A Survey on Software-Defined Networking

Wenfeng Xia, Yonggang Wen, *Senior Member, IEEE*, Chuan Heng Foh, *Senior Member, IEEE*, Dusit Niyato, *Member, IEEE*, and Haiyong Xie, *Member, IEEE*

***Abstract*—Emerging mega-trends (e.g., mobile, social, cloud, and big data) in information and communication technologies (ICT) are commanding new challenges to future Internet, for which ubiquitous accessibility, high bandwidth, and dynamic management are crucial. However, traditional approaches based on manual configuration of proprietary devices are cumbersome and error-prone, and they cannot fully utilize the capability of physical network infrastructure. Recently, software-defined networking (SDN) has been touted as one of the most promising solutions for future Internet. SDN is characterized by its two distinguished features, including decoupling the control plane from the data plane and providing programmability for network application development. As a result, SDN is positioned to provide more efficient configuration, better performance, and higher flexibility to accommodate innovative network designs. This paper surveys latest developments in this active research area of SDN. We first present a generally accepted definition for SDN with the aforementioned two characteristic features and potential benefits of SDN. We then dwell on its three-layer architecture, including an infrastructure layer, a control layer, and an application layer, and substantiate each layer with existing research efforts and its related research areas. We follow that with an overview of the de facto SDN implementation (i.e., OpenFlow). Finally, we conclude this survey paper with some suggested open research challenges.**

***Index Terms*—Software-defined networking, SDN, network virtualization, OpenFlow.**

Manuscript received May 31, 2013; revised December 7, 2013 and March 19, 2014; accepted May 15, 2014. Date of publication June 13, 2014; date of current version March 13, 2015. The work of H. Xie was supported in part by the National Natural Science Foundation of China under Grant 61073192, by the Grand Fundamental Research Program of China (973 Program) under Grant 2011CB302905, by the New Century Excellent Talents Program, and by the Fundamental Research Funds for Central Universities under Grant WK0110000014. The work of Y. Wen was supported in part by NTU under a Start-Up Grant, by the Singapore MOE under MOE Tier-1 Grant (RG 31/11), by Singapore EMA under a EIRP02 Grant, and by the Singapore National Research Foundation under its IDM Futures Funding Initiative and administered by the Interactive & Digital Media Programme Office, Media Development Authority. The work of W. Xia was supported by NSFC under Grant 61073192, by 973 Program under Grant 2011CB302905, and by Singapore EMA under an EIRP02 Grant.

W. Xia is with the School of Computer Science, University of Science and Technology of China, Hefei 230026, China, and also with the School of Computer Engineering, Nanyang Technological University, Singapore 639798 (e-mail: wenfeng_xia@ntu.edu.sg).

Y. Wen and D. Niyato are with the School of Computer Engineering, Nanyang Technological University, Singapore 639798 (e-mail: ygwen@ntu.edu.sg; dniyato@ntu.edu.sg).

C. H. Foh is with Centre for Communication Systems Research at the University of Surrey, Guildford GU2 7XH, U.K. (e-mail: c.foh@surrey.ac.uk).

H. Xie is with the Cyberspace and Data Science Laboratory, Chinese Academy of Electronics and Information Technology, Beijing 100846, China, and also with the School of Computer Science and Technology, University of Science and Technology of China, Hefei 230026, China (e-mail: haiyong.xie@acm.org).

Digital Object Identifier 10.1109/COMST.2014.2330903

I. Introduction

EMERGING mega trends in the ICT domain [1], in particular, mobile, social, cloud [2] and big data [3], [4], are urging computer networks for high bandwidth, ubiquitous accessibility, and dynamic management. First, the growing popularity of rich multimedia contents and increasing demand for big data analytics of a diverse set of data sources, are demanding higher network connection speed than ever. For example, social TV [5]–[7] and Ultra High Definition (UHD) television bring "north-south" client-server traffic tsunami to data centers, and big data analytic applications, like MapReduce [8], trigger large "east-west" server-to-server traffic in data centers to partition input data and combine output results. Second, a wide penetration of mobile devices and social networks is demanding ubiquitous communications to fulfill the social needs of general population. The number of mobile-connected devices is predicted to exceed the number of people on earth by the end of 2014, and by 2018 there will be nearly 1.4 mobile devices per capita [9]. Social networks have also experienced a dramatic growth in recent years. For instance, Facebook expanded from 1 million users in December 2004 to more than 1 billion active users in October 2012 [10]. Finally, cloud computing has added further demands on the flexibility and agility of computer networks. Specifically, one of the key characteristics for Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) is the self-managed service [2], dictating a high level of automatic configuration in the system. At the same time, with more computing and storage resources placed remotely in the cloud, efficient access to these resources via a network is becoming critical to fulfill today's computing needs. As such, computer networking has become the crucial enabling technology to move forward these emerging ICT mega trends.

In response to the aforementioned requirements for computer networks, one immediate solution would be to make additional investment in the network infrastructure to enhance the capability of existing computer networks, as practiced in reality. It is reported that the worldwide network infrastructure will accommodate nearly three networked devices and 15 gigabytes data per capita in 2016, up from over one networked device and 4 gigabytes data per capita in 2011 [11]. However, such an expansion of network infrastructure would result in an increase in complexity. First, networks are enormous in size. Even the network for a medium size organization, for example, a campus network, could be composed of hundreds or even thousands of devices [12]. Second, networks are highly heterogeneous, especially when equipment, applications, and services are provided by different manufacturers, vendors, and providers. Third, networks are very complex to manage. Human factors

1553-877X © 2014 IEEE. Translations and content mining are permitted for academic research only. Personal use is also permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

are reported to be the biggest contributor to network downtime, responsible for 50 to 80 percent of network device outages [13]. This growing complexity further demands novel approaches to future computer networks, in which the complexity can be managed.

Owing to size, heterogeneity, and complexity of current and, possibly, future computer networks, traditional approaches for configuration, optimization, and troubleshooting would become inefficient, and in some cases, insufficient. For example, Autonomous System (AS) based approaches often focus on managing a subset of networks and optimizing performance or quality of user experience for some network services, as in the case of network-oblivious P2P applications [14] and video streaming rate picking [15]. As a result, they often lead to suboptimal performance with a marginal global performance gain. Moreover, implementation of local optimizations in a single domain, without cross-domain coordination, may cause unnecessary conflicting operations with undesirable outcomes. The situation could be made worse as legacy network platforms does not have inbuilt programmability, flexibility and support to implement and test new networking ideas without interrupting ongoing services [16]. Even when new network configuration, optimization, or recovery methods are developed, implementation and testing can take years from design to standardization before a possible deployment. A protocol can take years to be standardized as an RFC [17], [18]. These observations have demanded a novel approach for future networks to support implementation, testing, and deployment of innovative ideas.

Indeed, networking research community and industry have long noticed the aforementioned problems. Previously a few new ideas have been introduced for a better design of future networks [19], including Named Data Networking (NDN) [20], programmable networks [21], "HTTP as the narrow waist" [22] and Software-Defined Networking (SDN) [23]. In particular, SDN is touted as a most promising solution. The key idea of SDN is to decouple the control plane from the data plane and allow flexible and efficient management and operation of the network via software programs. Specifically, devices (e.g., switches and routers) in the data plane perform packet forwarding, based on rules installed by controllers. Controllers in the control plane oversee the underlying network and provide a flexible and efficient platform to implement various network applications and services. Under this new paradigm, innovative solutions for specific purposes (e.g., network security, network virtualization and green networking) can be rapidly implemented in form of software and deployed in networks with real traffic. Moreover, SDN allows logical centralization of feedback control with better decisions based on a global network view and cross-layer information.

