks and several experimental testbeds are running and testing OpenFlow networks worldwide, including NDDI [164], OFELIA [165], FIBRE [166], JGN-X [167] and GENI [168]. Thus, a survey on SDN that studies various aspects of this novel networking paradigm was needed.

In this paper, we elaborated a thorough survey and tutorial on SDN to investigate the potential of SDN in revolutionizing networks as we know them today. First, we went back to the roots from where SDN and OpenFlow have emerged. Then,

we presented SDN concepts and described its architecture. Therein, we detailed the main SDN's components, namely the forwarding devices and the logically centralized controller, along with their functionalities and interactions. We also compared various available products conceived to support SDN deployment, such as controllers software, OpenFlow-enabled switches, and frameworks for SDN programming. Afterward, we studied existing SDN-related taxonomies and proposed a layered taxonomy that allows classifying the reviewed research works. The proposed taxonomy presents a hierarchical view and classifies the identified issues and solutions per layer (or layers) they belong to.

In the second part of this paper, we surveyed the research initiatives aiming at solving the already identified issues and described some relevant application domains where SDN is expected to make the difference, particularly for emerging technologies such as cloud computing. Finally, we have investigated some of the open issues that have been poorly addressed by the literature and thus need to be addressed by future research efforts.

A recent IDC study [169] projected that the SDN market will increase from $360 million in 2013 to $3.7 billion in 2016. However, in order to reach wide acceptance, we believe that the maturity of SDN is a critical factor. This maturity depends on the advancement in the design and implementation of various SDN components, namely the controllers, the switches, and the application services as well as the interfaces across them. Furthermore, several other issues including security, interoperability, reliability, and scalability need further investigation. Once the maturity of SDN reaches an acceptable level, training and education of networks stakeholders is an important step for a smooth transition to the SDN paradigm.

REFERENCES

[1] A. T. Campbell *et al.*, "A survey of programmable networks," *SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 2, pp. 7–23, Apr. 1999.

[2] Defense Advanced Research Projects Agency. (1997). Active network program. [Online]. Available: http://www.sds.lcs.mit.edu/darpa-activenet/

[3] 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.

[4] N. Feamster, J. Rexford, and E. Zegura, "The Road to SDN: An Intellectual History of Programmable Networks," ACM Queue, New York, NY, USA, Tech. Rep., 2013.

[5] M. Caesar *et al.*, "Design and implementation of a routing control platform," in *Proc. ACM/USENIX NSDI*, 2005, pp. 15–28.

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

[7] H. Yan *et al.*, "Tesseract: A 4D network control plane," in *Proc. Netw. Syst. Des. Implementation*, 2007, pp. 369–382.

[8] M. Casado *et al.*, "SANE: A protection architecture for enterprise networks," in *Proc. USENIX Security Symp.*, 2006, pp. 1–15.

[9] M. Casado *et al.*, "Ethane: Taking control of the enterprise," in *Proc. SIGCOMM Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2007, pp. 1–12.

[10] M. Casado *et al.*, "Rethinking enterprise network control," *IEEE/ACM Trans. Netw.*, vol. 17, no. 4, pp. 1270–1283, Aug. 2009.

[11] "Software-Defined Networking: The New Norm for Networks," Open Netw. Found., Palo Alto, CA, USA, White Paper, Apr. 2013.

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

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

[14] "OpenFlow switch specification," version 1.4.0 (Wire Protocol 0x05), Oct. 2013.

[15] Z. Bozakov and V. Sander, "OpenFlow: A perspective for building versatile networks," in *Network-Embedded Management and Applications*, A. Clemm and R. Wolter, Eds. New York, NY, USA: Springer-Verlag, 2013, pp. 217–245.

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

[17] 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.

[18] P. Lin, J. Bi, H. Hu, T. Feng, and X. Jiang, "A quick survey on selected approaches for preparing programmable networks," in *Proc. 7th AINTEC*, 2011, pp. 160–163.

