Network Innovation using OpenFlow: A Survey

Adrian Lara, Anisha Kolasani, and Byrav Ramamurthy

***Abstract*—OpenFlow is currently the most commonly deployed Software Defined Networking (SDN) technology. SDN consists of decoupling the control and data planes of a network. A software-based controller is responsible for managing the forwarding information of one or more switches; the hardware only handles the forwarding of traffic according to the rules set by the controller. OpenFlow is an SDN technology proposed to standardize the way that a controller communicates with network devices in an SDN architecture. It was proposed to enable researchers to test new ideas in a production environment. OpenFlow provides a specification to migrate the control logic from a switch into the controller. It also defines a protocol for the communication between the controller and the switches.**

As discussed in this survey paper, OpenFlow-based architectures have specific capabilities that can be exploited by researchers to experiment with new ideas and test novel applications. These capabilities include software-based traffic analysis, centralized control, dynamic updating of forwarding rules and flow abstraction. OpenFlow-based applications have been proposed to ease the configuration of a network, to simplify network management and to add security features, to virtualize networks and data centers and to deploy mobile systems. These applications run on top of networking operating systems such as Nox, Beacon, Maestro, Floodlight, Trema or Node.Flow. Larger scale OpenFlow infrastructures have been deployed to allow the research community to run experiments and test their applications in more realistic scenarios. Also, studies have measured the performance of OpenFlow networks through modelling and experimentation. We describe the challenges facing the large scale deployment of OpenFlow-based networks and we discuss future research directions of this technology.

***Index Terms*—Software Defined Networking, OpenFlow, Capabilities, Applications, Deployments, Networking Challenges.**

I. Introduction

A RECENT approach to programmable networks is the Software Defined Networking (SDN) architecture. SDN consists of decoupling the control and data planes of a network. It relies on the fact that the simplest function of a switch is to forward packets according to a set of rules. However, the rules followed by the switch to forward packets are managed by a software-based controller [1]. One motivation of SDN is to perform network tasks that could not be done without additional software for each of the switching elements. Developed applications can control the switches by running on top of a network operating system, which works as an intermediate layer between the switch and the application. Another motivation is to move part of the complexity of the network to the software-based controller instead of relying only on the hardware network devices.

OpenFlow [1] was proposed to standardize the communication between the switches and the software-based controller in an SDN architecture. The authors identify that it is difficult for the networking research community to test new ideas in current hardware. This happens because the source code of the software running on the switches cannot be modified and the network infrastructure has been "ossified" [1], as new network ideas cannot be tested in realistic traffic settings. By identifying common features in the flow tables of the Ethernet switches, the authors provide a standardized protocol to control the flow table of a switch through software. OpenFlow provides a means to control a switch without requiring the vendors to expose the code of their devices.

OpenFlow was initially deployed in academic campus networks [1]. Today, at least nine universities in the US have deployed this technology [2]. The goal of OpenFlow was to provide a platform that would allow researchers to run experiments in production networks. However, industry has also embraced SDN and OpenFlow as a strategy to increase the functionality of the network while reducing costs and hardware complexity. Table I shows a list of several OpenFlow-compliant switches available in the market. The Open Networking Foundation (ONF) [3] was founded in 2011 by Deutsche Telekom, Facebook, Google, Microsoft, Verizon, and Yahoo to promote the implementation of SDN and OpenFlow-based networks. Currently, ONF has more than 95 members including several major vendors.

OpenFlow networks have specific capabilities. For example, it is possible to control multiple switches from a single controller. It is also feasible to analyze traffic statistics using software. Forwarding information can be updated dynamically as well and different types of traffic can be abstracted and managed as flows. These capabilities have been exploited by the research community to experiment with innovative ideas and propose new applications. Ease of configuration, network management, security, availability, network and data center virtualization and wireless applications are those that have been investigated the most using OpenFlow. They have been implemented in different environments, including virtual or real hardware networks and simulations. Researchers have also focused on evaluating the performance of OpenFlow networks and on proposing methods to improve their performance.

OpenFlow offers great opportunities for network innovation but it also faces challenges. The fact that the availability of the network depends on a single controller at a given time, creates scalability and availability problems. There are security

Manuscript received May 23, 2012; revised November 9, 2012, March 5, 2012, and May 9, 2013. This material is based upon work supported by the National Science Foundation under Grant No. CNS-1040765.

The authors are with the Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE 68588-0115, USA (e-mail: {alara,akolasan,byrav}@cse.unl.edu).

Digital Object Identifier 10.1109/SURV.2013.081313.00105

[1]We assume each OpenFlow network consists of a single logically centralized controller, which could be implemented by multiple controllers, in practice.

TABLE I
EXAMPLE OPENFLOW-COMPLIANT SWITCHES.

Switch Company	Series
Arista	Arista extensible modular operating system (EOS), Arista 7124FX application switch
Ciena	Ciena Coredirector running firmware version 6.1.1
Cisco	Cisco cat6k, catalyst 3750, 6500 series
Juniper	Juniper MX-240, T-640
HP	HP procurve series- 5400 zl, 8200 zl, 6200 yl, 3500 yl, 6600
NEC	NEC IP8800
Pronto	Pronto 3240, 3290
Toroki	Toroki Lightswitch 4810
Dell	Dell Z9000 and S4810
Quanta	Quanta LB4G
Open vSwitch	Software switch. Latest version: 1.10.0

concerns regarding the fact that all the network information is contained in one single server. Compatibility issues must also be taken into consideration. Questions remain about future directions of OpenFlow research as well. We discuss the extension of this technology to network-layer devices such as IP routers, as well as the deployment of OpenFlow in wide area networks (WAN).

This survey paper is the first comprehensive document, in our opinion, to discuss the capabilities, applications, deployments and challenges of OpenFlow networks in local and wide area environments. We also describe SDN and alternative standards such as ForCES [4]. We explain how OpenFlow has received major attention among SDN technologies but we also point out the difference between SDN and OpenFlow.

We begin by giving a background of programmable networks and describing SDN in Section II. We explain the OpenFlow specification in Section III. Then we present the capabilities of OpenFlow networks in Section IV and we survey how they have been exploited in different applications in Section V. We describe deployments of OpenFlow-based networks in Section VI. Next we discuss studies that have evaluated the performance of OpenFlow in Section VII. Then we discuss the challenges faced by OpenFlow in Section VIII. We conclude by proposing future research directions in Section IX.

II. BACKGROUND OF PROGRAMMABLE NETWORKS

In this section we present several contributions to programmable networks prior to SDN and OpenFlow. One of the first approaches was SOFTNET [5], an experimental multihop packet radio network that introduced the idea of adding commands to the contents of each packet. The goal was to modify a network node during operation time, using commands written in the SOFTNET language. The motivation of the authors in creating this network was to enable experiments with different network protocols. SOFTNET was deployed as a proof of concept. There were no further large scale deployments, but the idea behind it was the motivation for Active Networks [6], [7].

The main idea of Active Networks (AN) was to allow packets to contain programs that could be executed by the network devices that they traversed. The concept of active network is due to the fact that switches perform computations on the data of the packets flowing through them and the users can inject programs into the network [6]. A survey on AN research is available in [8]. Although AN became an active field of research, it ultimately failed at being widely used. Recently, NetServ [9] was proposed as ActiveNetworks 2.0. The authors argue that NetServ contains all the necessary elements to be deployed.

SOFTNET and ActiveNetworks did not use software components to control the network devices. The programmability of the network was achieved by adding source code to the payload of the packets. More recent approaches proposed separating the control plane from the data plane by moving the first one to general purpose servers. We describe SoftRouter [10], ForCES [4] and finally we focus on OpenFlow [1]. They are all based on software defined networking architectures, where the network devices are controlled by software components.

A. Software Defined Networking

The difference between SDN and the previous approaches is that a software component running on a server or a CPU is added to the architecture of the network. In SDN, the software component is responsible for the control plane of the network. This is why we say that SDN decouples the control and data planes, as this distinction was not as clear in previous approaches.

One important feature of SDN is its ability to provide a network wide abstraction. Keller et al. [11] discuss the idea of the "platform as a service" model for networking. According to the authors, it is a common trend to decouple the infrastructure management from the service management. In this model, the underlying physical network and the topology are hidden to the user. Instead, the abstraction presented to the user is a single router. According to them, the customer is mostly interested in being able to configure policies and defining how packets are handled. We will see during the rest of this survey that a large number of publications aim at hiding the complexity of the network and providing an easier way to configure a service. Using names instead of IP addresses, or high level policies instead of access control configuration files are examples of this abstraction.

Network operating system is a key concept in SDN. It comes from the idea of abstracting the complexity of the underlying network. Lazar [12] explains how an early approach to programmable networks introduced the term of

kernel in terms of networking. The idea was precisely to draw a parallel between the network operating system and the typical operating system. In an operating system, the abstraction includes the hardware components of the CPU. In a network, the abstraction hides the topology and the network devices. Therefore, the network operating system is responsible for the abstraction provided by SDN to its users.

Another important advantage of SDN is that it enables innovation and flexibility. If the control and data plane are managed by a hardware network devices, there is little room for innovating and experiment, as the software or firmware of those devices cannot be easily modified. Instead, by having access to a software component to manage the control plane, many ideas can be explored.

B. Standardizing the communication between the control plane and the data plane

SDN provides network-wide abstraction to the user and any software-based technique can be used to manage the control plane. However, we have not discussed how is the communication between the control and data plane standardized. Next we describe how several researchers have proposed to standardize this communication.

One early proposal is the IEEE P1520 Standards Initiative for Programmable Networks Interfaces [13]. The authors identify the need of abstracting the complexity of the network to the user as well as the necessity of a programming interface to define the network. They also discuss the need of having a protocol to access the network elements.

The SoftRouter architecture [10] allows dynamic binding between the network element running the data plane and the control element (software-based). This architecture was proposed for network-layer devices that can be controlled by standard purpose servers. The software component does not need to be wired to the network device and a network element can have more than one control element across the network.

ForCES (Forwarding and Control Element Separation) [4] was created by the Internet Engineering Task Force (IETF). ForCES was proposed to standardize the way that controlling elements communicate with network elements. However, this standard did not experience widespread adoption by the vendor community. The Internet Research Task Force (IRTF) has also undertaken efforts regarding SDN. The Software Defined Networking Research Group (SDNRG) [14] aims to identify SDN approaches that can be used in the nearby future, as well as to identify future challenges. It also aims at providing a forum to SDN researchers [15].

OpenFlow [1] came next and was based on the same motivation: how to standardize the communication between the control plane and the data plane. It describes how software applications can program the flow table of different switches. OpenFlow quickly became an active research topic and we describe it in detail in the next section. Before, we briefly compare ForCES and OpenFlow.

The IETF documented the differences between ForCES and OpenFlow [16]. According to this document, both standards decouple the control and data planes and they both standardize the communication between the two planes. Regarding the architecture of the network, one difference can be found between ForCES and OpenFlow. ForCES defines networking and forwarding elements and how they can communicate with each other. The architecture of the network remains unchanged. On the other hand, OpenFlow modifies the architecture in the sense that data plane elements become simple devices that forward packets according to rules given by the control element. ForCES allows multiple control and data elements within the same network and the logic can be spread through all the elements. OpenFlow aims at having a centralized control plane.

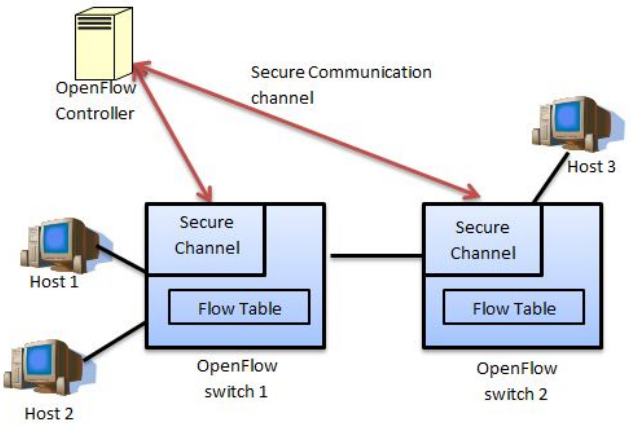

Fig. 1. OpenFlow components.

Due to the emergence of OpenFlow as the SDN architecture that has received major attention, we focus this survey on network innovation using OpenFlow.