In this article, we survey the SDN literature and aim at presenting the definition of SDN and its architectural principle, providing an overview of the recent developments in SDN, and discussing about research issues and approaches for future SDN developments. The rest of this article is organized as follows. We first present the definition of SDN and its key benefits and challenges in Section II. The next three sections describe the SDN architecture with three layers in detail. Specifically, Section III focuses on the infrastructure layer, which discusses approaches to build SDN-capable switching devices and challenges of utilizing different transmission media. Section IV deals with the control layer, which introduces operations of an SDN controller and performance issues of the controller. Section V addresses issues at the application layer. This section presents some applications developed on SDN platforms, including adaptive routing, boundless mobility, network management, network security, network virtualization, green networking, and a special SDN use case with cloud computing. Section VI covers OpenFlow, which is considered as the de facto implementation of SDN. A brief conclusion with some discussion on current implementations and further developments of SDN is presented in Section VII.

II. SDN: Definition, Benefits, and Challenges

Lately SDN has become one of the most popular subjects in the ICT domain. However, being a new concept, a consensus has not yet been reached on its exact definition. In fact, a lot of different definitions [23]–[28] have surfaced over the last couple of years, each of which has its own merits. In this section, we first present a generally accepted definition of SDN, and then outline a set of key benefits and challenges of SDN, and finally introduce an SDN reference model as the anchor of this survey paper.

A. Definition of SDN

The Open Networking Foundation (ONF) [29] is a non-profit consortium dedicated to development, standardization, and commercialization of SDN. ONF has provided the most explicit and well received definition of SDN as follows:

> Software-Defined Networking (SDN) is an emerging network architecture where network control is decoupled from forwarding and is directly programmable [23].

Per this definition, SDN is defined by two characteristics, namely decoupling of control and data planes, and programmability on the control plane. Nevertheless, neither of these two signatures of SDN is totally new in network architecture, as detailed in the following.

First, several previous efforts have been made to promote network programmability. One example is the concept of active networking that attempts to control a network in a real-time manner using software. SwitchWare [30], [31] is an active networking solution, allowing packets flowing through a network to modify operations of the network dynamically. Similarly, software routing suites on conventional PC hardware, such as Click [32], XORP [33], Quagga [34], and BIRD [35], also attempt to create extensible software routers by making network devices programmable. Behavior of these network devices can be modified by loading different or modifying existing routing software.

Second, the spirit of decoupling between control and data planes has been proliferated during the last decade. Caesar *et al.* first presented a Routing Control Platform (RCP) in 2004 [36], in which Border Gateway Protocol (BGP) inter-domain routing is replaced by centralized routing control to reduce complexity of fully distributed path computation. In the same

Ex. 1011
Juniper Networks, Inc. / Page 2 of 25

year, IETF released the Forwarding and Control Element Separation (ForCES) framework, which separates control and packet forwarding elements in a ForCES Network [37]–[40]. In 2005, Greenberg *et al.* proposed a 4D approach [41]–[43], introducing a clean slate design of the entire network architecture with four planes. These planes are "decision", "dissemination", "discovery", and "data", respectively, which are organized from top to bottom. In 2006, the Path Computation Element (PCE) architecture was presented to compute label switched paths separately from actual packet forwarding in MPLS and GMPLS networks [44]. In 2007, Casado *et al.* presented Ethane, where simple flow-based Ethernet switches are supplemented with a centralized controller to manage admittance and routing of flows [45]–[48]. In this latest development, the principle of data-control plane separation has been explicitly stated. Commercial networking devices have also adopted the idea of data-control plane separation. For example, in the Cisco ASR 1000 series routers and Nexus 7000 series switches, the control plane is decoupled from the data plane and modularized, allowing coexistence of an active control plane instance and a standby one for high fault tolerance and transparent software upgrade.

In the context of SDN, its uniqueness resides on the fact that it provides programmability through decoupling of control and data planes. Specifically, SDN offers simple programmable network devices rather than making networking devices more complex as in the case of active networking. Moreover, SDN proposes separation of control and data planes in the network architectural design. With this design, network control can be done separately on the control plane without affecting data flows. As such, network intelligence can be taken out of switching devices and placed on controllers. At the same time, switching devices can now be externally controlled by software without onboard intelligence. The decoupling of control plane from data plane offers not only a simpler programmable environment but also a greater freedom for external software to define the behavior of a network.

B. SDN Benefits

SDN, with its inherent decoupling of control plane from data plane, offers a greater control of a network through programming. This combined feature would bring potential benefits of enhanced configuration, improved performance, and encouraged innovation in network architecture and operations, as summarized in Table I. For example, the control embraced by SDN may include not only packet forwarding at a switching level but also link tuning at a data link level, breaking the barrier of layering. Moreover, with an ability to acquire instantaneous network status, SDN permits a real-time centralized control of a network based on both instantaneous network status and user defined policies. This further leads to benefits in optimizing network configurations and improving network performance. The potential benefit of SDN is further evidenced by the fact that SDN offers a convenient platform for experimentations of new techniques and encourages new network designs, attributed to its network programmability and the ability to define isolated virtual networks via the control plane. In this subsection, we dwell on these aforementioned benefits of SDN.

1) Enhancing Configuration: In network management, configuration is one of the most important functions. Specifically, when new equipment is added into an existing network, proper configurations are required to achieve coherent network operation as a whole. However, owing to the heterogeneity among network device manufacturers and configuration interfaces, current network configuration typically involves a certain level of manual processing. This manual configuration procedure is tedious and error prone. At the same time, significant effort is also required to troubleshoot a network with configuration errors. It is generally accepted that, with the current network design, automatic and dynamic reconfiguration of a network remains a big challenge. SDN will help to remedy such a situation in network management. In SDN, unification of the control plane over all kinds of network devices [50], including switches, routers, Network Address Translators (NATs), firewalls, and load balancers, renders it possible to configure network devices from a single point, automatically via software controlling. As such, an entire network can be programmatically configured and dynamically optimized based on network status.

2) Improving Performance: In network operations, one of the key objectives is to maximize utilization of the invested network infrastructure. However, owing to coexistence of various technologies and stakeholders in a single network, optimizing performance of the network as a whole has been considered difficult. Current approaches often focus on optimizing performance of a subset of networks or the quality of user experience for some network services. Obviously, these approaches, based on local information without cross-layer consideration, could lead to suboptimal performance, if not conflicting network operations. The introduction of SDN offers an opportunity to improve network performance globally. Specifically, SDN allows for a centralized control with a global network view and a feedback control with information exchanged between different layers in the network architecture. As such, many challenging performance optimization problems would become manageable with properly designed centralized algorithms. It follows that new solutions for classical problems, such as data traffic scheduling [51], end-to-end congestion control [52], load balanced packet routing [53], energy efficient operation [54], and Quality of Service (QoS) support [55], [56], can be developed and easily deployed to verify their effectiveness in improving network performance.