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

[20] A. Doria *et al.* (2010, Mar.). Forwarding and Control Element Separation (ForCES) Protocol Specification, (RFC) 5810, 2070-1721. [Online]. Available: http://tools.ietf.org/html/rfc5810/

[21] J. Vasseur and J. L. Roux. (2009, Mar.). Path Computation Element (PCE) Communication Protocol (PCEP), (RFC) 5440. [Online]. Available: http://tools.ietf.org/html/rfc5440

[22] "SDN security considerations in the data center," Open Networking Foundation, Palo Alto, CA, USA, Solution Brief, Oct. 2013.

[23] Big Switch Networks. (2012, Oct.). The Open SDN Architecture. [Online]. Available: http://www.bigswitch.com/sites/default/files/sdn_overview.pdf

[24] CISCO, Cisco's One Platform Kit (onePK). [Online]. Available: http://www.cisco.com/en/US/prod/iosswrel/onepk.html

[25] Juniper Networks, Contrail: A SDN Solution Purpose-Built for the Cloud. [Online]. Available: http://www.juniper.net/us/en/products-services/sdn/contrail/

[26] Z. Wang, T. Tsou, J. Huang, X. Shi, and X. Yin. (2012, Mar.). Analysis of Comparisons Between OpenFlow and ForCES. [Online]. Available: http://tools.ietf.org/html/draft-wang-forces-compare-openflow-forces-01

[27] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Univ. California, Irvine, CA, USA, 2000.

[28] "Realizing the Power of SDN With HP Virtual Application Networks," Hewlett-Packard Development Co., Cupertino, CA, USA, Tech. Rep., 4AA4-3871ENW, Oct. 2012.

[29] M. Kind, F. Westphal, A. Gladisch, and S. Topp, "SplitArchitecture: Applying the software defined networking concept to carrier networks," in *Proc. WTC*, 2012, pp. 1–6.

[30] B. Sonkoly *et al.*, "OpenFlow virtualization framework with advanced capabilities," in *Proc. EWSDN*, 2012, pp. 18–23.

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

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

[33] M. Casado, T. Koponen, R. Ramanathan, and S. Shenker, "Virtualizing the network forwarding plane," in *Proc. Workshop PRESTO*, 2010, pp. 8:1–8:6.

[34] Open vSwitch. (2012, May). OpenvSwitch: An Open Virtual Switch. [Online]. Available: http://openvswitch.org/

[35] J. Pettit, J. Gross, B. Pfaff, M. Casado, and S. Crosby, "Virtual switching in an era of advanced edges," in *Proc. 2nd Workshop DC CAVES*, Sep. 2010, pp. 1–7.

[36] B. Pfaff *et al.*, "Extending networking into the virtualization layer," presented at the Workshop on Hot Topics in Networks (HotNets-VIII), New York City, NY, USA, 2009.

[37] 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. ANCS*, 2008, pp. 1–9.

[38] R. Neugebauer, "Selective and Transparent Acceleration of OpenFlow Switches," Netronome, Santa Clara, CA, USA, Tech. Rep., 2013.

[39] Stanford University, OpenFlow Reference Software Repository. [Online]. Available: http://yuba.stanford.edu/git/gitweb.cgi?p=openflow.git;a=summary

[40] Pica8, Pica8's OS for Open Switches. [Online]. Available: http://www.pica8.com/open-switching/open-switching-overview.php

[41] Big Switch, Indigo. [Online]. Available: http://www.projectfloodlight.org/indigo/

[42] Y. Yiakoumis, Pantou: OpenFlow 1.0 for OpenWRT. [Online]. Available: http://www.openflow.org/wk/index.php/OpenFlow_1.0_for_OpenWRT

[43] "OpenFlow switch specification," version 1.0.0 (Wire Protocol 0x01), Dec. 2009.