III. OpenFlow specification

The OpenFlow specification describes an open protocol to allow software applications to program the flow table of different switches. An OpenFlow architecture consists of three main components: an OpenFlow-compliant switch, a secure channel and a controller, as shown in Fig 1. Switches use flow tables to forward packets. A flow table is a list of flow entries. Each entry has match fields, counters and instructions. Incoming packets are compared with the match fields of each entry and if there is a match, the packet is processed according to the action contained by that entry. Counters are used to keep statistics about packets. The packet can also be encapsulated and sent to the controller.

The controller is a software program responsible for manipulating the switch's flow table, using the OpenFlow protocol. The secure channel is the interface that connects the controller to all switches. Through this channel, the controller manages the switches, receives packets from the switches and sends packets to the switches. An OpenFlow-compliant switch must be capable of forwarding packets according to the rules defined in the flow table. Figure 2 shows a high level description of how a network device processes a packet. First, the communication between the switch and the controller is possible through flow table rules. Internally, a switch uses Ternary Content Addressable Memory (TCAM) and Random Access Memory (RAM) to process each packet.

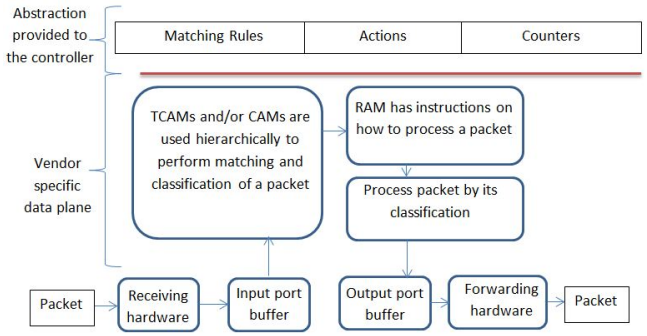

Fig. 2. Elements of an OpenFlow-compliant switch.

TABLE II
MATCH FIELDS OF A FLOW TABLE ENTRY IN AN OPENFLOW 1.0.0 SWITCH.

Ingress Port
Ether src
Ether dst
Ether type
VLAN id
VLAN priority CoS
IP src
IP dst
IP Proto
IP ToS bits
TCP/UDP src port
TCP/UDP dst port

Different versions of the OpenFlow protocol specification are available. The first version was the OpenFlow version 0.2.0 released in March, 2008. Versions 0.8.0 and 0.8.1 came next in May, 2008. Version 0.8.2, released in October, 2008, added the Echo Request and Echo Reply messages. Then, version 0.8.9 was released in December, 2008. It included IP netmasks, additional statistic information and several other updates. OpenFlow 0.9 was released in July, 2009. Finally, OpenFlow version 1.0, the most widely deployed version, was released in December, 2009. Next, we focus on versions 1.0.0 [17], 1.1.0 [18], 1.2 [19] and 1.3.0 [20], as previous versions are now deprecated. A detailed list of changes included in every version is available in the OpenFlow 1.3.0 specification document [20].

A. OpenFlow 1.0.0

Currently, the most widely used specification is the version 1.0.0. A switch supporting OpenFlow specification 1.0.0 uses 12 header fields present in the header and payload of the Ethernet packets coming into the switch. Table II shows all the header fields.

A packet can be matched to a particular flow entry in the flow table by using one or more header fields of the packet. A field in the flow table can have the value of ANY and it will match all packets. If the forwarding table is implemented using Ternary Content Addressable Memory (TCAM), ANY can be implemented in the switch hardware using the third masking state of the TCAM.

In Fig. 2 we showed the main elements of an OpenFlow switch. Figure 3 shows the details of the data plane in an OpenFlow 1.0.0 switch. In step 1, the Ethernet packet entering the switch goes to a packet parsing system. In step 2, the header fields are extracted and placed in a packet lookup header, as they are used for matching purposes. In step 3, the packet lookup header generated is sent to the packet matching system. In step 4, the packet lookup header is compared to the rules defined for each flow entry in the OpenFlow flow table. Note that the flow entries in the table are present in the descending order of priority. Therefore, the comparison of the packet lookup header is done starting from the first flow entry on the flow table. If a match is found, the actions in the matched flow entry are performed on the packet (step 5B). Otherwise, the first 200 bytes of the packet are sent to the controller for processing (step 5A).

TABLE III
MATCH FIELDS OF A FLOW TABLE ENTRY IN AN OPENFLOW 1.1.0 SWITCH.

Ingress port
Metadata
Ether src
Ether dst
Ether type
VLAN id
VLAN priority
MPLS label
MPLS EXP traffic class
IPv4 src
IPv4 dst
IPv4 proto / ARP opcode
IPv4 ToS bits
TCP/UDP/SCTP src port. ICMP Type
TCP/UDP/SCTP dst port. ICMP Code

B. OpenFlow 1.1.0

In the OpenFlow 1.1.0 specification, a switch contains several flow tables and a group table, instead of just one single flow table, as in OpenFlow 1.0.0. Figure 4 shows the main components of the OpenFlow 1.1.0 switch. The match fields are also different, as shown in Table III. We have highlighted in bold the added cells. The metadata field is used to pass information between the tables as the packet traverses through them. It is a register used to carry information between the tables. The Multiprotocol Label Switching (MPLS) fields are used to support MPLS tagging.

The processing of a packet entering the switch has changed as there are multiple flow tables available in the switch. The flow tables in the switch are linked to each other through a process termed as pipeline processing.

Pipeline processing involves a set of flow tables linked together to process the packet coming in. When the packet first enters the switch, it is sent to the first table to look for the flow entry to be matched. If there is a match, the packet gets processed there and if there is another table that the particular flow entry points to, the packet is then sent to that flow table. This happens until a particular flow entry does not point to any other flow table.

The flow entries in the flow tables can also point to the group table. The group table is a special kind of table designed to perform operations that are common across multiple flows. This means that actions belonging to a set of flows are grouped

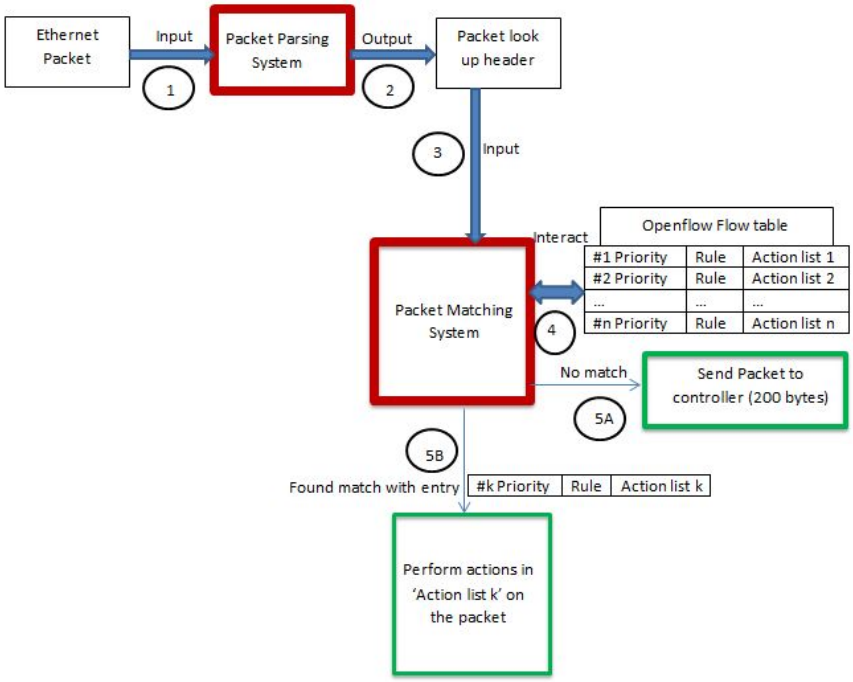

Fig. 3. How a packet is processed and forwarded in an OpenFlow 1.0.0 switch.

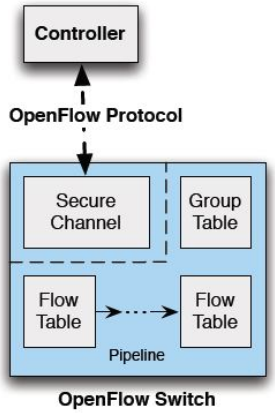

Fig. 4. Components of an OpenFlow 1.1.0 switch. Source: [18].

together. Also, the set of flows is controlled to perform various actions collectively under a single group. Complex forwarding actions such as multipath and link aggregation are enabled through the group table.

Finally, specification 1.1.0 introduces instructions instead of actions. Previously, an action was associated to each flow table entry. That action could be to forward the packet or to drop it, as well as processing it normally as it would be in a regular switch. Instructions are more complex and they include modifying a packet, updating an action set or updating the metadata.

C. OpenFlow 1.2

The OpenFlow specification version 1.2, was released in December 2011 and it includes a few major features. First of all, support to IPv6 addressing is added. Matching could be done using the IPv6 source and destination addresses. Another important feature supported is the possibility of connecting a switch to multiple controllers concurrently. The switch maintains connections with all the controllers and these can communicate with each other to do hand overs. Having multiple controllers provides faster recovery during failure and it is also possible to achieve load balancing.

D. OpenFlow 1.3.0

The OpenFlow specification version 1.3 was released in June 2012. Some of the improvements over version 1.2 are listed next. It is possible to control the rate of packets through per flow meters. Also, auxiliary connections between the switch and the controller have been enabled. Another improvement is that cookies can be added to the packets sent from the switch to the controller and specific durations field have been added to most statistics. A complete list of changes is available in the specification's document [20].

Table IV compares specifications 1.0.0, 1.1.0, 1.2 and 1.3.0.

E. Implementing applications using OpenFlow

In order to run applications on top of a single controller to manipulate the flow table of a switch, a network operating system is required (see Fig. 1). It acts as an intermediate layer between the OpenFlow switch and the user application. The network operating system communicates with the switch using the OpenFlow protocol and notifies the application of network events. Nox [21], Beacon [22] and Maestro [23] are examples of network operating systems. Recently, Big Switch released Floodlight [24], an open source Java based controller. Foster et al. [25] proposed Frenetic, a network programming language that simplifies the development of applications on top of network operating systems. NEC proposed Trema [26]

TABLE IV
COMPARISON OF OPENFLOW SPECIFICATIONS.

Specification	1.0.0	1.1.0	1.2	1.3.0
Widely deployed	Yes	No	No	No
Flow table	Single flow table	Multiple flow tables	Multiple flow tables	Multiple flow tables
MPLS matching	No	Yes	Yes	Yes, bottom of stack bit added
Group table	No	Yes	Yes	Yes, more flexible table miss support
IPv6 support	No	No	Yes	Yes, new header field added
Simultaneous communication with multiple controllers	No	No	Yes	Yes, auxiliary connections enabled

to develop OpenFlow applications using Ruby and C. Finally, DreamersLab developed Node.flow [27], a package to build a JavaScript based flow controller using Node.js [28]. Table V summarizes comparative data for the OpenFlow controllers that we have mentioned.

There are at least four possibilities to implement OpenFlow-based applications. First, an OpenFlow-compliant hardware switch can be used. We have provided a list in Table I. It is also possible to implement an OpenFlow-compliant software-based switch using Open vSwitch [29], [30]. A third option is to deploy virtual networks using Mininet [31], a virtual environment developed by the Stanford University that can be used to simulate multiple hosts in virtual network within one single host machine. Finally, a NetFPGA platform can be used. It consists of a PCI card that provides four 1G Ethernet ports, static RAM and other network functionalities [32]. The NetFPGA is also available with four 10G Ethernet ports.

Since physical and virtual switches can be used to deploy an OpenFlow network, it is important to note some similarities and differences between them. The advantage of a virtual switch is definitely the cost. Open vSwitch can be downloaded for free and it can be installed using commonly used virtual machine tools. A virtual switch performs the operations shown in Fig. 2 and Fig. 3 in software. Therefore, its main drawback is the performance. Hardware based switches perform data plane operations faster.

It is worth mentioning that debugging network applications is not a common technique yet. However, a first prototype of a debugger has recently been proposed by Handigol et al. [33].