3) Encouraging Innovation: In the presence of continuing evolution of network applications, future network should encourage innovation rather than attempt to precisely predict and perfectly meet requirements of future applications [57]. Unfortunately, any new idea or design immediately faces challenges in implementation, experimentation, and deployment into existing networks. The main hurdle arises from widely used proprietary hardware in conventional network components, preventing modification for experimentation. Besides, even when experimentations are possible, they are often conducted in a separate simplified testbed. These experimentations do not give sufficient confidence for industrial adaptation of these new ideas or network designs. The idea behind community efforts like PlanetLab [58] and GENI [59] to enabled large scale experimentations, cannot solve the problem completely.

Ex. 1011
Juniper Networks, Inc. / Page 3 of 25

TABLE I
COMPARISONS BETWEEN SDN AND CONVENTIONAL NETWORKING

	SDN	Conventional Networking
Features	decoupled data and control plane, and programmability	a new protocol per problem, complex network control [49]
Configuration	automated configuration with centralized validation	error prone manual configuration
Performance	dynamic global control with cross layer information	limited information, and relatively static configuration
Innovation	easy software implementation for new ideas, sufficient test environment with isolation, and quick deployment using software upgrade	difficult hardware implementation for new ideas, limited testing environment, long standardization process

In comparison, SDN encourages innovation by providing a programmable network platform to implement [60], experiment [61], and deploy new ideas, new applications, and new revenue earning services conveniently and flexibly. High configurability of SDN offers clear separation among virtual networks permitting experimentation on a real environment. Progressive deployment of new ideas can be performed through a seamless transition from an experimental phase to an operational phase.

C. SDN Challenges

Given the promises of enhanced configuration, improved performance, and encouraged innovation, SDN is still in its infancy. Many fundamental issues still remain not fully solved, among which standardization and adoption are the most urgent ones.

Though the ONF definition of SDN is most received one, OpenFlow sponsored by ONF is by no means the only SDN standard and by no means a mature solution. An open-source OpenFlow driver is still absent for SDN controller development, a standard north-bound API or a high level programming language is still missing for SDN application development. A healthy ecosystem combining network device vendors, SDN application developers, and network device consumers, has yet to appear.

SDN offers a platform for innovative networking techniques, however the shift from traditional networking to SDN can be disruptive and painful. Common concerns include SDN interoperability with legacy network devices, performance and privacy concerns of centralized control, and lack of experts for technical support. Existing deployments of SDN are often limited to small testbed for research prototypes. Prototypes for research purpose remain premature to offer confidence for real world deployment.

D. SDN Reference Model

ONF has also suggested a reference model for SDN, as illustrated in Fig. 1. This model consists of three layers, namely an infrastructure layer, a control layer, and an application layer, stacking over each other.

The infrastructure layer consists of switching devices (e.g., switches, routers, etc.) in the data plane. Functions of these switching devices are mostly two-fold. First, they are responsible for collecting network status, storing them temporally in local devices and sending them to controllers. The network status may include information such as network topology, traffic statistics, and network usages. Second, they are responsible for processing packets based on rules provided by a controller.

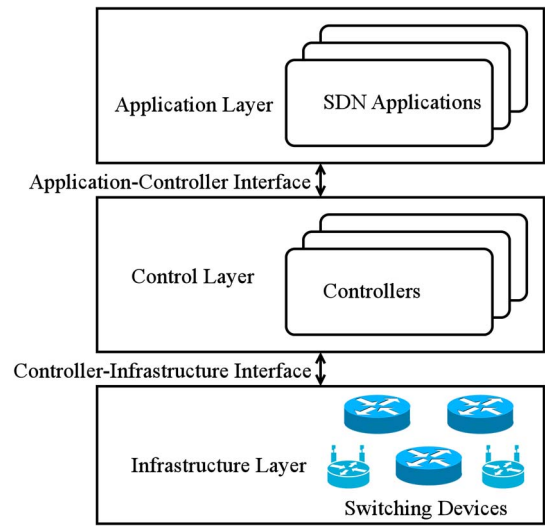

Fig. 1. SDN Reference Model: a three-layer model, ranging from an infrastructure layer to a control layer to an application layer, in a bottom-up manner.

The control layer bridges the application layer and the infrastructure layer, via its two interfaces. For downward interacting with the infrastructure layer (i.e., the south-bound interface), it specifies functions for controllers to access functions provided by switching devices. The functions may include reporting network status and importing packet forwarding rules. For upward interacting with the application layer (i.e., the north-bound interface), it provides service access points in various forms, for example, an Application Programming Interface (API). SDN applications can access network status information reported from switching devices through this API, make system tuning decisions based on this information, and carry out these decisions by setting packet forwarding rules to switching devices using this API. Since multiple controllers will exist for a large administrative network domain, an "east-west" communication interface among the controllers will also be needed for the controllers to share network information and coordinate their decision-making processes [62], [63].

The application layer contains SDN applications designed to fulfill user requirements. Through the programmable platform provided by the control layer, SDN applications are able to access and control switching devices at the infrastructure layer. Example of SDN applications could include dynamic access control, seamless mobility and migration, server load balancing, and network virtualization.

In this survey, we adopt this reference model as a thread to organize existing research efforts in SDN into three sections.

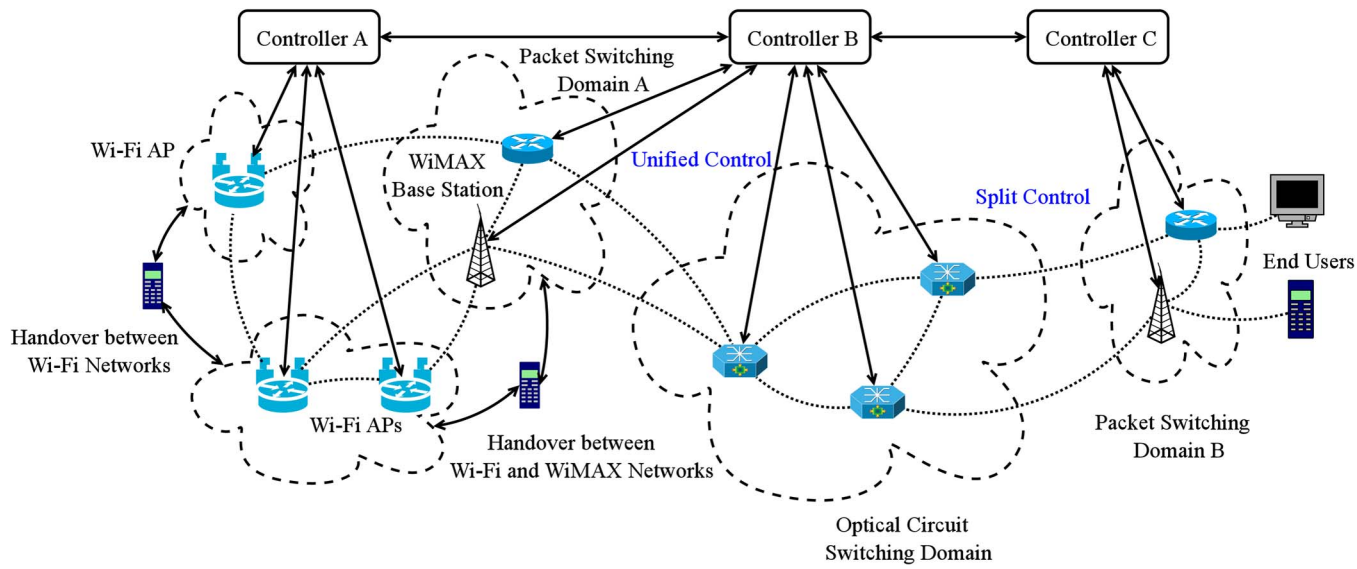

Fig. 2. SDN Infrastructure Architecture: switching devices are connected to formulate a mesh topology via various transmission media, including copper wires, wireless radio and optical fibre.