[44] "OpenFlow switch specification," version 1.3.0 (Wire Protocol 0x04), Jun. 2012.

[45] D. Bansal, S. Bailey, T. Dietz, and A. A. Shaikh, "OpenFlow Management and Configuration Protocol (OF-Config 1.1.1)," Open Networking Foundation, Palo Alto, CA, USA, Mar. 2013.

[46] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proc. 2nd USENIX Conf. Hot-ICE*, 2012, p. 10.

[47] Nicira, About POX. [Online]. Available: http://www.noxrepo.org/pox/about-pox/

[48] Maestro. [Online]. Available: http://code.google.com/p/maestro-platform/

[49] D. Erickson, "The beacon OpenFlow controller," in *Proc. ACM SIGCOMM Workshop HotSDN*, Aug. 2013, pp. 13–18.

[50] SNAC. [Online]. Available: http://www.openflow.org/wp/snac/

[51] S. Ishii *et al.*, "Extending the RISE controller for the interconnection of RISE and OS3E/NDDI," in *Proc. 18th IEEE ICON*, 2012, pp. 243–248.

[52] Floodlight. [Online]. Available: http://www.projectfloodlight.org/floodlight/

[53] A. Voellmy and J. Wang, "Scalable software defined network controllers," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 289–290, Aug. 2012.

[54] KulCloud, MUL OpenFlow Controller. [Online]. Available: http://sourceforge.net/p/mul/wiki/Home/

[55] NTT, NTT Laboratories OSRG Group. [Online]. Available: http://osrg.github.com/ryu/

[56] OpenDaylight, OpenDaylight: Technical Overview. [Online]. Available: http://www.opendaylight.org/project/technical-overview

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

[58] P. Porras *et al.*, "A security enforcement kernel for OpenFlow networks," in *Proc. ACM SIGCOMM Workshop HotSDN*, Aug. 2012, pp. 121–126.

[59] Z. Cai, A. L. Cox, and T. S. E. Ng, "Maestro: Balancing fairness, latency and throughput in the OpenFlow control plane," Dept. Comput. Sci., Rice Univ., Houston, TX, USA, Tech. Rep. TR11-07, Jul. 2011.

[60] Z. Cai, F. Dinu, J. Zheng, A. L. Cox, and T. S. E. Ng, "The preliminary design and implementation of the Maestro network control platform," Dept. Comput. Sci., Rice Univ., Houston, TX, USA, Tech. Rep., Oct. 2008.

[61] A. Voellmy and P. Hudak, "Nettle: Taking the sting out of programming network routers," in *Proc. 13th Intl. Conf. PADL*, 2011, pp. 235–249.

[62] *OSGi Core Release 5*, OSGi Alliance, San Ramon, CA, USA, Mar. 2012.

[63] K. Kirkpatrick, "Software-defined networking," *Commun. ACM*, vol. 56, no. 9, pp. 16–19, Sep. 2013.

[64] S. Azodolmolky, *Software Defined Networking With OpenFlow*. Birmingham, U.K.: Packt Publishing, Oct. 2013.

[65] P. Hudak, "Functional reactive programming," in *Programming Languages and Systems*, S. Swierstra, Ed. Berlin, U.K.: Springer-Verlag, 1999, vol. 1576, ser. Lecture Notes in Computer Science, pp. 1–1.

[66] N. Foster *et al.*, "Frenetic: A network programming language," *SIGPLAN Not.*, vol. 46, no. 9, pp. 279–291, Sep. 2011.

[67] N. Foster *et al.*, "Languages for software-defined networks," *IEEE Commun. Mag.*, vol. 51, no. 2, pp. 128–134, Feb. 2013.

[68] C. Monsanto, N. Foster, R. Harrison, and D. Walker, "A compiler and run-time system for network programming languages," in *Proc. 39th Annu. ACM SIGPLAN-SIGACT Symp. POPL*, 2012, pp. 217–230.