Using OpenFlow, experimental and production traffic can share the same OpenFlow switch. The action of a flow table entry of an OpenFlow switch can be to send the packet to the switch data path. On the other hand, a different flow entry can be defined for experimental traffic. This way, experimental traffic can be tested without interfering with the production traffic [1]. In order to further enhance this, Sherwood et al. proposed FlowVisor [36]. Using this technique, it is possible for several single controllers to share the control of a switch. A centralized OpenFlow- based controller "slices" the network and acts as an intermediate layer between the switch and all the OpenFlow controllers that manipulate the switch.

F. OpenFlow: a specification, a protocol or an architecture?

OpenFlow can be viewed as a specification when it is in the context of an OpenFlow switch. An OpenFlow switch is achieved by implementing the requirements specified in the OpenFlow specification, in the device. For instance, in the OpenFlow specification, it is required that the switch has to support the flood action on the packets belonging to a particular flow. The flood action floods the packet using the normal pipeline of the switch [18]. Whether or not to implement this feature is a decision made by the vendor, but an OpenFlow switch must provide this functionality.

The OpenFlow protocol deals with defining the format of the messages passed between the control plane and the OpenFlow switch through the secure channel. The format of the messages has to be understood as well as generated by both the entities. This standard format of message passing is defined in the OpenFlow protocol. In fact, the OpenFlow protocol is part of the OpenFlow specification and it applies to the OpenFlow control plane as well as to the OpenFlow switch.

Finally, OpenFlow is viewed as architecture in the context of an entire network. In an OpenFlow network, OpenFlow switches are being controlled by one or more OpenFlow controllers. Such a network can be viewed as supporting the OpenFlow architecture.

It is important to keep in mind that the data plane implementation of the switch is vendor specific. As long as a switch can communicate with an OpenFlow controller, the data plane can be implemented differently by each vendor. Therefore, the fact that two switches are OpenFlow-compliant does not make them equal. Actually, not all switches implement all the features of the OpenFlow specification. It is possible that an OpenFlow-based application works using one switch but does not work using a different switch.

G. OpenFlow and SDN

Since OpenFlow has become the most popular SDN technology, some consider these terms as synonyms. However, it is important to note the difference between them. SDN consists of decoupling the control plane from the data plane, whereas OpenFlow describes how a software controller and a switch should communicate in an SDN architecture. SDN gives the user an abstraction of the network-wide state and OpenFlow abstracts a network component. As an analogy, an operating system provides a system-wide abstraction, just like SDN provides a network-wide abstraction. On the other hand, just like the operating system communicates with hardware through drivers, OpenFlow can be considered a driver to communicate a single controller and a network component.

As an SDN technology, OpenFlow networks have specific capabilities that we describe next.

IV. CAPABILITIES OF OPENFLOW

OpenFlow architectures allow centralized control of the network, software-based traffic analysis, dynamic updating

TABLE V
OPENFLOW CONTROLLERS.

Controller	Language	Created by	Comments
NOX	C++	Nicira Networks	NOX was donated to the research community in 2008. It has several branches at Stanford University, such as classic NOX, new NOX and POX. New NOX is the version that will be further developed. POX supports Python and it is used for educational or research applications [34].
Beacon	Java	Stanford University	Supports both event-based and threaded operation. Mostly used for research and experimentation [22].
Maestro	Java	Rice University	Licensed under licensed under LGPL v2.1. Not as common as other controllers such as NOX [35].
Floodlight	Java	Big Switch Networks	Forked from Beacon and extended for enterprise usage. Apache-licensed [24].
Trema	Ruby and C	NEC	Supports Linux applications only [26].
Node.Flow	JavaScript	DreamersLab	Works on top of Node.js, a platform built on Chrome's JavaScript runtime [27], [28].

of forwarding rules and flow abstraction. In this section we describe these capabilities and we give examples that illustrate how they can be exploited.

A. Centralized control of the network

One important capability of an OpenFlow network is that the controller has network-wide knowledge of the system. Several OpenFlow switches can be connected to a single controller and it is then possible to make decisions in a centralized manner. Instead of having several network devices with a limited knowledge of the network, a single controller can take decisions based on its knowledge of a broader part of the network.

One example of this is Ethane [37], an architecture proposed for managing the network of an enterprise. The key idea is to create a centralized policy that is managed by the controller. The switches become simple machines that forward and drop packets according to the rules defined by the controller. Using this architecture, it is possible to manage the network policies using high-end names. Routing decisions are also considered by the policy and finally, it becomes easier to bind a packet to its origin.

Another example of this capability deals with link failure recovery. In a traditional network, each switch has a limited knowledge of the network. When a link fails, then routes get adjusted at each switch until new routes are found. In an OpenFlow network, a centralized controller can find new paths in a much faster and easier way.

A comparison between the Path Computation Element (PCE) [38] architecture and OpenFlow is worth being mentioned when discussing this capability. Path computation in large and complex networks may require cooperation between different domains. The PCE architecture was proposed to address these challenges. A PCE is an entity that is capable of computing a network path or route based on a network graph [38]. A PCE architecture is not fully centralized. However, a cooperation between different entities does exist. Nevertheless, it can also occur that an entity does not have visibility over another element. Therefore, the knowledge of the network is not full. In OpenFlow-based networks, the controller usually has a broader knowledge of the network and therefore the control of the network is centralized. On the other hand, OpenFlow controllers do not cooperate together as it happens in a PCE architecture. Giorgetti et al. [39] propose OpenFlow and PCE architectures to control wavelength switched optical networks.

To illustrate the difference between PCE and OpenFlow architectures, we describe how the OSCARS [40] (On-Demand Secure Circuits and Advance Reservation System) project provides a PCE module [41]. Through this module, researchers can deploy PCE elements in the network in a distributed manner. Therefore, it is possible to perform path computation without using a single centralized point. If we compare this to an OpenFlow testbed, we will find that researchers deploy the code on top of an OpenFlow controller and all computations are performed from there.

Another centralized approach towards network management is the Bandwidth Broker (BB) architecture [42]. A BB consists of one or more servers that perform network functionalities such as quality of service (QoS), policy enforcement or admission control. The data plane communicates with the BB modules. The advantage of this architecture is that part of the complexity is assumed by the BB and minimal configuration is required in the network device. This architecture can be used at the edge of a network to control bandwidth allocation.

B. Software-based traffic analysis

Software-based traffic analysis is a powerful capability of OpenFlow networks. This capability greatly enables innovation, as it is possible to improve the capabilities of a switch using any software-based technique. Traffic analysis can be performed in real time using machine learning algorithms, databases and any other software tool.

As an example, a distributed denial of service attack (DDoS) detection method is proposed in [43] and it heavily relies in traffic analysis. The method is based on retrieving traffic data on periodic intervals and using self organizing maps to classify traffic as normal or malicious. Because the traffic analysis is done by software, there are more possibilities of using advanced features to perform the analysis, such as neural networks.

Another application of this capability is source address validation. Yao et al. [44] proposed checking the source address of each new flow. When a switch forwards a packet to the controller because it does not match any rule in the flow table, the controller can validate whether or not that source address corresponds to a valid flow.

C. Dynamic updating of forwarding rules

Another capability of OpenFlow networks is that they allow dynamic updates of forwarding rules. All kinds of changes

in the topology can be performed in real time, based on the decisions taken by a software controller. No human interaction is required. This is possible because the controller can modify the flow table entries at any time.

In [45], the controller is notified of a link failure and it modifies the entries of the flow table to re-route the traffic. By doing this, the network can react to link failures without requiring any action by the network administrator. The authors also suggest that the controller can automatically allocate more or less bandwidth according to the traffic load, to save energy.

Another application of this capability is load balancing. The controller can assess the load of several servers and dynamically change the forwarding rules to make sure that the load is properly balanced. Handigol et al. [46] proposed Plug-n-Serve, a load balancer that can dynamically add new servers to the cluster without interrupting the service.

D. Flow abstraction

Finally, networks using OpenFlow abstract all traffic as flows. For each flow there is an entry in the flow table. For each entry, different rules can be defined. One flow could be all traffic using one specific TCP protocol. Another could be all packets travelling between two defined MAC addresses or all data with one IP address destination. One could also define a non standard header to identify traffic of a specific entry. This allows managing different kinds of flows using the same control element.

Merging packet and circuit networks in a single infrastructure has been studied by several authors and it relies on this capability. Packet and circuit networks are treated as two different flows but they can be managed by the same controller.

In the next section we survey how the capabilities described above have been exploited in OpenFlow-based applications.

V. OpenFlow-based applications

In this section we survey studies that use OpenFlow for different kinds of applications. Ease of configuration, network management, security and availability are examples of these applications. OpenFlow has also been used to achieve network and data center virtualization, as we describe next.

A. Ease of configuration

OpenFlow-based applications can simplify the configuration of the network. Common approaches include access control lists and configuration files whose administration is time consuming and can lead to errors. By using SDN, it is possible to use software to take care of this. Yamasaki et al. [52] proposed using OpenFlow to manage the VLANs of a campus network. They describe how the number of VLAN ids is limited and how the configuration tasks are time consuming. In their approach, the controller analyzes incoming traffic and detects if the communication should be allowed or not, based on virtual group ids (GID) instead of VLANs. Using this approach, the number of VLANs limitation is overcome and the configuration of the network is simplified.

Several authors have addressed how to ensure consistent network updates using SDN. Reitblatt et al. [53] describe how to provide abstract operations that allow updating rules across the entire network in one fell swoop. In another paper, Reitblatt et al. [54] describe how updating network policies can lead to inconsistencies when packets are processed by both the old and the new policy. The authors note that achieving per-packet and per-flow consistency is critical to avoid inconsistencies and they describe techniques to implement both features. Also, Katta et al. [55] introduce algorithms that trade time against TCAM space in order to do the updates in an efficient manner. McGeer [56] proposes a network update protocol as well. His method uses boolean formulas and it ensures that flows are treated consistently. As an example, if a ruleset 1 is updated to a ruleset 2, the protocol ensures that the packets that were being processed using ruleset 1 are conserved, then the update takes place in all routers and finally the packets are released and processed by ruleset 2. Finally, Ghorbani et al. [57] propose a method to migrate virtual machines in a consistent manner and respecting bandwidth requirements. The authors have implemented an algorithm that outputs the order in which virtual machines must be migrated in order to ensure that no inconsistencies occur.

As we described earlier, Casado et al. [37] proposed Ethane, an SDN architecture explicitly designed to simplify the management of the network in an enterprise. Ethane relies on the idea that the network policy should be known by the controller and enforced in all switches. The main requirement is that all communications between two hosts require explicit permission. Instead of creating configuration files for all the switches in the network, these devices are kept simple and the rules are managed by the controller. An implementation of an Ethane switch in hardware is described in [58].

Some common points can be extracted from these studies. We mentioned in Section II that a user is interested in defining policies and configuring how their packets are forwarded. Here we notice that the studies by Reitblatt et al. [54] and by Casado et al. [37] focus on simplifying the creation of policies and hiding how these policies are implemented underneath. The study by Yamasaki et al. [52] provides another way of creating VLANs in such a way that the user must not deal with troublesome configuration files.

B. Network management

Deploying OpenFlow-based networks has also motivated research on OpenFlow management infrastructures. These studies aim at simplifying network management through OpenFlow. Mattos et al. [59] implemented a user friendly interface that allows the user to manage the network. Their implementation is based on NOX. Several applications are developed on top of that network operating system and a web based interface is provided to the user. Also, a multiagent system is capable of autonomously perform management.

Gibb et al. [60] propose an architecture in which network appliances (middleboxes) are not located at points of the topology that are traversed by plenty of traffic. They argue that these chokepoints are not suitable for middleboxes, as performance and correctness issues arise. Instead, they suggest using processing units in waypoints of the network. An OpenFlow switch, located at the chokepoint, is capable of

TABLE VI
COMPARISON OF SECURITY APPLICATIONS USING OPENFLOW.