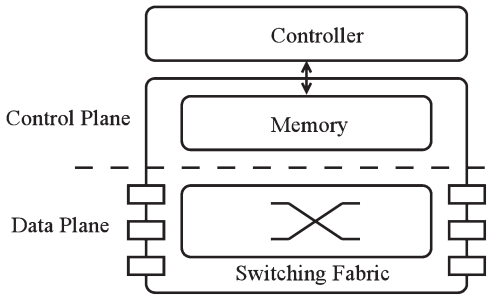

Fig. 3. Switching Device Model in SDN: a two-layer logical model consisting of a processor for data forwarding and onboard memory for control information.

III. Infrastructure Layer

At the lowest layer in the SDN reference model, the infrastructure layer consists of switching devices (e.g., switches, routers, etc.), which are interconnected to formulate a single network. The connections among switching devices are through different transmission media, including copper wires, wireless radio, and also optical fibers. In Fig. 2, we illustrate an SDN-enabled reference network. In particular, the main research concerns associated with the infrastructure layer include both efficient operations of switching devices and utilization of transmission media, as detailed in the next two subsections.

A. Switching Devices

In Fig. 3, we illustrate the architectural design of an SDN switching device, consisting of two logical components for the data plane and the control plane. In the data plane, the switching device, in particular, through its processor, performs packet forwarding, based on the forwarding rules imposed by the control layer. Examples of network processors include XLP processor family (MIPS64 architecture) from Broadcom, XScale processor (ARM architecture) from Intel, NP-x NPUs from EZChip, PowerQUICC Communications Processors (Power architecture) from freescale, NFP series processors (ARM architecture) from Netronome, Xelerated HX family from Marvell, OCTEON series processors (MIPS64 architecture) form Cavium and general purpose CPUs from Intel and AMD. In the control plane, the switching device communicates with controllers at the control layer to receive rules, including packet forwarding rules at a switching level and link tuning rules at a data-link level, and stores the rules in its local memory. Examples of memory include TCAM and SRAM.

This new architectural principle lends SDN competitive advantages. Unlike conventional switching devices that also run routing protocols to decide how to forward packets, routing decision makings are stripped from switching devices in SDN. As a result, the switching devices are simply responsible for gathering and reporting network status as well as processing packets based on imposed forwarding rules. It follows that the SDN switching devices are simpler and will be easier to manufacture. The reduced complexity in turn leads to a low cost solution.

This new architecture, however, requires new hardware design for SDN-enabled switching devices. In this subsection, we describe recent research progresses in switching device hardware design, focusing on both the control plane and the data plane. We will also classify the most popular switching device platforms, and discuss testing and evaluation of these switching devices.

1) Control Plane: In the control plane of SDN switching devices, one of the main design challenges resides on the efficient use of onboard memory. Fundamentally, memory usage in a switching device depends on the network scale. Specifically, switching devices in a larger scale network would need a larger memory space; otherwise, they may need constant hardware upgrades to avoid memory exhaustion. In case of insufficient memory space, packets would be dropped or directed to controllers for further decisions on how to process them, resulting in a degraded network performance [64].

Memory management techniques in traditional switch design can be extended to optimize the SDN switch design for rule storage in order to reduce memory usage and use the limited

TABLE II
SDN SWITCHING DEVICES

Category	Implementation on General PC Hardware	Implementation on Open Network Hardware	Implementation on Vendor's Switch
Platform or Representative	OpenWrt Wireless Freedom; SyClick! Modular Router	NetFPGA; AdvancedTCA™	Indigo
Processing Speed	Low	Middle, 1Gbps or 10Gbps for NetFPGA	High, usually 1Gbps and above
Port Density	Low, limited to the amount of NICs	Middle, 4 ports for NetFPGA	High, usually 48 ports and more
Flexibility	High, all features are software defined	Middle	Low

memory efficiently. Specifically, to deal with massive routing records, conventional routers use techniques such as route aggregation or summarization and proper cache replacement policy [65], [66]. Route aggregation or summarization can reduce the memory usage by aggregating several routing records with a common routing prefix to a single new routing record with the common prefix. A proper cache replacement policy can improve packet forwarding rule hit rate of all packets, thus the limited memory can be used efficiently. These techniques can be adopted to improve SDN switching device design.

Another major principle in improving SDN switching device design is judicious combination of different storage technologies to achieve desired memory size, processing speed, and flexibility with reasonable price and complexity. Different storage hardware exhibits different characteristics [67], [68]. For example, Static Random Access Memory (SRAM) can be easily scaled up and is more flexible; Ternary Content Addressable Memory (TCAM) offers a faster searching speed for packet classification. SRAM and TCAM can be used jointly to balance the trade-off between packet classification performance and flexibility.

2) Data Plane: The main function of an SDN switching device's data plane is packet forwarding. Specifically, upon receiving of a packet, the switching device first identifies the forwarding rule that matches with the packet and then forwards the packet to next hop accordingly. Compared to packet forwarding in legacy networks based on IP or MAC addresses, SDN packet forwarding can also be based on other parameters, for example, TCP or UDP port, Virtual Local Area Network (VLAN) tag, and ingress switch port. Using a long vector for forwarding decision would undoubtedly increases processing complexity in computation, resulting a fundamental trade-off between cost and efficiency in SDN packet processing. Several solutions conceived for fast data path packet processing have been proposed, among which two are explained as follows.

First, in PC-based switching devices, using software for packet processing may result in inefficient performance. As an improvement, Tanyingyong *et al.* suggest using hardware classification to increase processing throughput [69], [70]. In this design, incoming packets are directed to an onboard Network Interface Controller (NIC) for hardware classification based on flow signatures. As a result, a CPU is exempted from the lookup process.

Second, the different nature for the "elephant" and "mice" flows can be exploited. Contrary to "elephant" flows, "mice" flows are numerous, but each of them has few packets. Web page retrieving flows are examples of "mice" flows. In fact, "mice" flows contribute primary to the frequent events to be handled by switching devices, but they have little influence on the overall network performance. Taking this observation, Lu *et al.* suggest offloading "elephant" flows to an ASIC while leaving "mice" flows to be handled by a CPU with relatively slower processing speed [71].

3) Classification and Evaluation of Switching Devices: Currently, SDN switching devices can be classified to three major categories, according to their hardware specifications, as show in Table II, including:

- *Implementation on general PC hardware*: SDN switches can be implemented as software running on a host operating system (OS), usually Linux. The host OS can run on standard x86/x64 PC hardware or other compatible hardware. Examples of software switches include Pantou [72] and OpenFlowClick [73]. Pantou is based on OpenWRT, which is a Linux distribution for embedded devices, especially routers. OpenFlowClick is based on Click [32], which is implemented on general-purpose PC hardware as an extension to the Linux kernel. Software switches provide a port density limited to the number of NICs onboard and relatively slow packet processing speed using software processing. A significant advantage of software implemented SDN switches is that they can provide virtual switching for VMs in the popular paradigm of server virtualization and cloud computing. Software implemented SDN switches like Open vSwitch [74], [75] can provide network visibility and control in a straightforward way. Traffic among VMs hosted by the same physical server is kept on the server, while in hairpin switching, all traffic is redirected to the physical switch connected with the server then bounces off.
- *Implementation on open network hardware*: Open network hardware platform offers a vendor independent and programmable platform to build networks for research and classroom experiments. The industry is also paying more attention to open network hardware platforms. Examples of open network hardware platform based SDN switching device implementations include NetFPGA [76] based implementations such as SwitchBlade [77] and ServerSwitch [78], and Advanced Telecommunications Computing Architecture (ATCA) based implementations such as ORAN [79]. Open network hardware platform based switches are the most commonly used to build SDN prototypes in laboratories [80], [81], since they are more flexible than vendor's switches and provide higher throughput than that of software implemented ones.