[69] T. L. Hinrichs, N. S. Gude, M. Casado, J. C. Mitchell, and S. Shenker, "Practical declarative network management," in *Proc. 1st ACM WREN*, 2009, pp. 1–10.

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

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

[72] J. Hughes, "Generalising monads to arrows," *Sci. Comput. Programm.*, vol. 37, no. 1–3, pp. 67–111, May 1998.

[73] A. K. S. Joe Skorupa and M. Fabbi. (2013, Mar.). Ending the Confusion About Software-Defined Networking: A Taxonomy. [Online]. Available: http://www.gartner.com/id=2367616

[74] A. M. Alberti, "A conceptual-driven survey on future internet requirements, technologies, and challenges," *J. Brazilian Comput. Soc.*, vol. 19, no. 3, pp. 291–311, Sep. 2013.

[75] S. Schmid and J. Suomela, "Exploiting locality in distributed SDN control," in *Proc. ACM SIGCOMM Workshop HotSDN*, Aug. 2013, pp. 12–126.

[76] K. Kannan and S. Banerjee, "Compact TCAM: Flow entry compaction in TCAM for power aware SDN," in *Distributed Computing and Networking*, D. Frey, M. Raynal, S. Sarkar, R. Shyamasundar, and P. Sinha, Eds. Berlin, Germany: Springer-Verlag, 2013, vol. 7730, ser. Lecture Notes in Computer Science, pp. 439–444.

[77] R. Narayanan *et al.*, "Macroflows and Microflows: Enabling rapid network innovation through a split SDN data plane," in *Proc. EWSDN*, 2012, pp. 79–84.

[78] G. Lu, R. Miao, Y. Xiong, and C. Guo, "Using CPU as a traffic co-processing unit in commodity switches," in *Proc. 1st Workshop on HotSDN*, 2012, pp. 31–36.

[79] Y. Luo, P. Cascon, E. Murray, and J. Ortega, "Accelerating OpenFlow switching with network processors," in *Proc. 5th ACM/IEEE Symp. ANCS*, 2009, pp. 70–71.

[80] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the "One big switch" abstraction in software-defined networks," in *Proc. 9th CoNEXT*, 2013, pp. 13–24.

[81] V. Tanyingyong, M. Hidell, and P. Sjodin, "Using hardware classification to improve PC-based OpenFlow switching," in *Proc. IEEE 12th Intl. Conf. HPSR*, 2011, pp. 215–221.

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

[83] R. McGeer, "Verification of switching network properties using satisfiability," in *Proc. IEEE ICC*, 2012, pp. 6638–6644.

[84] R. W. Skowyra, A. Lapets, A. Bestavros, and A. Kfoury, "Verifiably-safe software-defined networks for CPS," in *Proc. 2nd ACM Int. Conf. High Confidence Netw. Syst.*, 2013, pp. 101–110.

[85] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow," in *Proc. INM/WREN*, 2010, pp. 3–3.

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

[87] V. Yazıcı, M. O. Sunay, and A. Ö. Ercan, "Controlling a software-defined network via distributed controllers," in *Proc. NEM Summit, Implementing Future Media Internet Towards New Horizons*, 2012, pp. 16–21.

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

[89] C. Macapuna, C. Rothenberg, and M. Magalha es, "In-packet bloom filter based data center networking with distributed OpenFlow controllers," in *Proc. IEEE GC Wkshps*, 2010, pp. 584–588.

[90] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proc. 1st Workshop HotSDN*, 2012, pp. 7–12.

[91] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng, "On the placement of controllers in software-defined networks," *J. China Univ. Posts Telecommun.*, vol. 19, no. S2, pp. 92–171, Oct. 2012.

[92] 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 HotSDN*, 2012, pp. 1–6.

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

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

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

[96] 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 HotSDN*, 2012, pp. 127–132.

[97] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, "Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments," *Comput. Netw.*, vol. 67, pp. 122–136, Apr. 2014.