Publication	Problem approached	Description of the solution	Implementation	SDN capabilities exploited
Suh et al. (CONA) [47]	DDoS attack detection	Frequency and pattern of requests are analyzed to detect DDoS attacks.	NetFPGA-OpenFlow switches	Traffic analysis and dynamic rules updating
Braga et al. [43]	DDoS attack detection	Statistic information in the flow table is used to classify traffic as normal or malicious.	Simulation of a NOX based network.	Traffic analysis and centralized control.
Chu et al. [48]	DDoS attack detection	Locator/ID separation protocol (LISP) is used to identify authorized and malicious sources.	Small network with one controller and two OpenFlow switches. Specialized hardware simulates DDoS attacks.	Traffic analysis and dynamic rules updating
Liu et al. [49]	Covert channel protection	The controller uses a second software node that filters authorized communication.	Simulation of a network using a virtual OpenFlow switch.	Dynamic rules updating and centralized control.
Yao et al. (VAVE) [44]	Source address validation	The controller analyzes traffic and calculates the flow path to decide if the source address is valid.	Simulation of a network using a virtual OpenFlow switch.	Traffic analysis and dynamic rules updating.
Jafarian et al. [50]	Moving target defense	The controller periodically assigns different virtual IP addresses to hosts to hide the real IP addresses to an intruder.	Simulation using Mininet.	Centralized control, dynamic rules updating.
Gutz et al. [51]	Traffic isolation	Network slices are defined through a programming language instead of using network-level techniques.	A tool was developed to test whether traffic isolation was correct	Centralized control

routing to the processing units only the traffic that needs to be processed by the middlebox. By doing this, less traffic traverses the network appliances and a much simpler hardware is used at the chokepoint of the network.

Defining and implementing network policies has also been addressed using OpenFlow. Voellmy et al. [61] propose Procera, a controller architecture and a high level network control language that can be used to reactively define network policies. Regarding implementation, Fergusson et al. [62] propose an OpenFlow-based method to perform policies delegation in SDN networks. Their idea consists of creating delegation trees, where each path can be managed by different network administrators. The authors create hierarchical flow tables that can be used to delegate policies. An incoming packet is matched to these policies and processed accordingly.

Finally, an innovative way of managing IP multicast in overlay networks was proposed by Nakagawa et al. [63]. The authors propose using OpenFlow instead of a more common approach such as Internet Group Management Protocol (IGMP). Two important contribution of their approach are eliminating periodical join/leave messages and making use of multipath in the layer-2 network.

Outsourcing network functionality is another interesting innovation to simplify the network management. Gibb et al. [64] propose Jingling, an architecture that allows adding functionality to a network in an outsourced manner. Feature providers can be located anywhere outside the network. Policies defined how feature providers must be used and a network controller maps the policies to the feature providers. Following the idea of having services outside the network, the idea of Networking-as-a-Service (NaaS) has emerged. Raghavendra et al. [65] propose using OpenFlow to manage networks in such a way that they are ready to user services provided as NaaS.

In this section, we notice that the common trend is to exploit how OpenFlow can dynamically update the forwarding rules. Having a network-aware controller allows the network manager to dynamically forward traffic according to specific needs. Once again, we also note how several studies simplify the creation of network policies.

C. Security

OpenFlow has also been used to create applications that provide security to the network. Table VI compares the problems approached, the solutions proposed and the infrastructures used to test the implementations.

Methods to detect DDoS using OpenFlow have been proposed recently [47], [43], [48]. Suh et al. [47] proposed a content oriented networking architecture. This approach relies on creating flows based on the identity of the client and the type of content requested. A DDoS attack is detected when the server that provides a given content type receives more requests than expected, based on a pre-defined range. Chu et al. [48] proposed a method that analyzes the frequency of traffic. If a threshold is exceeded, then the controller considers that a DDoS attack is happening and it starts dropping packets. Finally, as we mentioned earlier, Braga et al. [43] proposed a method that gathers traffic information and uses self organizing maps to classify the traffic as normal or malicious.

Liu et al. [49] proposed an SDN architecture where nodes with different levels of security clearance can exchange communication. The OpenFlow controller sets up the rules so that traffic is authorized only when the requester has a higher security clearance than the receiver.

Yao et al. proposed VAVE [44], an OpenFlow-based architecture designed to validate the address of all incoming packets. When the switch receives a packet that does not match any rule, the packet is sent to the controller and the source address is validated. If spoofing is detected, then a rule is created to stop that traffic.

Jafarian et al. [50] propose a moving target defense (MTD) technique using OpenFlow. The proposed defense assigns virtual IP addresses to hosts and the controller maps virtual addresses to physical addresses. This is performed once and again, in an unpredictable way such that the attacker cannot identify which host is behind each IP address.

Finally, traffic isolation has been studied by Gutz et al. [51]. The authors argue that current traffic isolation techniques such as VLANs increase the complexity of the network configuration. They propose creating network slices at a higher level. Under their approach, a network programming language should be able to create this slices to isolate traffic. This way, slices are defined at a high level and then forwarding rules are automatically added to the switches.

When it comes to security, we notice how the researchers heavily rely on the ability of processing data in the controller. In all these publications, some kind of intelligence is added to the switch through the controller. For example, Braga et al. [43] use self organizing maps, which could not be implemented on regular switches. Also, Yao et al. [44] exploit the idea that, since a given packet must be analyzed by the controller, then a more rigorous address validation can be performed. Once again, in the study by Gutz et al. [51], we note how more capability is given to a higher layer. In this case, it is about isolating network traffic using a programming language. This is a common trend in SDN: how to allow a user to perform network tasks without needing full access to the network topology.

D. Availability

OpenFlow-based applications have focused on providing availability to the network as well, including load balancing and fault tolerance. Load balancing is a commonly used technique to distribute a working load between two or more nodes. This improves the availability of a network since the system can support one or several single failures. Fault tolerance refers to the property of a system to continue operating when a failure occurs.

1) Load balancing: Handigol et al. proposed Plug-n-Serve [46], a load balancer for unstructured networks that attempts to reduce the response time by taking into consideration the load of the servers and the congestion of the network. The proposed method displays the load of the network in real time. The software running on the controller takes the load of the network and servers into consideration and decides where to direct the traffic. Using this solution, it is also possible to add new servers to the cluster and the software will dynamically detect them and add them to the load balancing. An improved version of Plug-n-Serve, Aster*x was also proposed in [74]. Aster*x runs on the Global Environment for Network Innovations (GENI) infrastructure and it is used at a much larger scale than Plug-n-Serve.

Wang et al. [75] argue that Plug-n-Serve works by reactively creating forwarding rules for incoming requests. They proposed a proactive approach, based on wild cards. They divide the entire client address space into different rules. These rules forward the traffic to specific servers. The controller knows what percentage of traffic should be handled by each server and it creates the rules so that the expected loads are respected. We can see that the approach by Wang proactively creates the rules to make sure that each server handles the required percentage of connections. This requires a smaller number of rules than the approach used by Plug-n-Serve, which improves its scalability. On the other hand, Plug-n-Serve takes into consideration the load of the server and the network and does not require a specific percentage of traffic for each server and it is more flexible, since each client can be handled individually.

2) Fault tolerance: Sharma et al. [76] and Staessens et al. [45] have explored fault tolerance using OpenFlow. In [76], the authors describe how failure recovery can be implemented using OpenFlow. They explain how the controller can dynamically change the routing rules when a failure is detected in a link. In [45], experiments are designed to analyze if an OpenFlow based network can recover from a link failure. The authors argue that carrier grade networks must be able to recover in less than 50 ms. The experiments show that restoration is successful but that the dependency on the centralized controller makes the goal of 50 ms challenging to achieve.

Another way of ensuring availability is to verify that there are no configuration errors that might cause a disruption. Khurshid et al. [77] propose VeriFlow to check network invariants in real time. This includes loops in the routing tables, unavailable paths and other problems that can be identified before deploying the network. Moreover, the authors are interested in doing this in real time. VeriFlow sits between the controller and the switch and monitors the communication between these two parts. By modelling the network as a graph, network invariants are checked in the order of hundreds of microseconds.

Porras et al. [78] propose a policy enforcement mechanism that is also based in analyzing the forwarding rules that are added to or deleted from the flow table. The author introduce FortNOX and they aim at performing role based authentication and security constraint enforcement. The application checks for conflicting rules after every update of the flow table. When two rules incur in a contradiction, then the rule defined by the user with the highest security clearance is kept.

These studies have some common trends. First, the capability of dynamically updating forwarding rules is heavily exploited. Load balancing is performed based on the ability of the controller to alter the forwarding rules. The fact that the controller is network-aware is also helpful. In the studies by Sharma [76] and by Staessens [45], finding new paths after a failure occurred is easily done in a centralized manner, since the topology is known. Traditionally, this kind of recovery is done by decisions taken by switches that are not network-aware and a centralized method simplifies this task.

TABLE VII
COMPARISON OF NETWORK VIRTUALIZATION APPLICATIONS USING OPENFLOW.

Publication	Problem approached	Description of the solution	Implementation
Simeonidou et al. [66]	Packet and circuit network integration	An OpenFlow controller is integrated with a GMPLS controller	No implementation provided
Das et al. [67]	Packet and circuit network integration	An OpenFlow controller is integrated with a GMPLS controller	Prototype network using NetFPGA switches that emulates a WAN
Das et al. [68]	Packet and circuit network integration	An OpenFlow controller is integrated with a GMPLS controller	Fully functional hardware based network. Used as a proof of concept for a demonstration.
Das et al. [69]	Application aware aggregation and traffic engineering in a circuit-packet network	The capabilities of SDN are exploited in a circuit-packet network to provide application aware routing.	Hardware based network used to emulate a WAN
Das et al. [70]	Complexity of IP/MPLS control plane	The MPLS data plane is controlled by OpenFlow instead of the traditional IP/MPLS control plane.	Open vSwitch and Mininet are used to emulate a WAN
Ferkouss et al. [71]	Flexibility of MPLS nodes	An OpenFlow controller is used to dynamically modify MPLS nodes	Hardware implementation that exploits the pipelining of OpenFlow 1.1.0.
Kempf et al. [72]	Supporting MPLS forwarding in OpenFlow 1.0.0	Additional match fields are added to the flow entry format and MPLS actions are added to the OpenFlow 1.0 specification	NetFPGA-OpenFlow switches
Sharafat et al. [73]	MPLS implementation complexity	The centralized control capability is exploited to implement MPLS-TE and MPLS-VPN in a simpler way than the traditional approach	Physical and virtual switches supporting the MPLS section of OpenFlow 1.1 and simulation using Mininet

E. Network virtualization using MPLS and GMPLS

Network virtualization is another research area where OpenFlow has been applied. Circuit and packet switched networks are typically managed using separate infrastructure and this is costly. Several authors have proposed OpenFlow-based architectures that could be used to manage both packet and optical circuit networks using the same infrastructure [66], [67], [68], [69], [79]. Azodolmolky et al. [79] provide a good explanation on how OpenFlow and GMPLS can be used together as an integrated control plane. This approach relies on the fact that packet and optical circuit networks can be managed as different flows in the switch's flow table. In order to manage both flows, a GMPLS controller is integrated to the standard OpenFlow controller. The OpenFlow controller is responsible for managing the flow table. However, when a flow corresponds to traffic over an optical circuit, then the GMPLS controller takes care of the routing decisions and a flow entry containing the forwarding action and the required wavelength is added to the flow table. This way, switches can handle two kind of flows, one for circuit networks and one for packet networks.

MPLS and GMPLS have also been used in other applications. Kempf et al. [72] add an extension to OpenFlow 1.0 that allows a switch to forward MPLS on the data plane. Das et al. [70] proposed using MPLS in the data plane but OpenFlow in the control plane instead of the traditional IP/MPLS control plane. El Ferkouss et al. [71] argue that OpenFlow can be used to "deossify" an MPLS architecture. They show how an MPLS node can play multiple roles for different MPLS domains, which provides greater flexibility to the nodes. Sharafat et al. [73] implement MPLS-TE and MPLS-VPN using an OpenFlow controller to show that centralized control makes the implementation easier. Table VII compares the different applications that use OpenFlow to virtualize networks using MPLS and GMPLS. Centralized control, dynamic rules updating and flow abstraction are the most commonly exploited capabilities for these applications.

The studies that we have mentioned exploit the circuit switching capability of GMPLS and not the VLAN-switching capability. In summary, the research direction regarding GMPLS and OpenFlow is to simplify the creation of end-to-end circuits. Das et al. [80] discuss why GMPLS has not been as successful as expected in the control plane and how combining it with software defined networking is a more suitable approach.