Ex. 1011
Juniper Networks, Inc. / Page 6 of 25

- *Implementation on vendor's switch*: Nowadays, more and more networking hardware vendors are releasing their SDN strategies and solutions, along with a vast variety of SDN-enabled switches, including NEC PF5240, IBM G8264, and Pica8 3920. There are also projects, for example, Indigo [82], to enable SDN features using firmware upgrades on vendor's switches that do not support SDN features originally.

Performance benchmark plays an important role in further innovation in switching devices. For example, testing and evaluation of switching devices can ensure proper operations and facilitate to performance improvement. In this aspect, empirical comparison conducted by Bianco *et al.* shows higher forwarding performance and better fairness of Linux software SDN switching than that of Linux software layer-2 Ethernet switching and layer-3 IP routing [83]. This result gives confidence on performance of SDN switching over conventional non-SDN switching. To facilitate performance benchmark, Rotsos *et al.* present the OpenFLow Operations Per Second (OFLOPS) framework that supports multiple packet generation, capturing, and timestamping mechanisms with different precisions and impacts [84]. OFLOPS measures performance metrics of control plane operations like rule insertion delay and traffic statistic query delay. OFLOPS permits detailed performance measurement control plane operations for both software implementations and hardware implementations of SDN switching devices and will be useful to evaluate performance of SDN switching devices.

B. Transmission Media

As illustrated in Fig. 2, SDN should embrace all possible transmission media, including wired, wireless and optical environments, in order to fulfill a ubiquitous coverage. At the same time, different transmission media have their unique characteristics, which in turn often result in specific configuration and management technologies. As such, SDN should integrate with these technologies in wireless and optical networks. For example, Software-Defined Radio (SDR) [85] supports cost-effective evolution of radio devices and Generalized Multi-Protocol Label Switching (GMPLS) [86] is the de facto control plane for wavelength switched optical networks. Integrating these technologies gives SDN controllers a great opportunity to have a widespread control over all the network behaviors, including packet forwarding, wireless mode or channel, and optical wavelength. It follows that SDN can gain more appropriate control of the network infrastructure and achieve more efficient infrastructure resource utilization.

1) Wireless Radio: Many advanced wireless transmission technologies have been developed to maximize spectrum utilization in wireless networks. Among them, Software-Defined Radio (SDR) permits control of wireless transmission strategy via software [85]. Given its similar nature, the SDR technology can be easily integrated with SDN. For example, Bansal *et al.* point out that many computationally intensive processing blocks are common at the physical layer of all modern wireless systems, differing only in configurations [87]. One instance is that almost all wireless systems use Fast Fourier Transform (FFT) with probably different FFT-lengths. This observation motivates them to propose OpenRadio to decouple wireless protocol definition from the hardware and use a declarative interface to program wireless protocols. In another study, Murty *et al.* present Dyson where a wireless NIC driver is modified to support statistic collection and Dyson command API [88]. Clients and Access Points (APs) passively report measurement information, including total number of packets, total packet size, and total airtime utilization, to a central controller. The central controller can manage link association, channel selection, transmission rate and traffic shaping for both clients and APs through the API based on current and historical measurement information.

Essentially, both OpenRadio and Dyson allow physical layer functions to be controlled by software. In this sense, they are SDR systems. Moreover, the Software Communication Architecture (SCA) [89], [90] is designed by the US Military's Joint Tactical Radio System (JTRS) and has been sponsored by the main SDR proponent, namely Wireless Innovation Forum [91]. Software reconfigurability of an SDR system like SCA gives SDN controllers an interface to control the SDR system. In fact, SDR can benefit from the central control and global view of SDN, as used in Dyson [88]. In return, SDN controllers can control SDR systems and have a widespread and precise control of all the network devices.

2) Optical Fibers: Optical fibers offer a high capacity with low power consumption. They are widely used in backbones for aggregated traffic. The idea of software reconfiguration used in wireless networks can also be adopted in optical networks by employing Reconfigurable Optical Add/Drop Multiplexers (ROADMs) [92]. Integrating these technologies into the SDN control plane helps achieve more precise and efficient control of the data plane [93].

Unified approaches with a single SDN control plane over both packet switching domains and circuit switching domains are firstly considered, as shown in Fig. 2 where "Controller B" manages an optical circuit switching domain and the "Packet Switching Domain A". Das *et al.* suggest extending parameters used in forwarding rule matching from layer 2, 3, and 4 headers of a packet to include layer 1 switching technologies, such as time-slot, wavelength, and fiber switching. Thus it provides a single unified control plane over packet and optical networks [94]–[97]. The proposed scheme provides a simplified control model, but needs to upgrade optical circuit switching devices to support this extension. Liu *et al.* propose to use a virtual switch on each optical switching node to obtain unified control plane [98]–[103]. Each physical interface of an optical switching node is mapped to a virtual interface correspondingly. Messages between the controller and the virtual switch are converted to commands acceptable by the optical switching devices. A similar idea is applied to reuse and integrate legacy equipment with SDN switching devices. During deployment, there will be an extra layer to bridge controllers and legacy switches [104]. Though these approaches can reuse legacy network equipment, they cause extra communication latency by using message proxying.

Given its long haul nature in an optical network, an end-to-end data path from source to destination can be controlled by

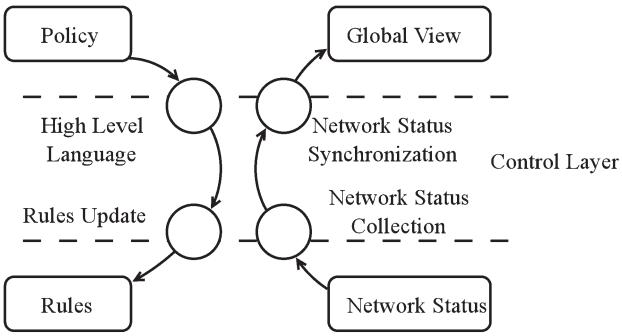

Fig. 4. Controller Logical Design: a high level language for SDN applications to define their network operation policies; a rule update process to install rules generated from those policies; a network status collection process to gather network infrastructure information; a network status synchronization process to build a global network view using network status collected by each individual controller.

multiple stakeholders, each of which controls parts of the data path. In this case, it may not be practical to have a single control plane along the data path. Split-control approaches, as shown in Fig. 2 where "Controller B" manages an optical circuit switching domain and "Controller C" manages the "Packet Switching Domain B", may be a natural choice and can reuse advanced techniques in optical circuit switching. For example, a GMPLS control plane that manages the optical network. Along this line of logic, Casado *et al.* suggest decoupling transport edge and core [105], [106]. They present "Fabric" where edge controllers handle operator requirements; ingress edge switches along with their controllers handle host requirements; switches in the "Fabric" just forward packets. Similarly, Azodolmolky *et al.* propose to integrate the GMPLS control plane to an SDN controller to unify the packet switching and optical circuit switching domains [107]–[109]. The GMPLS control plane manages the core optical domain and interacts with an extended SDN controller that manages the packet switching domain.