[98] S. Shirali-Shahreza and Y. Ganjali, "Efficient implementation of security applications in OpenFlow controller with FleXam," in *Proc. IEEE 21st Annu. Symp HOTI*, 2013, pp. 49–54.

[99] H. E. Egilmez, S. T. Dane, B. Gorkemli, and A. M. Tekalp, "OpenQoS: OpenFlow controller design and test network for multimedia delivery

with quality of service," in *Proc. NEM Summit, Implementing Future Media Internet Towards New Horizons*, 2012, pp. 22–27.
[100] B. Sonkoly *et al.*, "On QoS support to Ofelia and OpenFlow," in *Proc. EWSDN*, 2012, pp. 109–113.
[101] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined WAN," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, Aug. 2013.
[102] G. Wang, T. E. Ng, and A. Shaikh, "Programming your Network at runtime for big data applications," in *Proc. 1st Workshop HotSDN*, 2012, pp. 103–108.
[103] S. Das *et al.*, "Application-aware aggregation and traffic engineering in a converged packet-circuit network," in *Proc. OFC/NFOEC*, 2011, pp. 1–3.
[104] N. Kang, J. Reich, J. Rexford, and D. Walker, "Policy transformation in software defined networks," in *Proc. ACM SIGCOMM Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2012, pp. 309–310.
[105] R. Wang, D. Butnariu, and J. Rexford, "OpenFlow-based server load balancing gone wild," in *Proc. 11th USENIX Conf. Hot-ICE*, 2011, pp. 12–12.
[106] J. Matias, E. Jacob, D. Sanchez, and Y. Demchenko, "An OpenFlow-based network virtualization framework for the cloud," in *Proc. IEEE 3rd Int. Conf. CloudCom*, 2011, pp. 672–678.
[107] L. Sun, K. Suzuki, C. Yasunobu, Y. Hatano, and H. Shimonishi, "A network management solution based on OpenFlow towards new challenges of multitenant data center," in *Proc. 9th APSITT*, 2012, pp. 1–6.
[108] C. Rotsos, R. Mortier, A. Madhavapeddy, B. Singh, and A. Moore, "Cost, performance & flexibility in OpenFlow: Pick three," in *Proc. IEEE ICC*, 2012, pp. 6601–6605.
[109] The OpenStack Foundation, Openstack Cloud Software. [Online]. Available: http://www.openstack.org/
[110] 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.
[111] G. Stabler, A. Rosen, S. Goasguen, and K.-C. Wang, "Elastic IP and security groups implementation using OpenFlow," in *Proc. 6th Int. Workshop VTDC*, 2012, pp. 53–60.
[112] T. Koorevaar, "Dynamic enforcement of security policies in multi-tenant cloud networks," M.S. thesis, Départ. Génie Inf. et Génie Logiciel, Ecole Polytechnique de Montreal, Montréal, QC, Canada, Nov., 2012.
[113] V. Mann, A. Vishnoi, K. Kannan, and S. Kalyanaraman, "CrossRoads: Seamless VM mobility across data centers through software defined networking," in *Proc. IEEE NOMS*, 2012, pp. 88–96.
[114] F. Hao, T. V. Lakshman, S. Mukherjee, and H. Song, "Enhancing dynamic cloud-based services using network virtualization," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 67–74, Jan. 2010.
[115] B. Boughzala, R. Ben Ali, M. Lemay, Y. Lemieux, and O. Cherkaoui, "OpenFlow supporting inter-domain virtual machine migration," in *Proc. 8th Int. Conf. WOCN*, 2011, pp. 1–7.
[116] T. Xing, D. Huang, L. Xu, C.-J. Chung, and P. Khatkar, "SnortFlow: A OpenFlow-based intrusion prevention system in cloud environment," in *Proc. 2nd GREE*, 2013, pp. 89–92.
[117] C.-J. Chung, P. Khatkar, T. Xing, J. Lee, and D. Huang, "NICE: Network intrusion detection and countermeasure selection in virtual network systems," *IEEE Trans. Dependable Secure Comput.*, vol. 10, no. 4, pp. 198–211, Jul.-Aug. 2013.
[118] M. Mendonca, K. Obraczka, and T. Turletti, "The case for software-defined networking in heterogeneous networked environments," in *Proc. ACM Conf. CoNEXT Student*, 2012, pp. 59–60.
[119] K.-K. Yap *et al.*, "OpenRoads: Empowering research in mobile networks," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 125–126, Jan. 2010.
[120] P. Dely, A. Kassler, and N. Bayer, "OpenFlow for wireless mesh networks," in *Proc. 20th ICCCN*, 2011, pp. 1–6.
[121] L. Li, Z. Mao, and J. Rexford, "Toward software-defined cellular networks," in *Proc. EWSDN*, 2012, pp. 7–12.
[122] M. Bansal, J. Mehlman, S. Katti, and P. Levis, "OpenRadio: A programmable wireless dataplane," in *Proc. 1st Workshop HotSDN*, 2012, pp. 109–114.
[123] N. Blefari-Melazzi, A. Detti, G. Morabito, S. Salsano, and L. Veltri, "Information centric networking over SDN and OpenFlow: Architectural aspects and experiments on the OFELIA Testbed," *J. Comput. Netw.*, vol. 57, no. 16, pp. 3207–3221, Nov. 2013.