F. Data center virtualization

Similar to network virtualization, virtualizing data centers using OpenFlow has also been an active research area. SDN architectures have been considered to meet the requirements of a data center: efficiency, agility, scalability and simplicity [81]. Al-Fares et al. [82] proposed Hedera, a dynamic flow scheduling method for data center networks. They proposed an OpenFlow-based architecture that can dynamically modify the flows according to the traffic load. The authors argue that this approach achieves a larger network utilization. Rotsos et al. [83] also use OpenFlow to dynamically virtualize the network. They argue that VLANs and MPLS can be used to create virtual networks in a static way. However, the network utilization can be optimised if the network virtualization is performed according to the traffic load.

G. Wide area network applications

A majority of studies have deployed their experiments in local area networks. However, some studies address the possibility of deploying OpenFlow in a wide area network (WAN). First, in [70] the authors show that OpenFlow could be deployed in a WAN by emulating this kind of network.

Studies such as [67], [69] show that OpenFlow could be used to control this type of network.

Bennesby et al. [84] propose an inter-domain routing solution using an OpenFlow architecture running on a NOX controller. The authors explain how the different autonomous systems (or domains) interact with each other through the Internet. They propose a routing scheme based on OpenFlow that would allow autonomous systems to communicate with each other.

H. Wireless applications

OpenRoads [85] was designed to enable research in mobile networks. It can be considered as the wireless version of OpenFlow. In this architecture, a flow visor [36] (as introduced in Section III) controls network devices through the SNMP protocol. Several controllers can be deployed on top of the flow visor. Details of the OpenRoads architecture are available in [86]. A deployment of OpenRoads in the campus of Stanford University is described in [87]. Other works using OpenRoads include [88], [89].

There are also other wireless applications that do not use OpenRoads. Huang et al. [90] proposed PhoneNet, an infrastructure which supports group communication among phones. A group of users can interact using their phones after a multicast address is created so that it can be accessed by all the members in the group.

Bansal et al. [91] propose OpenRadio, a design for a programmable wireless network dataplane to automatize how devices' software is updated. They argue that software updates have become more frequent (there used to be a release every few years and now updates are available monthly). OpenRadio aims to providing an infrastructure to update base stations of wireless systems via software. Without this approach, devices must be collected so that the software can be manually updated. This frequent hardware collection is expensive and network software updates are more adequate. As an example, they describe that in an urban area, there could be one device per block to provide adequate coverage. In this scenario, collecting the sensors every time an update must be installed would be prohibitively expensive. OpenRadio enables updating the devices without having to physically collect them.

Regarding wireless enterprise local area networks (WLAN), Suresh et al. [92] propose Odin, a prototype SDN architecture that simplifies client management in a WLAN. The network is given programmability and light virtual access points are introduced. These access points are managed from an OpenFlow controller.

I. Other applications

OpenFlow has also been used in other areas not listed above, such as routing and network congestion control. Liu et al. [93] proposed a method to control congestion using queuing systems and a centrally controlled network. Yap et al. [94] also consider network congestion, as well as bandwidth reservation and multicast. Nascimento et al. [95] proposed QuaqFlow, a Quagga implementation using OpenFlow. Quagga is a routing package that provides implementation of TCP/IP routing protocols. RouteFlow [96], an architecture that provides routing as a service, was proposed as an extended work of Quagga. Rothenberg et al. [97] proposed an OpenFlow-based approach that allows the introduction of advanced routing systems. This study was built by extending the earlier RouteFlow [96]. Egilmez et al. [98] proposed an architecture to provide routing for video streaming.

In the next section, we focus on larger-scale deployments rather than the applications themselves.

VI. OpenFlow deployments

Deployments of OpenFlow-based networks mainly include campus networks and testbeds, as well as deployments undertaken by the industry.

Stanford University has deployed an OpenFlow-based network in one of its buildings. The network includes production, experimental and demonstration traffic. It connects approximately fifty switches and around 25 users, both wired and wireless. Details of the topology can be found at [99]. Other universities have also deployed OpenFlow-based networks. The full list is available at [2] and it includes Clemson University [100], Georgia Tech [101], Indiana University [102], Kansas State University [103], Rutgers University [104], University of Washington [105], University of Wisconsin [106] and Princeton University [2].

At a larger scale, the Global Environment for Network Innovations (GENI) [107] provides a research infrastructure where OpenFlow experiments can be conducted. The OpenFlow core of this network consists of several interconnected OpenFlow-compliant switches on both Internet2 [108] and National LambdaRail (NLR) [109] networks. The connection to the NLR network is achieved through HP6600 switches deployed at Sunnyvale, Seattle, Denver, Chicago, and Atlanta and through NetFPGA switches in Sunnyvale, Houston, Chicago, and New York [110]. Internet2 has OpenFlow-compliant switches installed in Los Angeles, New York, Washington DC, Atlanta [111]. Campus networks can connect to the GENI deployment to run larger scale experiments.

As of October 2012, Internet2 provides a nationwide 100G software defined network [113]. The network is currently operational for member institutions of Internet2. The deployment includes routers of the Brocade MLX family and related Brocade NetIron platforms, as well as Juniper Networks MX Series routers [114]. It also provides a 100G Ethernet network and a 8.8 Terabit per seconds optical network. Internet2 will operate the U.S UCAN (United States Unified Community Anchor Network) program [112]. Their goal is to use this software defined network to provide a platform to interconnect research, educational and health care institutions. Figure 5 shows a draft of the expected deployment.

The Energy Science Network (ESnet) [115] is funded by the Department of Energy (DOE) and operated at the Lawrence Berkeley National Laboratory. ESnet has also deployed an OpenFlow testbed, originally funded by the Advanced Networking Initiative (ANI) [116]. ANI was an investment in next-generation technology infrastructure to speed of scientific discovery. ESnet operates two testbeds: the Long Island Metropolitan Area Network (LIMAN) and the 100G. The LIMAN is a 10G testbed. It includes four NEC IP8800 OpenFlow switches [117]. The OpenFlow network operates on

Fig. 5. Draft of the planned U.S. UCAN network using the Internet2 100G deployment. (Source: [112]).

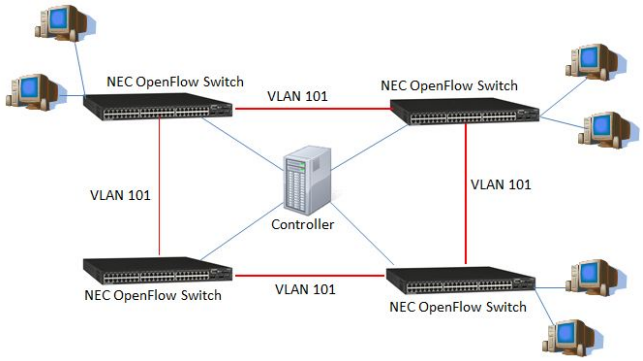

Fig. 6. Topology of the ANI OpenFlow testbed.

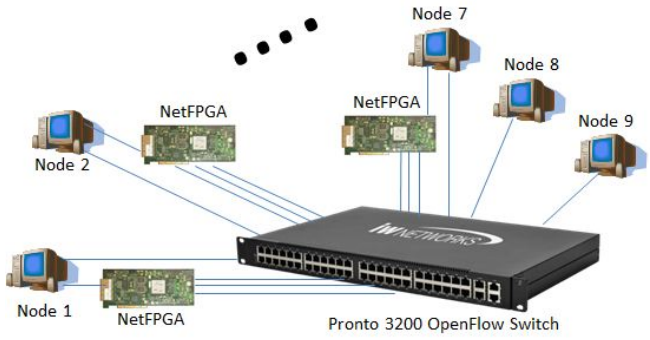

Fig. 7. Topology of the ORBIT OpenFlow testbed.

the VLAN 101. There are two ways of running an experiment on the testbed. One option is to connect the controller directly to the OpenFlow switches through the management VLAN. The second option is to connect to the flow visor controller and getting a partition of the network to run the experiments. The first option requires the researches to reserve the testbed beforehand. The second option does not require any reservation of resources. The flow visor configuration file has to be sent to the administrator to get connected. The 100G testbed runs between the DOE Supercomputer centers in Argonne National Lab (Chicago) and NERSC (California) through a 100G dedicated network [118]. To deploy experiments using the 100G testbed, researchers must follow a proposal process that includes writing a 1-2 page proposal and demonstrating that the experiment is working in a small environment [119]. Figure 6 shows the topology of the ANI OpenFlow testbed.

Another smaller deployment is the Open Access Research Testbed for Next-Generation Wireless Networks (ORBIT) testbed [120], which is being developed and operated by WINLAB, Rutgers University. It is intended to be used to test and evaluate innovative protocols in real-world settings and it includes an OpenFlow-based network. The deployment consists of an OpenFlow-compliant switch Pronto 3290 connected to nine nodes. Out of the 9 nodes, 7 of them are connected to one NetFPGA each. Each of the NetFPGA is connected to the Pronto 3290 OpenFlow switch through four 3GbE connections. All of the 9 nodes are connected to the Pronto 3290 OpenFlow switch and they are connected to a control plane through which the nodes can be accessed through telnet/ssh sessions by the experimenter. Figure 7 shows the topology of the ORBIT OpenFlow testbed.

Similar testbeds have been deployed in Europe and Japan as well. Ofelia is a project funded by the European Union that provides an OpenFlow-based network with nodes in Belgium, Switzerland, UK, Spain, Germany, Italy and Brazil [121]. Also, the Dynamic Network System (DYNES) project [122], funded by the National Science Foundation (NSF), is exploring technologies such as OpenFlow to interconnect campus, regional and backbone networks. Other future deployments also include the Network Development and Deployment Initiative (NDDI) and the Open Science, Scholarship and Services Exchange (OS^3E) [122].

OpenFlow has also been deployed by several companies, as seen in the keynote lectures of the 2012 Open Networking Summit [123]. As an example, Google has deployed OpenFlow in the inter-datacenter backbone network that carries all the traffic between the different datacenters [124]. Currently,

this network is completely OpenFlow based. According to the speaker, adopting OpenFlow has been the most significant change in networking in the company [125].

By surveying OpenFlow-based applications and deployments, we have identified some challenges faced by OpenFlow-based networks. We discuss these challenges next.

VII. Performance of OpenFlow-based networks

We have surveyed different OpenFlow-based applications and deployments. Next we mention several studies that have designed experiments to evaluate the performance of OpenFlow architectures. We also discuss publications that propose alternatives to improve the performance of OpenFlow networks.

A. Measuring and modelling the performance of OpenFlow-based networks

Jarschel et al. [126] model an OpenFlow controller as a M/M/1 queuing system. This model allows obtaining results regarding the total sojourn time of a packet through the system. The model also captures the difference in terms of delay between a packet that is processed by the switch and a packet that must go to the controller. Also, the probability of dropping a packet because the controller is under high load is studied. The results show that the sojourn time depends largely on processing speed of the OpenFlow controller. Also, the authors are able to conclude that the processing time of the controller lies between 220 and 245 μs. Another interesting result shows that current controllers cannot handle a big number of flows in 10Gbps links.

Bianco et al. [127] compare the performance of OpenFlow switching, link layer Ethernet switching and network layer IP routing. Experiments include using packets of different sizes and comparing the results of single flows against multiple flows. In all the experiments, OpenFlow achieves good results in comparison to link layer Ethernet switching and network layer IP routing.

Levin et al. [128] address the following question: "How does distributed SDN state impact the performance of a logically centralized control application?" [128]. The authors argue that the SDN network control plane cannot be fully physically centralized because responsiveness, reliability and scalability issues arise. One possible solution is to have a distributed control plane where a logically centralized control plane operates. This design faces consistency challenges and the authors study how much inconsistencies in the global network view affect the performance of the network. The authors compare two applications: one is ignorant to possible inconsistencies and the other takes inconsistency into consideration when operating. This study concludes that optimality is significantly affected when inconsistencies are not considered and that the robustness of an application is increased when it is aware of the network state distribution.

Heller et al. [129] address two important questions regarding reliability, scalability and performance. First, they analyze how many controllers are needed in a network. Second, they discuss where in the topology should these controllers go. The authors introduce these questions as an important part of the controller placement problem. Regarding the number of controllers needed, the authors analyze the latency of different topologies and they observe that one controller is often enough to keep the latency at a reasonable rate. They also explain that, in general, adding k controllers reduces the latency by a factor of k. However, they also show examples where this is not the case and more controllers are required. Regarding the placement of controllers, they show how this decision can also affect the latency of the network. They also show that randomly selecting the location of the controller yields results that are far from optimal.