IV. Control Layer

As illustrated in Fig. 1, the control layer bridges the application layer and the infrastructure layer. In this section, we first present a logical design for SDN control layer, which consists of four main components, namely a high level language, a rule update process, a network status collection process and a network status synchronization process, as illustrated in Fig. 4. Following that, we focus on two critical issues at the control layer, namely policy and rule validation, and performance challenges and possible solutions for the control layer.

A. Controller Design

The controller is the most important component in the SDN architecture, where the complexity resides. In this subsection, we adopt a "divide-and-conquer" strategy to present a logical controller architecture. Our proposed architecture, as depicted in Fig. 4, is based on two principles, including:

- Objects: the SDN controller deals with two types of objects. One is used for network controlling, including policies imposed by the application layer and packet forwarding rules for the infrastructure. The other is related to network monitoring, in the format of local and global network status. It follows that the logical architecture has two counter-directional information flows, as illustrated in Fig. 4. In the downward flow, the controller translates the application policy into packet forwarding rules, in respect to network status. The main concern of this process is to ensure validity and consistency of the forwarding rules. In the upward flow, the controller synchronizes network status collected from the infrastructure for networking decision making.
- Interfaces: the SDN controller has two interfaces. The south-bound interface, which is marked as the controller-infrastructure interface in Fig. 1, deals with transactions with the infrastructure layer, i.e., collecting network status and updates packet forwarding rules to switching devices at the infrastructure layer accordingly. The north-bound interface, which is marked as the application-controller interface in Fig. 1, handles transactions with the application layer, i.e., receiving policies described in high level languages from SDN applications and providing a synchronized global view.

Leveraging these architectural principles, the logical design for SDN controllers can be decoupled into four building components, namely, a high-level language, a rule update process, a network status collection process, and a network status synchronization process. In this subsection, we adopt this structure to explain existing research efforts in controller design into the aforementioned groups.

1) High Level Language: One of the key controller functions is to translate application requirements into packet forwarding rules. This function dictates a communication protocol (e.g., a programming language) between the application layer and the control layer. One straightforward approach is to adopt some common configuration languages, for example, the Command Line Interface (CLI) for Cisco Internetwork Operating System (IOS). However, these common configuration languages only offer primitive abstractions derived from capabilities of underlying hardware. Given that they are designed for hardware configurations, they are typically inadequate to accommodate dynamic and stateful network status. Moreover, they are error-prone and demanding extra effort in the process of programming. Therefore, it is imperative to provide a high level language for SDN applications to interface with controllers. Such a high-level language should embrace an expressive and comprehensive syntax for SDN applications to easily describe their requirements and network management strategies.

Strategies to design qualified high-level languages for SDN controllers take at least two formats. One strategy is to utilize existing mature high-level languages, such as, C++, Java and Python, for application development. This approach normally provides a Software Development Kit (SDK) with libraries for desirable features, such as security, guaranteed bandwidth and on-demand provisioning. One example in the category is the One Platform Kit (onePK) from Cisco [110]. The other strategy adopts a clean-state design to propose new high-level languages with special features to achieve efficient network behavior

control for SDN. Compared to the first approach, few published work or released implements of SDKs can be found currently, nor does a dominating new high level language exist. In the following paragraphs, we review several high-level languages for SDN, including the earlier work on this domain called Flow-based Management Language (FML) [111], a well-developed programming language for SDN called Frenetic [112], [113], and another high level language called Nettle [114]–[117].

Flow-based Management Language (FML) [111], previously known as Flow-based Security Language (FSL) [118], is a language for convenient and expressive description of network connectivity policies in SDN. FML offers fine granular operations on unidirectional network flows and supports expressive constraints in allowing/denying traffic and limiting bandwidth, latency, and jitter. Order irrelevance of FML makes it straightforward to combine a set of independent policies. Moreover, a long-policy code can easily be understood without knowledge of the context. Ferguson *et al.* present PANE that extends FML with queries and hints for network status and a time dimension [55]. Such an extension can define how long a request for guaranteed bandwidth could be fulfilled. PANE improves policy expressiveness with precise knowledge and predictions of network status. PANE also introduces hierarchical flow tables to realize hierarchical policies for better policy management [119].

Frenetic [112], [113] is proposed to eliminate complicated asynchronous and event-driven interactions between SDN applications and switching devices. In particular, Frenetic introduces an SQL-like declarative network query language for classifying and aggregating network traffic statistics. Moreover, a functional reactive network policy management library is introduced to handle details of installing and uninstalling switch level rules. The query language includes a rich pattern algebra, for example, set operations, which provides a convenient way to describe sets of packets. After the appearance of Frenetic, several functional and performance enhancements are introduced. Monsanto *et al.* introduce the use of wild card rules and proactive generation. As a result, more packets can be matched by wild card rules than that of exact match rules, and the packets can be processed on switching devices without requests to controllers [120]. Gutz *et al.* add syntaxes to describe isolated network slices and enable network virtualization using Frenetic [121]. Pyretic [122] introduces sequential composition that allows one rule to act on packets already processed by another rule, and topology abstraction that maps between physical switches and virtual switches.

Voellmy *et al.* present Nettle to improve responsiveness to dynamic network changes. Nettle uses reactive languages to describe policies by combining functional reactive programming (FRP) and domain-specific languages (DSLs) [114]–[117]. FRP enables programming for real-time interactive network control in a declarative manner. A piece of Nettle program takes network events as input, for example, detection of a new user. The program then outputs rule updates for specific usage, for example, user authentication. A single one-size-fits-all language may not be possible, given the diversity of network management concerns. DSLs enable Nettle to provide an extensible family of DSLs, each of which is designed for a specific issue, for example, user authentication and traffic engineering. Nettle uses Haskell as the host language because of its remarkable flexibility in supporting embedded DSLs. Later, Voellmy *et al.* present "Maple" to simplify SDN programming by allowing a programmer to use a standard programming language to design an arbitrary centralized algorithm for every packet entering the network, hence removing low-level details [123]. Maple consists of two key components, namely an "optimizer" and a "scheduler". The optimizer uses a "trace tree" data structure to record the invocation of the programmer-supplied algorithm on a specific packet, then generalizes rules in the flow table of individual switches. A trace tree captures the reusability of previous computations and hence substantially reduces the number of invocations of the same algorithm. The optimizer uses various techniques to improve the efficiency, including trace tree augmentation, trace tree compression, and rule priority minimization. Flow table missed packets have to be processed at the controller, and the scheduler applies "switch-level parallelism" on multi-core servers, binding the controller's thread, memory, and event processing loop to a particular "client" switch.

2) Rules Update: An SDN controller is also responsible for generating packet forwarding rules describing the policies and installing them into appropriate switching devices for operation. At the same time, forwarding rules in switching devices need to be updated because of configuration changes and dynamic control, such as directing traffic from one replica to another for dynamical load balancing [124], Virtual Machine (VM) migration [125], and network recovery after unexpected failure. In the presence of network dynamics, consistency is a basic feature that rule update should preserve to ensure proper network operations and preferred network properties, such as, loop free, no black hole, and security.