[124] D. Syrivelis *et al.*, "Pursuing a software defined information-centric network," in *Proc. EWSDN*, 2012, pp. 103–108.
[125] A. Chanda, C. Westphal, and D. Raychaudhuri, "Content based traffic engineering in software defined information centric networks," *CoRR*, vol. abs/1301.7517, 2013.
[126] Open Networking Foundation. (2012, Oct.). Network Functions Virtualisation. An Introduction, Benefits, Enablers, Challenges and Call for Action, NFV Industry Specification Group, nFV white paper. [Online]. Available: http://portal.etsi.org/NFV/NFV_White_Paper.pdf/
[127] 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.
[128] A. R. Curtis *et al.*, "DevoFlow: Scaling flow management for high-performance networks," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 254–265, Aug. 2011.
[129] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, Aug. 2010.
[130] A. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *Proc. IEEE INFOCOM*, 2011, pp. 1629–1637.
[131] A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey, "VeriFlow: Verifying network-wide invariants in real-time," in *Proc. 1st HotSDN*, 2012, pp. 49–54.
[132] P. Kazemian *et al.*, "Real time network policy checking using header space analysis," in *Proc. 10th USENIX Conf. NSDI*, 2013, pp. 99–112.
[133] J. Wang *et al.*, "Towards a security-enhanced firewall application for OpenFlow networks," in *Cyberspace Safety and Security*. Berlin, Germany: Springer-Verlag, 2013, pp. 92–103.
[134] S. Son, S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Model checking invariant security properties in OpenFlow," in *Proc. IEEE ICC*, Jun. 2013, pp. 1974–1979.
[135] M. Reitblatt, N. Foster, J. Rexford, and D. Walker, "Consistent updates for software-defined networks: Change you can believe in!" in *Proc. 10th ACM Workshop HotNets-X*, 2011, pp. 7:1–7:6.
[136] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *Proc. ACM SIGCOMM Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2012, pp. 323–334.
[137] R. McGeer, "A safe, efficient update protocol for OpenFlow networks," in *Proc. 1st Workshop HotSDN*, 2012, pp. 61–66.
[138] M. Canini, D. Venzano, P. Perešíni, D. Kostić, and J. Rexford, "A NICE way to test OpenFlow applications," in *Proc. 9th USENIX Conf. NSDI*, 2012, pp. 10–10.
[139] M. Kuzniar, M. Canini, and D. Kostic, "OFTEN testing OpenFlow networks," in *Proc. EWSDN*, 2012, pp. 54–60.
[140] S. Zhang, S. Malik, and R. McGeer, "Verification of computer switching networks: An overview," in *Automated Technology for Verification and Analysis*, S. Chakraborty and M. Mukund, Eds. Berlin, Germany: Springer-Verlag, 2012, ser. Lecture Notes in Computer Science, pp. 1–16.
[141] P. Fonseca, R. Bennesby, E. Mota, and A. Passito, "A replication component for resilient OpenFlow-based networking," in *Proc. IEEE NOMS*, 2012, pp. 933–939.
[142] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker, "Applying NOX to the datacenter," in *Proc. 8th ACM SIGCOMM Workshop HOTNETS*, 2009.
[143] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: Measurements & analysis," in *Proc. 9th ACM SIGCOMM Conf. IMC*, 2009, pp. 202–208.
[144] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller," in *Proc. 2nd ACM SIGCOMM Workshop HotSDN*, 2013, pp. 7–12.
[145] 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.
[146] M. L. Peter Phaal. (2004, Jul.). sFlow Specification Version 5. [Online]. Available: http://www.sflow.org/sflow_version_5.txt
[147] A. Madhavapeddy *et al.*, "Turning down the LAMP: Software specialisation for the cloud," in *Proc. 2nd USENIX Conf. HotCloud*, 2010, p. 11.
[148] L. Veltri, G. Morabito, S. Salsano, N. Blefari-Melazzi, and A. Detti, "Supporting information-centric functionality in software defined networks," in *Proc. IEEE ICC*, 2012, pp. 6645–6650.
[149] CONVERGENCE, The CONVERGENCE EU FP7 project. [Online]. Available: http://www.ict-convergence.eu/
[150] SAIL, The sail eu fp7 project. [Online]. Available: http://www.sail-project.eu/
[151] PURSUIT, The pursuit eu fp7 project. [Online]. Available: http://www.fp7-pursuit.eu/
[152] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Comput. Netw.*, vol. 54, no. 5, pp. 862–876, Apr. 2010.