Finally, the performance of OpenFlow has also been evaluated in the optical networks domain. Liu et al. [130] evaluate the performance of an OpenFlow-based wavelength path control in transparent optical networks. They study two different approaches for lightpath setup (sequential and delayed) and two ways of lightpath release (active and passive). The experimental setup includes four OF-PXCs connected in a mesh topology, with one OpenFlow switch and one client node attached to each OF-PXC. A photonic cross-connect (PXC) devices switches optical signals in an all-optical device. The results show that a path between two clients (thus traversing two switches) can be provisioned faster using the sequential approach. Also, releasing a path can be done faster if the active approach.

B. Improving the performance of OpenFlow-based networks

Several authors have also proposed modifications to OpenFlow or alternative ways of using it to increase the scalability, reliability or performance of the network.

Yeganeh et al. [131] propose Kandoo, a framework that aims at reducing the number of events that are received at the control plane of the network. To do this, two layers of controllers are used. The upper layer maintains the network-wide state. The bottom layer consists of several controllers that do not know the network-wide state and that are not interconnected. The bottom layer handles most of the events and reduces the overhead at the upper layer. This framework also increases the scalability of an OpenFlow network.

At least two studies have proposed additional ways to take profit of a CPU being connected to the switch. Mogul et al. [132] propose software defined counters. Recall that an OpenFlow switch collects statistic data for each flow. The authors explain that this data is stored in the switch using application specific integrated circuits (ASIC). The propose keeping and processing information in a CPU, where more variable and flexible statistics could be processed. The study does not include implementation or simulation results, but the feasibility of software defined counters is analyzed theoretically.

Lu et al. [133] also propose combining ASIC and CPU processing. The authors point out two limitations of current switches: a limited size forwarding table and a limited size packet buffer. They argue that a their approach relaxes these limitations by using a CPU. A prototype is developed and a 3.9Gb/s software forwarding throughput is achieved. Also, large TCP traffic bursts are absorbed without packet losses. The experimental setup consists of sending 50k bidirectional TCP flows among four servers.

Vanbever et al. [134] propose HotSwap, a system that enables correct and efficient upgrades of SDN controllers. The goal of HotSwap is to be able to change from one controller to another (when upgrading the controller is needed) without disrupting the network. They argue that stopping the old controller and starting the new one introduces delays and can also create errors in the network. HotSwap records relevant messages between the switches and the controller and bootstraps the new controller by replicating previous network events. By the time the new controller starts operating, the network state is the same as when the previous controller was operating.

VIII. Challenges of OpenFlow-based networks

OpenFlow deployments face several challenges that must be taken into consideration, including security, availability, scalability, reliability, expenditure and compatibility.

A. Security

One principal challenge of an OpenFlow-based network is the dependence on the controller. The controller becomes a component with a critical knowledge of the network and a very attractive target for an attacker. Security measures must be considered to ensure the availability of the controller. At the same time, since this component has access to all the network, it must be strongly protected from intruders.

The channel between the controller and the switches can also be vulnerable. According to the OpenFlow specification, Transport Layer Security (TLS) can be used to secure the communication. However, this feature is not a requirement and it is also acceptable to communicate the controller and the switches using plain text traffic. TLS can then provide security to the channel, but its usage depends on the design of the network since it is not required.

The flow table is a component that could also present security risks, although there are no published vulnerabilities yet. It is possible to manage a flow table from two different controllers, where one of them is a production hardware and the other one is just experimental. Since the latter one will be subject to lower security controls, it is important to make sure that the consistency of the flow table remains and that a malicious update coming from one controller will not tamper other flow entries. Currently, the flow visor takes care of those considerations but since OpenFlow is a recent protocol, this needs to be kept in mind.

A centralized software-based controller can also have security advantages. In a distributed network, many vulnerabilities must be addressed in different protocols and different devices. Having a software controller outside of the data plane can simplify how security is enforced, as there is plenty of expertise on securing servers through hardening instead of securing network devices.

B. Availability

The dependence on the controller is also a challenge regarding availability. An OpenFlow-compliant switch is capable of forwarding packets using cached rules. However, the communication with the controller is eventually needed for any kind of modification of the rules. One advantage of a traditional, distributed network architecture is that if a switch fails, the availability of the network can be maintained. In an OpenFlow network, the communication with the controller must be ensured. As we mentioned in the previous subsection, the controller becomes a single point of failure.

How to handle the delay needed to create new flows is also a challenge. When an OpenFlow switch receives a packet that does not match any rule in the flow table, then the first 200 bytes of the packet are sent to the controller. After this, the controller can install a new forwarding rule. Therefore, the delay to process the first packet is larger. If this delay is too large, then the availability requirements of a network might not be met.

C. Scalability

The controller can also become a bottleneck. If too many packets must be forwarded to the controller, then performance issues can occur. A well designed network should ensure that the most part of the traffic can be handled by the switches without needing to forward data to the controller. It is also important to assess whether the controller will become a bottleneck when the number of nodes grows. As we discussed in Section VII, authors have addressed this challenge while evaluating the performance of OpenFlow. In particular, Heller et al. [129] show how a single controller is usually enough to keep an acceptable latency. They also show that introducing k controllers reduces the latency by k.

OpenFlow-based architectures also face two important scalability challenges: a limited flow table size and hardware constraints. First, the number of flows that can be contained in the flow table is limited. It is still a challenge to handle a very large number of flows using an OpenFlow-compliant switch. Manipulating packets at the control plane is slow as well. Therefore, end-to-end traffic control is hard to implement if many different flows must be manipulated. Second, there are hardware limitations on the speed at which flows can be added. For these two reasons, it is still unclear if OpenFlow deployments can be used to control the core of a network. Currently, OpenFlow is being used at the edge of a network instead.

D. Survivability

The dependency on the controller also creates reliability issues. One example can be found in [45]. In this OpenFlow-based network, a link failure is reported to the controller and a new path is found. According to the results, the network recovers successfully but not quickly enough. The authors explain that the expected recovery time is not met because of the time lost contacting the controller. A common requirement by carriers is to achieve a network recovery in less than 50 seconds. In the study by [45], this goal is not met.

On the other hand, a centralized control also has advantages regarding network recovery. In a distributed network, recovering from a broken path can be a slow process. However, an OpenFlow controller is network-aware and it can find the new path faster.

A multipath proposal for OpenFlow addresses how to recover faster from failures. This proposal includes a fast reroute support, where backup flows can be installed in advance. If the switch detects that a specific port has lost connectivity, then the backup flow is installed. This is a proactive way of dealing with link failures and it has the advantage that the controller does not need to be contacted immediately after the failure.

E. CAPEX and OPEX

It has been debated whether OpenFlow can reduce the capital and operational expenses (CAPEX and OPEX) of an organization.

OpenFlow adopters argue that by moving the complexity to the software-based controller, network devices become simpler and therefore, cheaper. This would reduce the CAPEX. However, OpenFlow also has limitations and advanced hardware is still required to operate a network. It does not seem likely that network switches and routers will become simple commodities in the short term. Also, ensuring the availability of the control plane can increase the CAPEX. It is important that the controller remains reachable even in case of a failure in the data plane. Achieving this could increase the costs of a deployment.

A similar trade-off occurs for OPEX. We have discussed several studies that simplify the network configuration and management. Certainly, OpenFlow can be used to reduce the number of human based configuration tasks that are time consuming and error prone. This reduces the OPEX. On the other hand, moving the complexity of the network to the software control plane requires work. Project administrators, software developers, testers, debuggers and other costs are examples of expenses that must be incurred in an OpenFlow-based deployment. Therefore, it is not clear either whether OpenFlow greatly reduces the OPEX.

F. Compatibility

Another important challenge for OpenFlow deployments is that the network operating systems support specific versions of the OpenFlow specification. Currently, most of them support OpenFlow 1.0.0. Even though OpenFlow 1.1.0 has been available for several months, the network operating systems do not support specific features of the newer version. The challenge is then to upgrade both the OpenFlow specification and the software of each network operating system.

This compatibility issue also applies to the network devices, whose software must be updated to meet the requirements of new OpenFlow specifications. For instance, in the HP ProCurve switches series, modifying the packet header fields (for example: IPv4 destination address) in the switch hardware is not supported. But, it is possible to do the same in the switch software which is a slower path for processing. Therefore, it is likely that switch vendors would fine tune their hardware to support additional features in the switch hardware to improve efficiency. This updating process must be taken in consideration when new versions become available.

User developed applications face compatibility issues as well. We have shown how there are significant differences between specifications 1.0.0 and 1.1.0. Another example is that version 0.8.9 became deprecated when version 1.0.0 was available. Therefore, it is important to consider if applications running under version 1.0.0 will still work on version 1.1.0 or if all affected developments must also be updated. This scenario could occur again in further releases.

Finally, we believe that compatibility among controllers should also be taken into consideration. Currently, multiple network devices perform switching and routing in a standardized way. However, if the devices are controlled by software-based controllers, then standardization should be achieved too. Controllers from different domains should use the same protocols to ensure that the communication is possible between hosts in different domains.

Next we conclude the paper by discussing the future research directions in OpenFlow-based networks.

IX. Conclusions and Future Directions

OpenFlow is a promising technology for enabling advanced functionality in programmable networks. This survey paper is, in our opinion, the first one to discuss the capabilities, application, deployments and challenges of SDN/OpenFlow-based networks. We also explained and compared the OpenFlow specifications. Below, we identify future research directions in OpenFlow-based networks.

First of all, applications have been developed in areas such as security, ease of configuration, availability, network and data center virtualization, wireless applications and others. Currently, a majority of the surveyed applications consist of small, simple networks with some OpenFlow switches and hosts. Only a small number of studies demonstrate their work in a WAN. In [70], the authors emulated an OpenFlow-enabled WAN, but this is an exception to the majority of studies. Whether OpenFlow can be used in WAN deployments or not is still an open question. Studies show that OpenFlow could be used to control a WAN ([67], [69], [59]). However, scalability and performance experiments have not been conducted yet.

Second, we observe that OpenFlow switches have been used as a multi-layer network device. This technology was first proposed to control Ethernet switches. However, OpenFlow has also been used in routing ([75], [83], [94], [95]), IP address validation ([44]) and MPLS control ([66], [67], [68], [69], [70], [71], [72], [73]). This shows that OpenFlow can be used at multiple layers. Future directions include tighter integration of OpenFlow features with routers and MPLS switches to reduce their complexity and cost.

Third, we find an open problem in the design of OpenFlow architectures. So far, mostly all applications and deployments use only one controller to manage all the switches. Distributed architectures with more than one controller could be used to address some of the challenges such as availability or reliability [129]. In fact, a vast majority of networks contain duplication as a means to ensure the availability of the system. We believe that the possibility of communicating controllers in the OpenFlow 1.2 specification ([19]) is an opportunity to deploy this kind of architecture. Coordinating tasks across multiple controllers and using them during normal and failover conditions are tasks for future investigations.

Fourth, we believe that most studies do not involve real hardware but use virtualization tools such as Mininet [31] and Open vSwitch [30]. Also, the number of hosts is small in most of the applications. Scenarios such as Ethane [37], where validation includes real hardware and up to 300 hosts are not very common. Realistic hardware simulations would also yield better results regarding the advantages and disadvantages of using OpenFlow in real networks. Using testbeds such as those described in this paper is a good way to strengthen the validation of new applications.

Finally, it is important to mention that data center virtualization is one of the active areas that has received a lot of attention in the industry. The deployment of OpenFlow by Google [123] in one of their backbone networks and active participation of the Open Networking Foundation are good examples of the interest of industry in OpenFlow. Integrating OpenFlow into such large scale real-world applications is an important future direction.

In conclusion, OpenFlow is one of the transformational technologies to affect the networking vendor community in the last decade and exhibits tremendous scope for future research and deployment.

X. Acknowledgment

We would like to thank the anonymous reviewers for their valuable suggestions. The quality of the manuscript was significantly improved based on their comments.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[2] OpenFlow Current Deployments. [Online]. Available: http://www.openflow.org/wp/current-deployments/

[3] Open Networking Foundation. [Online]. Available: https://www.opennetworking.org/

[4] A. Doria, J. H. Salim, R. Haas, H. Khosravi, W. Wang, L. Dong, R. Gopal, and J. Halpern. Forwarding and Control Element Separation (ForCES) Protocol Specification. [Online]. Available: http://tools.ietf.org/html/rfc5810

[5] J. Zander and R. Forchheimer, "The SOFTNET project: a retrospect," in *8th European Conference on Electrotechnics*, June 1988, pp. 343 –345.