Rule consistency can be established in different flavors. In literature, two alternative consistency definitions are discussed, including:

- *Strict Consistency*: it ensures that either the original rule set or the updated rule set is used. Strict consistency could be enforced in a per-packet level, where each packet is processed, or in a per-flow level, where all packets of a flow are processed by either the original rule set or the updated rule set.
- *Eventual Consistency*: it ensures that the later packets use the updated rule set eventually after the update procedure finishes and allows the earlier packets of the same flow to use the original rule set before or during the update procedure.

In the former category, Reitblatt *et al.* propose a strict consistency implementation that combines versioning with rule timeouts [126], [127]. The idea is to stamp each packet with a version number at its ingress switch indicating which rule set should be applied. Then, the packet will be processed depending on the version number. The later packets will be stamped to take the updated rule set. Thus, no more packets will take the original rule set after a time long enough. Then, the original rule set will be removed. Nevertheless, both the original and updated rule sets are kept in switching devices before the original rule set expires and is removed.

In the latter category, McGeer *et al.* implement eventual consistency to conserve switch memory space by ensuring that only a single set of rules is presented in a switching device at any time [128]. When a new policy is about to be implemented, all corresponding switching devices are first informed to direct affected packets to a controller. The controller then generates new packet forwarding rules based on the policy and replaces rules in corresponding switching devices with these new rules. When the replacements are completed, affected packets buffered in the controller earlier are released back into switching devices for processing. A flow will take the original rule set before the flow is directed to the controller and the updated rule set after the flow is released back to switching devices. This is an undesired condition that may occur in eventual consistency. Nonetheless, eventual consistency could be a choice when memory space is scarce.

3) Network Status Collection: In the upward flow, controllers collect network status to build a global view of an entire network and provide the application layer with necessary information, for example, network topology graph [129], for network operation decisions. One main network status is traffic statistics, such as, duration time, packet number, data size, and bandwidth share of a flow. Typically network status collection works in the following way. Each switching device collects and stores local traffic statistics within its own storage. These local traffic statistics may be retrieved by controllers (i.e., a "pull" mode), or proactively reported to controllers (i.e., a "push" mode). Different mode and strategy have different characteristics in measurement overhead and accuracy. In this manner, a key research objective is to find a "sweet spot" (i.e., optimal point) with adequate accuracy yet maintaining low measurement overhead.

A useful and commonly used form for network status data is Traffic Matrix (TM). TM reflects volume of traffic that flows between all possible pairs of sources and destinations in a network [130]. Tootoonchiane *et al.* present OpenTM that reads byte and packet counters maintained by switching devices (i.e., a "pull" mode) for active flows to estimate the TM [131]. Querying the last switching device in a path from source to destination results in the most accurate measurement of traffic from source to destination, since typically the receiver will have the most complete picture of the path. However, this approach may result in overloading the last switching device in the path. OpenTM uses selective query strategies with various query distributions along the path to balance the trade-off between measurement accuracy and query load on individual switching devices. Based on the observation that the amount of data observed in hosts' TCP buffers rises much quicker than that observed at the network layer for a flow. Curtis *et al.* propose Mahout to detect "elephant" flows at the hosts. Type of Service (ToS) byte is marked by the host to notify "elephant" flows to the controller [132].

In addition to the aforementioned strategy, ignoring "mice" flows in measurement can be another strategy to relieve overhead while causing little side effects. For example, Jose *et al.* suggest using a hierarchical heavy hitters algorithm to identify "elephant" flows and ignore "mice" flows [133]. Switching devices match packets against a small collection of rules and update traffic counters for the highest-priority match. Thus only traffic statistics of the "elephant" flows will be updated and collected.

Streaming algorithms show low memory usage, bounded estimation error, and high processing speed accommodating high arrive speed of input data in processing data steams [134], [135]. They are naturally suitable for network traffic flow monitoring [136]–[138]. Yu *et al.* present OpenSketch, which is a network traffic measurement architecture for SDN leverage the advantages of streaming algorithms [139]. The data plane of OpenSketch consists of a three stage pipeline, namely hashing, filtering, and counting. First, fields of interest of an incoming packet are hashed to save memory space, the hash code is then filtered to decide whether the packet should be counted, last corresponding counters are updated. OpenSketch also provides a measurement library in the control plane that automatically configures the pipeline and allocates resources for various measurement tasks, including unique source and destination count, heavy hitters, flow size distribution. Their prototype implementation of OpenSketch on NetFPGA shows no effect on data plane throughput and about 200 nanoseconds processing delay for 5 counter updates per packet.

4) Network Status Synchronization: Delegating control to a centralized controller can cause performance bottleneck at the centralized controller. A common solution to overcome this bottleneck is deploying multiple controllers acting peer, backup, or replicate controllers [140]. Maintaining a consistent global view among all controllers is essential to ensure proper network operations. Inconsistent or stale states may result in the application layer making incorrect decisions, which then leads to inappropriate or suboptimal operations of the network [141].

Publish/subscribe systems are widely used to achieve a synchronized global network view. For example, Tootoonchian *et al.* introduce HyperFlow that allows sharing of a synchronized consistent network-wide view among multiple controllers [142]. HyperFlow uses a publishing mechanism to maintain a consistent global view across controllers. Whenever a system status change is detected, each controller selectively publishes an event about the change through a publish/subscribe system. New status is then pushed to subscribed controllers for immediate updating.

Communication among multiple controllers is another method that can achieve synchronized global network view. In this category, Yin *et al.* propose "SDNi" for interconnectivity and message exchange among multiple SDN domains [62], [63]. An SDN domain is defined as the portion of the network being managed by a particular SDN controller. Specifically, "SDNi" is a more general purpose interface with a dual function. It can be used to share and synchronize network status information, and coordinate controllers' decision-making processes.

SDN applications also play an important role in that they might have different requirements on durability and consistency of the global network view. Using this observation, Koponen *et al.* present Onix allowing programmers to determine a trade-off between potentially simplified application and strict durability and consistency guarantee [143]. With Onix, an SDN application can choose and use a transactional persistent

database backed by a replicated state machine that ensures durability and consistency. As a result, this application can be simple without consistency consideration. The SDN applications can also choose a one-hop, eventually-consistent, memory-only Distributing Hash Table (DHT) that accommodates high update rate; however these applications need to handle inconsistencies themselves.

B. Policy and Rule Validation

Consistency in policies and rules stands out as an important design issue to stabilize the routing choice in SDN networks. This is due to the fact, in SDN networks, multiple applications could connect the same controller, and multiple controllers could be figured for performance improvement. As a result, conflicting configurations might surface, demanding an internal coordination among different participating units. Specifically, policies and rules should be validated to identify potential conflicts. Further, many well-developed methods, for example, role-based source authentication with priority [144], can be adopted to resolve these conflicts. In this subsection, we survey several existing research efforts to ensure validity of inter-domain and intra-switch policies as well as packet forwarding rules.