[153] M. F. Bari *et al.*, "Data center network virtualization: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 2, pp. 909–928, 2nd Quart. 2013.
[154] A. Wang, M. Iyer, R. Dutta, G. Rouskas, and I. Baldine, "Network virtualization: Technologies, perspectives, and frontiers," *J. Lightw. Technol.*, vol. 31, no. 4, pp. 523–537, Feb. 2013.
[155] M. Jarschel *et al.*, "Modeling and performance evaluation of an OpenFlow architecture," in *Proc. 23rd ITC*, 2011, pp. 1–7.
[156] P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: Static checking for networks," in *Proc. 9th USENIX Conf. NSDI*, 2012, pp. 9–9.
[157] D. Kreutz, F. M. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proc. ACM SIGCOMM Workshop HotSDN*, 2013, pp. 55–60.
[158] K. Benton, L. J. Camp, and C. Small, "OpenFlow vulnerability assessment," in *Proc. 2nd ACM SIGCOMM Workshop HotSDN*, 2013, pp. 151–152.
[159] R. Kloti, "OpenFlow: A security analysis," M.S. thesis, Inf. Technol. Elect. Eng. Depart., ETH Zurich, Zurich, Switzerland, Apr. 2013, MA-2012-20.
[160] M. Wasserman and S. Hartman. (2012, Apr.). Security Analysis of the Open Networking Foundation (ONF) OpenFlow Switch Specification. [Online]. Available: http://tools.ietf.org/pdf/draft-mrw-sdnsec-openflow-analysis-02.pdf
[161] M. Kuzniar, P. Peresini, M. Canini, D. Venzano, and D. Kostic, "A SOFT way for OpenFlow switch interoperability testing," in *Proc. 8th CoNEXT*, 2012, pp. 265–276.
[162] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Commun. Mag.*, vol. 51, no. 2, pp. 114–119, Feb. 2013.
[163] S. Shin *et al.*, "FRESCO: Modular composable security services for software-defined networks," in *Proc. 20th Annu. NDSS*, Feb. 2013, pp. 1–16.
[164] S. Wallace. (2011). Network Development and Deployment Initiative (NDDI). [Online]. Available: http://inddi.wikispaces.com/Internet2-based+NDDI
[165] European Center for Information and Communication Technologies (EICT) GmbH, OpenFlow in Europe Linking Infrastructure and Applications (OFELIA). [Online]. Available: http://www.fp7-ofelia.eu/
[166] i2CAT Distributed Applications and Networks Area, Future Internet Testbeds/Experimentation Between Brazil and Europe (FIBRE). [Online]. Available: http://dana.i2cat.net/fibre-kickoff/projects/
[167] National Institute of Information and Communications Technology (NICT), New Generation Network Testbed JGN-X. [Online]. Available: https://www.jgn.nict.go.jp/english/info/technologies/openflow.html
[168] Raytheon BBN Technologies. (2012, Sep.). Global Environment for Network Innovations (GENI). [Online]. Available: http://www.geni.net/
[169] IDC Corporate USA. (2012). SDN Shakes up the Status Quo in Datacenter Networking, IDC Says. [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prUS23888012