[6] D. L. Tennenhouse and D. Wetherall, "Towards an active network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 26, no. 2, pp. 5–17, April 1996.

[7] J. M. Smith and S. M. Nettles, "Active networking: one view of the past, present, and future," *IEEE Trans. Syst. Man Cybern, C, Appl. Rev.*, vol. 34, no. 1, pp. 4 –18, February 2004.

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

[9] J. W. Lee, R. Francescangeli, J. Janak, S. Srinivasan, S. A. Baset, H. Schulzrinne, Z. Despotovic, and W. Kellerer, "NetServ: Active Networking 2.0," in *2011 IEEE International Conference on Communications Workshops (ICC)*, June 2011, pp. 1 –6.

[10] T. V. Lakshman, T. Nandagopal, R. Ramjee, K. Sabnani, and T. Woo, "The SoftRouter Architecture," in *Proc. ACM SIGCOMM Workshop on Hot Topics in Networking*, 2004.

[11] E. Keller and J. Rexford, "The "Platform as a service" model for networking," in *Proc. 2010 Internet Network Management Conference on Research on Enterprise Networking*, 2010.

[12] A. A. Lazar. Aurel A. Lazar home page. [Online]. Available: http://www.ee.columbia.edu/~aurel/networking.html

[13] J. Biswas, A. A. Lazar, J. F. Huard, K. S. Lim, S. Mahjoub, L. F. Pau, M. Suzuki, S. Torstensson, W. Wang, and S. Weinstein, "The IEEE P1520 Standards Initiative for Programmable Network Interfaces," in *IEEE Commun. Mag.*, vol. 36, no. 10, 1998, pp. 64–70.

[14] Internet Engineering Task Force (IETF). Proposal: Software Defined Networking Research Group (SDNRG). [Online]. Available: http://trac.tools.ietf.org/group/irtf/trac/wiki/sdnrg

[15] Internet Research Task Force (IRTF). Software Defined Networking Research Group (SDNRG) - Charter. [Online]. Available: http://www.1-4-5.net/~dmm/sdnrg/sdnrg.html

[16] Internet Engineering Task Force (IETF). Analysis of Comparisons between OpenFlow and ForCES. [Online]. Available: http://tools.ietf.org/html/draft-wang-forces-compare-openflow-forces-01

[17] OpenFlow Switch Specification, Version 1.0.0 (Wire Protocol 0x01). [Online]. Available: http://www.openflow.org/documents/openflow-spec-v1.0.0.pdf

[18] OpenFlow Switch Specification, Version 1.1.0 Implemented (Wire Protocol 0x02). [Online]. Available: http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf

[19] OpenFlow Switch Specification, Version 1.2 (Wire Protocol 0x03). [Online]. Available: https://www.opennetworking.org/images/stories/downloads/openflow/openflow-spec-v1.2.pdf

[20] OpenFlow Switch Specification, Version 1.3.0 (Wire Protocol 0x04). [Online]. Available: https://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.3.0.pdf

[21] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: towards an operating system for networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, July 2008.

[22] David Erickson. Beacon Home. [Online]. Available: https://openflow.stanford.edu/display/Beacon/Home/

[23] Z. Cai, A. L. Cox, and T. S. Eugene. Maestro: A System for Scalable OpenFlow Control. [Online]. Available: http://www.cs.rice.edu/~eugeneng/papers/TR10-11.pdf

[24] Floodlight. [Online]. Available: http://floodlight.openflowhub.org/

[25] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: a network programming language," in *Proc. 16th ACM SIGPLAN International Conference on Functional Programming*, 2011.

[26] Trema, Full-Stack OpenFlow Framework in Ruby and C. [Online]. Available: http://trema.github.com/trema/

[27] Node.flow. [Online]. Available: https://github.com/dreamerslab/node.flow

[28] Node.js. [Online]. Available: http://nodejs.org/

[29] B. Pfaff, J. Pettit, K. A. T. Koponen, M. Casado, and S. Shenker, "Extending networking into the virtualization layer," in *Proc. ACM SIGCOMM HotNets*, 2009.

[30] Open vSwitch. [Online]. Available: http://openvswitch.org/

[31] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proc. Ninth ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010.

[32] NetFPGA. NetFPGA. [Online]. Available: http://netfpga.org/foswiki/bin/view/NetFPGA/OneGig/LearnMore

[33] N. Handigol, B. Heller, V. Jeyakumar, D. Maziéres, and N. McKeown, "Where is the debugger for my software-defined network?" in *Proc. First Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012.

[34] NOX. [Online]. Available: http://www.noxrepo.org

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

[36] R. Sherwood, M. Chan, A. Covington, G. Gibb, M. Flajslik, N. Handigol, T.-Y. Huang, P. Kazemian, M. Kobayashi, J. Naous, S. Seetharaman, D. Underhill, T. Yabe, K.-K. Yap, Y. Yiakoumis, H. Zeng, G. Appenzeller, R. Johari, N. McKeown, and G. Parulkar, "Carving research slices out of your production networks with OpenFlow," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 129–130, January 2010.

[37] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: taking control of the enterprise," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 1–12, October 2007.

[38] IETF. A Path Computation Element (PCE)-Based Architecture. [Online]. Available: http://tools.ietf.org/html/rfc4655

[39] A. Giorgetti, F. Cugini, F. Paolucci, and P. Castoldi, "OpenFlow and PCE architectures in Wavelength Switched Optical Networks," in *16th International Conference on Optical Network Design and Modeling (ONDM)*, April 2012, pp. 1 –6.

[40] Energy Sciences Network (ESNet). On-Demand Secure Circuits and Advance Reservation System. [Online]. Available: http://tools.ietf.org/html/rfc4655

[41] V. Vokkarane. Progress report. [Online]. Available: http://www.cis.umassd.edu/~vvokkarane/common/reports/Y2Q1report.pdf

[42] K. Nichols, V. Jacobson, and L. Zhan. A Two-bit Differentiated Services Architecture for the Internet. [Online]. Available: http://tools.ietf.org/html/rfc2638

[43] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS flooding attack detection using NOX/OpenFlow," in *2010 IEEE 35th Conference on Local Computer Networks (LCN)*, October 2010.

[44] G. Yao, J. Bi, and P. Xiao, "Source address validation solution with OpenFlow/NOX architecture," in *19th IEEE International Conference on Network Protocols (ICNP)*, 2011.

[45] D. Staessens, S. Sharma, D. Colle, M. Pickavet, and P. Demeester, "Software defined networking: Meeting carrier grade requirements," in *18th IEEE Workshop on Local Metropolitan Area Networks (LANMAN)*, 2011.

[46] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and R. Johari, "Plug-n-Serve: Load-balancing web traffic using Open-Flow," 2009.

[47] J. Suh, H. Choi, W. Yoon, T. You, T. Kwon, and Y. Choi, "Implementation of a Content-oriented Networking Architecture (CONA): A Focus on DDoS Countermeasure," in *European NetFPGA Developers Workshop*, 2010.

[48] Y. Chu, M. Tseng, Y. Chen, Y. Chou, and Y. Chen, "A novel design for future on-demand service and security," in *12th IEEE International Conference on Communication Technology (ICCT)*, 2010.

[49] X. Liu, H. Xue, X. Feng, and Y. Dai, "Design of the multi-level security network switch system which restricts covert channel," in *IEEE 3rd International Conference on Communication Software and Networks (ICCSN)*, 2011.

[50] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "OpenFlow random host mutation: transparent moving target defense using software defined networking," in *Proc. First Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012.

[51] S. Gutz, A. Story, C. Schlesinger, and N. Foster, "Splendid isolation: a slice abstraction for software-defined networks," in *Proc. First Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012.

[52] Y. Yamasaki, Y. Miyamoto, J. Yamato, H. Goto, and H. Sone, "Flexible Access Management System for Campus VLAN Based on OpenFlow," in *IEEE/IPSJ 11th International Symposium on Applications and the Internet (SAINT)*, 2011.

[53] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *ACM SIGCOMM*, 2012.

[54] 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 on Hot Topics in Networks*, 2011.

[55] N. Katta, J. Rexford, and D. Walker, "Incremental consistent updates," in *ACM SIGCOMM HotSDN Workshop*, 2013.

[56] R. McGeer, "A safe, efficient update protocol for OpenFlow networks," in *Proc. First Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012.

[57] S. Ghorbani and M. Caesar, "Walk the line: consistent network updates with bandwidth guarantees," in *Proc. First Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012.

[58] J. Luo, J. Pettit, M. Casado, J. Lockwood, and N. McKeown, "Prototyping Fast, Simple, Secure Switches for Ethane," in *15th Annual IEEE Symposium on High-Performance Interconnects*, 2007.

[59] D. M. F. Mattos, N. C. Fernandes, V. T. da Costa, L. P. Cardoso, M. E. M. Campista, L. H. M. K. Costa, and O. Duarte, "OMNI: OpenFlow MaNagement Infrastructure," in *2011 International Conference on the Network of the Future (NOF)*, 2011.

[60] G. Gibb, H. Zeng, and N. McKeown, "Initial thoughts on custom network processing via waypoint services," in *3rd Workshop on Infrastructures for Software/Hardware Co-Design*, 2011.

[61] A. Voellmy, H. Kim, and N. Feamster, "Procera: a language for high-level reactive network control," in *Proc. First Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012.

[62] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi, "Hierarchical policies for software defined networks," in *Proc. First Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012.

[63] Y. Nakagawa, K. Hyoudou, and T. Shimizu, "A management method of IP multicast in overlay networks using OpenFlow," in *Proc. First Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012.

[64] G. Gibb, H. Zeng, and N. McKeown, "Outsourcing network functionality," in *Proc. First Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012.

[65] R. Raghavendra, J. Lobo, and K.-W. Lee, "Dynamic graph query primitives for SDN-based cloudnetwork management," in *Proc. First Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012.

[66] D. Simeonidou, R. Nejabati, and S. Azodolmolky, "Enabling the future optical Internet with OpenFlow: A paradigm shift in providing intelligent optical network services," in *2011 13th International Conference on Transparent Optical Networks (ICTON)*, 2011.

[67] S. Das, G. Parulkar, and N. McKeown, "Unifying Packet and Circuit Switched Networks," in *GLOBECOM Workshops, 2009 IEEE*, 2009.

[68] S. Das, G. Parulkar, N. McKeown, P. Singh, D. Getachew, and L. Ong, "Packet and circuit network convergence with OpenFlow," in *2010 Conference on (OFC/NFOEC) Optical Fiber Communication (OFC), collocated National Fiber Optic Engineers Conference*, 2010.

[69] S. Das, Y. Yiakoumis, G. Parulkar, N. McKeown, P. Singh, D. Getachew, and P. D. Desai, "Application-aware aggregation and traffic engineering in a converged packet-circuit network," in *Optical Fiber Communication Conference and Exposition (OFC/NFOEC) and the National Fiber Optic Engineers Conference*, 2011.

[70] S. Das, A. R. Sharafat, G. Parulkar, and N. McKeown, "MPLS with a simple OPEN control plane," in *Optical Fiber Communication Conference and Exposition (OFC/NFOEC), 2011 and the National Fiber Optic Engineers Conference*, 2011.

[71] O. El Ferkouss, S. Correia, R. Ben Ali, Y. Lemieux, M. Julien, M. Tatipamula, and O. Cherkaoui, "On the Flexibility of MPLS Applications over an OpenFlow-Enabled Network," in *2011 IEEE Global Telecommunications Conference (GLOBECOM 2011)*, 2011.

[72] J. Kempf, S. Whyte, J. Ellithorpe, P. Kazemian, M. Haitjema, N. Beheshti, S. Stuart, and H. Green, "OpenFlow MPLS and the open source label switched router," in *Proc. 23rd International Teletraffic Congress*, 2011.

[73] A. R. Sharafat, S. Das, G. Parulkar, and N. McKeown, "MPLS-TE and MPLS VPNS with OpenFlow," in *Proc. ACM SIGCOMM 2011 Conference*, 2011.

[74] N. Handigol, S. Seetharaman, M. Flajslik, A. Gember, N. McKeown, G. Parulkar, A. Akella, N. Feamster, R. Clark, A. Krishnamurthy, V. Brajkovic, and T. Anderson, "Aster*x: Load-Balancing Web Traffic over Wide-Area Networks," 2011.

[75] R. Wang, D. Butnariu, and J. Rexford, "OpenFlow-Based Server Load Balancing Gone Wild," in *Proc. 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services*, 2011.

[76] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Enabling fast failure recovery in OpenFlow networks," in *8th International Workshop on the Design of Reliable Communication Networks (DRCN)*, 2011.

[77] A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey, "VeriFlow: verifying network-wide invariants in real time," in *Proc. First Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012.

[78] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for OpenFlow networks," in *Proc. First Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012.

[79] S. Azodolmolky, R. Nejabati, E. Escalona, R. Jayakumar, N. Efstathiou, and D. Simeonidou, "Integrated OpenFlow–GMPLS control plane: an overlay model for software defined packet over optical networks," *Opt. Express*, vol. 19, no. 26, pp. B421–B428, December 2011.

[80] S. Das, G. Parulkar, and N. McKeown, "Why OpenFlow/SDN Can Succeed Where GMPLS Failed," in *European Conference and Exhibition on Optical Communication*. Optical Society of America, 2012, p. Tu.1.D.1.

[81] O. Baldonado. SDN, OpenFlow, and next-generation data center networks. [Online]. Available: http://www.eetimes.com/design/embedded/4371543/SDN--OpenFlow--and-next-generation-data-center-networks

[82] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: dynamic flow scheduling for data center networks," in *Proc. 7th USENIX conference on Networked systems design and implementation*, 2010.

[83] C. Rotsos, R. Mortier, A. Madhavapeddy, B. Singh, and A. W. Moore, "Cost, performance and flexibility in OpenFlow: Pick three," *Workshop on Software Defined Networks Co-located with the IEEE International Conference on Communications (ICC)*, 2012. [Online]. Available: http://www.cs.nott.ac.uk/~rmm/papers/pdf/iccsdn12-mirageof.pdf

[84] R. Bennesby, P. Fonseca, E. Mota, and A. Passito, "An inter-as routing component for software-defined networks," in *IEEE Network Operations and Management Symposium (NOMS)*, 2012.

[85] K.-K. Yap, M. Kobayashi, R. Sherwood, T.-Y. Huang, M. Chan, N. Handigol, and N. McKeown, "OpenRoads: empowering research in

mobile networks," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 125–126, January 2010.
[86] OpenFlow Wireless. [Online]. Available: http://www.openflow.org/wk/index.php/OpenFlow_Wireless
[87] K.-K. Yap, M. Kobayashi, D. Underhill, S. Seetharaman, P. Kazemian, and N. McKeown, "The Stanford OpenRoads Deployment," in *Proc. 4th ACM International Workshop on Experimental Evaluation and Characterization*, 2009.
[88] K.-K. Yap, S. Katti, G. Parulkar, and N. McKeown, "Delivering capacity for the mobile internet by stitching together networks," in *Proc. 2010 ACM Workshop on Wireless of the Students, by the Students, for the Students*, 2010.
[89] K.-K. Yap, R. Sherwood, M. Kobayashi, T.-Y. Huang, M. Chan, N. Handigol, N. McKeown, and G. Parulkar, "Blueprint for introducing innovation into wireless mobile networks," in *Proc. Second ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures*, 2010.
[90] T.-Y. Huang, K.-K. Yap, B. Dodson, M. S. Lam, and N. McKeown, "PhoneNet: a phone-to-phone network for group communication within an administrative domain," in *Proc. Second ACM SIGCOMM Workshop on Networking, Systems, and Applications on Mobile Handhelds*, 2010.
[91] M. Bansal, J. Mehlman, S. Katti, and P. Levis, "OpenRadio: a programmable wireless dataplane," in *Proc. First Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012.
[92] L. Suresh, J. Schulz-Zander, R. Merz, A. Feldmann, and T. Vazao, "Towards programmable enterprise WLANS with Odin," in *Proc. First Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012.
[93] L. Lu, Y. Xiao, and H. Du, "OpenFlow control for cooperating AQM scheme," in *2010 IEEE 10th International Conference on Signal Processing (ICSP)*, 2010.
[94] K.-K. Yap, T.-Y. Huang, B. Dodson, M. S. Lam, and N. McKeown, "Towards software-friendly networks," in *Proc. First ACM Asia-Pacific Workshop on Systems*, 2010.
[95] M. R. Nascimento, C. E. Rothenberg, M. R. Salvador, and M. F. Magalhães, "QuagFlow: partnering Quagga with OpenFlow," in *Proc. ACM SIGCOMM 2010 Conference*, 2010.
[96] M. R. Nascimento, C. E. Rothenberg, M. R. Salvador, C. N. A. Corrêa, S. C. de Lucena, and M. F. Magalhães, "Virtual routers as a service: the RouteFlow approach leveraging software-defined networks," in *Proc. 6th International Conference on Future Internet Technologies*, 2011.
[97] C. E. Rothenberg, M. R. Nascimento, M. R. Salvador, C. N. A. Corrêa, S. Cunha de Lucena, and R. Raszuk, "Revisiting routing control platforms with the eyes and muscles of software-defined networking," in *Proc. First Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012.
[98] H. E. Egilmez, B. Gorkemli, A. M. Tekalp, and S. Civanlar, "Scalable video streaming over OpenFlow networks: An optimization framework for QoS routing," in *18th IEEE International Conference on Image Processing (ICIP)*, 2011.
[99] OpenFlow Stanford Deployment. [Online]. Available: http://www.openflow.org/wp/stanford-deployment/
[100] Clemson OpenFlow Agregate. [Online]. Available: http://groups.geni.net/geni/wiki/GeniAggregate/ClemsonOpenFlow
[101] Georgia Tech OpenFlow Agregate. [Online]. Available: http://groups.geni.net/geni/wiki/GeniAggregate/GeorgiaTechOpenFlow
[102] Indiana OpenFlow Agregate. [Online]. Available: http://groups.geni.net/geni/wiki/GeniAggregate/IndianaOpenFlow
[103] KSU Lab OpenFlow Aggregate. [Online]. Available: http://groups.geni.net/geni/wiki/GeniAggregate/KansasStateOpenFlow
[104] Rutgers OpenFlow Agregate. [Online]. Available: http://groups.geni.net/geni/wiki/GeniAggregate/RutgersOpenFlow
[105] University of Washington OpenFlow Agregate. [Online]. Available: http://groups.geni.net/geni/wiki/GeniAggregate/WashingtonOpenFlow
[106] Winsconsin OpenFlow Agregate. [Online]. Available: http://groups.geni.net/geni/wiki/GeniAggregate/WisconsinOpenFlow
[107] GENI, Exploring networks of the future. [Online]. Available: http://www.geni.net/
[108] Internet2. [Online]. Available: www.internet2.edu
[109] National LambdaRail. [Online]. Available: http://www.nlr.net
[110] Testbed Networks: Provided by NLR. [Online]. Available: www.nlr.net/testbeds.php
[111] GENI. GENI OpenFlow Backbone Deployment at Internet2. [Online]. Available: http://groups.geni.net/geni/wiki/OFI2
[112] United States Unified Community Anchor Network. United States Unified Community Anchor Network. [Online]. Available: http://www.usucan.org/about
[113] Internet2. Nation's First 100G Open, Nationwide, Software-Defined Network Launches for Education, Research, Industry and Innovators. [Online]. Available: http://internet2.edu/news/pr/2012.10.01.nations-first-100g-national-scale-network-launches.html
[114] Internet2. Internet2 Mailing List Service. [Online]. Available: https://lists.internet2.edu/sympa/arc/i2-news/2012-07/msg00002.html
[115] Energy Science Network. Energy Science Network. [Online]. Available: www.es.net
[116] Advanced Networking Initiative (ANI). [Online]. Available: http://www.es.net/RandD/advanced-networking-initiative/
[117] IP8800 OpenFlow Networking. [Online]. Available: http://support.necam.com/pflow/legacy/ip8800/
[118] ESNet. 100G Testbed. [Online]. Available: http://www.es.net/RandD/100g-testbed/
[119] Energy Sciences Network ESNet. Proposal Process. [Online]. Available: http://www.es.net/RandD/100g-testbed/proposal-process
[120] OpenFlow Experimentation in ORBIT. [Online]. Available: http://www.orbit-lab.org/wiki/Documentation/OpenFlow
[121] Ofelia. [Online]. Available: http://www.fp7-ofelia.eu/news-and-events/press-releases/ofelia-openflow-facility-now-open-for-experiments/
[122] MRI-R2 Consortium: Development of Dynamic Network System (DYNES). [Online]. Available: http://www.internet2.edu/ion/dynes.html
[123] Open Networking Summit 2012 Program. [Online]. Available: http://opennetsummit.org/
[124] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hlzle, S. Stuart, and A. Vahdat, B4: Experience with a Globally-Deployed Software Defined WAN, in Proceedings of the ACM SIGCOMM 2013 Conference, 2013.
[125] S. Levy. Going With the Flow: Google′s Secret Switch to the Next Wave of Networking. [Online]. Available: http://www.wired.com/wiredenterprise/2012/04/going-with-the-flow-google/all/1
[126] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and performance evaluation of an OpenFlow architecture," in *23rd International Teletraffic Congress (ITC)*, 2011.
[127] A. Bianco, R. Birke, L. Giraudo, and M. Palacin, "OpenFlow Switching: Data Plane Performance," in *IEEE International Conference on Communications (ICC)*, 2010.
[128] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized?: state distribution trade-offs in software defined networks," in *Proc. First Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012.
[129] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 473–478, Sep. 2012.
[130] L. Liu, T. Tsuritani, I. Morita, H. Guo, and J. Wu, "Experimental validation and performance evaluation of OpenFlow-based wavelength path control in transparent optical networks," *Opt. Express*, vol. 19, no. 27, pp. 26 578–26 578, Sep. 2012.
[131] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," in *Proc. First Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012.
[132] J. C. Mogul and P. Congdon, "Hey, you darned counters!: get off my ASIC!" in *Proc. First Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012.
[133] G. Lu, R. Miao, Y. Xiong, and C. Guo, "Using CPU as a traffic co-processing unit in commodity switches," in *Proc. First Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012.
[134] L. Vanbever, J. Reich, T. Benson, N. Foster, and J. Rexford, "HotSwap: Correct and efficient controller upgrades for Software-Defined Networks," in *ACM SIGCOMM HotSDN Workshop*, 2013.

Adrian Lara received his B.S. and M.Sc. in Computer Science from the University of Costa Rica in 2006 and 2011. He is currently a doctoral student under the supervision of Dr. Byrav Ramamurthy at the University of Nebraska-Lincoln. His research interests include Software Defined Networking using OpenFlow, big data networks and network security. Specifically, he looks at network virtualization, multi-layer bandwidth provisioning and secure authentication using OpenFlow.

Anisha Kolasani earned her B. Tech. in Information Technology from the Jawaharlal Nehru Technological University in Hyderabad, India in 2010. In 2012, she received her M. Sc. in Computer Science and Engineering from University of Nebraska-Lincoln under the supervision of Dr. Byrav Ramamurthy. She focused on dynamic network traffic isolation through OpenFlow. She is currently working for Intel on Sort Test Technology Development.

Byrav Ramamurthy is currently a Professor in the Department of Computer Science and Engineering at the University of Nebraska-Lincoln (UNL). He is the author of the book "Design of Optical WDM Networks - LAN, MAN and WAN Architectures" and a co-author of the book "Secure Group Communications over Data Networks" published by Kluwer Academic Publishers/Springer in 2000 and 2004 respectively. He served as the Chair of the IEEE Communication Society's Optical Networking Technical Committee (ONTC) during 2009-2011. He served as the IEEE INFOCOM 2011 TPC Co-Chair. His research areas include optical and wireless networks, peer-to-peer networks for multimedia streaming, network security and telecommunications. His research work is supported by the U.S. National Science Foundation, U.S. Department of Energy, U.S. Department of Agriculture, NASA, AT&T Corporation, Agilent Tech., Ciena, HP and OPNET Inc.