Yosr Jarraya (M'10) is currently a Research Associate with the Concordia Institute for Information Systems Engineering, Concordia University, Montreal, QC, Canada. She is an active member of the Computer Security Laboratory. She was previously a Postdoctoral Fellow and a Research Assistant with Concordia University. She received the Ph.D. degree in electrical and computer engineering from Concordia University and the M.Sc. degree in telecommunications from Ecole Supérieure des Communications de Tunis (Sup'Com), Ariana, Tunisia. She has authored one book and more than 15 research papers in journals and conferences. Her research interests include cloud computing, software-defined networking, network security, cybersecurity, verification and validation, and software and systems engineering.

Taous Madi She is currently working toward the Ph.D. degree in information and systems engineering at the Concordia Institute for Information Systems Engineering, Concordia University, Montreal, QC, Canada. She is currently a Research Assistant with Concordia University. She received the B.S. degree in engineering and the Magister degree from the Université des Sciences et de la Technologie Houari Boumediene, Algiers, Algeria, in 2007 and 2010, respectively.

Her research interests include cloud computing, mobile computing, software-defined networking, and security.

Mourad Debbabi (M'11) is a Full Professor with the Concordia Institute for Information Systems Engineering, Concordia University, Montreal, QC, Canada. He holds the Concordia Research Chair Tier I in information systems security. He is also the President of the National Cyber Forensics Training Alliance, Montreal, QC. He is the Founder and one of the leaders of the Computer Security Laboratory, Concordia University. In the past, he was the Specification Lead of four Standard Java Intelligent Networks Java Specification Requests dedicated to the elaboration of standard specifications for presence and instant messaging. Dr. Debbabi received the Ph.D. and M.Sc. degrees in computer science from Paris-XI Orsay University, Orsay, France. He has authored two books and more than 230 research papers in journals and conferences on computer security, cyber forensics, privacy, cryptographic protocols, threat intelligence generation, malware analysis, reverse engineering, specification and verification of safety-critical systems, formal methods, Java security and acceleration, programming languages, and type theory. He supervised the successful completion of 20 Ph.D. students and more than 60 Masters students. He served as a Senior Scientist with the Panasonic Information and Network Technologies Laboratory, Princeton, NJ, USA; Associate Professor with the Computer Science Department, Laval University, Quebec, QC, Canada; Senior Scientist with General Electric Research Center, New York, NY, USA; Research Associate with the Computer Science Department, Stanford University, California, USA; and Permanent Researcher with Bull Corporate Research Center, Paris, France.