Model Checking, which is widely used to automatically verify correctness of a finite-state system, can be readily adopted for policy and rule validation. Along this line of research, FlowChecker is proposed to identify intra-switch misconfigurations and inter-switch inconsistencies leveraging model checking [145], [146]. Specifically, FlowChecker uses Binary Decision Diagrams (BBDs) to encode network configurations and models global behavior of a network in a single state machine for "what-if" analysis. FlowChecker further provides a generic property-based interface to verify reachability and security properties written in Computational Tree Logic (CTL), using BDD-based symbolic model checking and temporal logic. The use of CTL language makes it easier to write queries to validate certain properties or extract statistics to be used for further analysis. In another example, Canini *et al.* present NICE (No bugs In Controller Execution), which also adopts model checking to check correctness properties in SDN [147]–[149]. NICE takes an SDN application, network topology and correctness properties, for example, loop-free, as inputs, and then performs a state space search and produces traces of property violations. It uses model checking to explore system execution paths, symbolic execution to reduce space of inputs and search strategies to reduce state space. In practice, OFTEN (OpenFlow Testing Environment) is built based on NICE using physical switching devices whose states are synchronized with the switch models used in NICE [150]. OFTEN can be used for physical switching device black-box testing.

From another perspective, rules can be validated statically or dynamically. On one hand, the rules can be checked statically for certain network invariants, such as reachability, loop-free, and consistency, based on network topology [151]. On the other hand, it is also useful to check rules in real-time, as network state evolves. However, achieving extremely low latency during these checks is ultimately important. Khurshid *et al.* present VeriFlow to show that the goal of extremely low latency during real-time checks is achievable [152]–[154]. In their design, a proxy is introduced between a controller and switching devices to check network-wide invariant violations dynamically as each forwarding rule is updated. It first divides rules into equivalence classes based on prefix overlapping and uses prefix tree data structure to quickly find overlapping rules. Then, the proxy generates individual forwarding graphs for all the equivalent classes. As a result, queries for loops or black holes can be quickly replied by traversing a corresponding forwarding graph.

C. Control Layer Performance

The performance of SDN networks highly depends on the control layer, which, in turn, is constrained by the scalability of centralized controllers. Indeed, all the transactions in the control plane are involved with controllers. Switching devices need to request controllers for packet forwarding rules reactively when the first packet of each flow arrives. Rule update and network status collection also involve in frequent communication between controllers and switching devices. In this aspect, bandwidth consumption and latency of frequent communication affect control layer scalability significantly.

To address the scalability issue with an SDN controller, researchers have previously proposed multiple controllers with proper geographical placement [155], which would need network status synchronization. Alternative research efforts are also sought from a design aspect to increase processing ability of a single controller or decrease frequency of requests to be processed. We describe these research efforts in this subsection, and present performance benchmark techniques for SDN controllers.

1) Increasing Processing Ability: A controller is essentially a piece of software. As such, conventional software optimization techniques like parallelism and batching can be used to improve controller's performance on request processing, which is used in Maestro [156], [157], NOX-MT [158], and McNettle [159], [160]. Specifically, Maestro [156], [157] is a Java based controller implementation. It exploits parallelism together with additional throughput optimization techniques, such as, input and output batching, core and thread binding. It is demonstrated that this design leads to improved performance and near linear performance scalability on multi-core processors. In another case, NOX-MT is a multi-thread controller based on the single thread C++ implementation of Network Operating System (NOX) [158]. Benchmarking on different controllers, including NOX, NOX-MT, Maestro, and Beacon [161], shows performance advantages of NOX-MT over the others in terms of minimum and maximum response time, as well as maximum throughput. McNettle is an SDN controller written in Haskell, leveraging the multi-core facilities of the Glasgow Haskell Compiler (GHC) and runtime system [159], [160]. McNettle schedules event handlers, allocates memory, optimizes message parsing and serialization, and reduces the number of system calls in order to optimize cache usage, OS processing, and runtime system overhead. Experiments show that McNettle can serve up to 5000 switches using a single controller with

46 cores, achieving throughput of over 14 million flows per second.

2) Reducing Request Frequency: Heavy request load on a controller could result in a longer delay in SDN controllers [162]. As such, many strategies can be adopted to decrease request frequency. One of them is to modify switching devices so as to handle requests in the data plane or near the data plane. Another strategy is to refine the structure in which switching devices are organized. We discuss these two strategies in the following paragraphs.

Following the strategy of handling requests in the data plane, Yu *et al.* suggest distributing rules across "authority switches" [163]. Packets are diverted through "authority switches" as needed to access appropriate rules, thus all packets can be handled in the data plane without requesting to controllers. However, some packets may have to be directed through a long path to get appropriate rules. Similarly, Curtis *et al.* present DevoFlow to handle most "mice" flows in switching devices [164]. DevoFlow proactively installs a small set of possible packet forwarding rules in switching devices. As a result, Equal-Cost Multi-Path (ECMP) routing and rapid re-routing after designated output port goes down can be supported without requesting controllers. DevoFlow also uses sampling, triggering report after a threshold condition has been met, and approximates counters that only track statistics of the top-k largest "mice" flows. As a result, the amount of data in communication with controllers during statistic collection is reduced.

Proper organization and labour division of switching devices can also improve the overall control layer performance. For example, Yeganeh and Ganjali propose Kandoo, a framework for preserving scalability without changing switching devices [165]. Specifically, Kandoo has a two-layer architecture to handle most of frequent events locally. The bottom layer is a group of controllers without a network-wide view that handle most of frequent events. "Elephant" flow detection, which needs to constantly query each switching device to see whether a flow has enough data to be an "elephant" flow, can be done at the bottom layer. At the same time, the top layer is a logically centralized controller that maintains a network-wide view and handles rare events, for example, requesting for routing decisions. As a result of the two-layer architecture, heavy communication burden is offloaded to highly replicable local controllers at the bottom layer.

3) Performance Benchmarking: Controller performance benchmarking can be used to identify performance bottlenecks and is essential to increase processing ability of a controller. Cbench (controller benchmarker) [166] and OFCBenchmark [167] are two tools designed for controller benchmarking. Cbench [166] tests controller performance by generating requests for packet forwarding rules and watching for responses from the controller. Cbench offers aggregated statistics of controller throughput and response time for all the switching devices. Aggregated statistics may not be sufficient enough to explore detailed controller behavior. On this consideration, Jarschel *et al.* present OFCBenchmark with fine-grained statistics for individual switching devices [167]. OFCBenchmark provides statistics of response rate, response time, and number of unanswered packets for each switching device.

V. Application Layer

As illustrated in Fig. 1, the application layer resides above the control layer. Through the control layer, SDN applications can conveniently access a global network view with instantaneous status through a northbound interface of controllers, for example, the Application Layer Traffic Optimization (ALTO) protocol [168], [169] and the eXtensible Session Protocol (XSP) [170]. Equipped with this information, SDN applications can programmatically implement strategies to manipulate the underlying physical networks using a high level language provided by the control layer. In this aspect, SDN offers "Platform as a Service" model for networking [171]. In the section, we describe several SDN applications built on this platform.

A. Adaptive Routing

Packet switching and routing are the main functions of a network. Traditionally, switching and routing designs are based on distributed approaches for robustness. However, such distributed designs have many shortcomings, including complex implementation, slow convergence [180], and limited ability to achieve adaptive control [181]. As an alternative solution, SDN offers closed loop control, feeding applications with timely global network status information and permitting applications to adaptively control a network. Seeing this opportunity, several proposals have been made to utilize the SDN platform for better routing designs. In the following paragraphs, we describe two popular SDN applications in this domain, namely load balancing and cross-layer design.

1) Load Balancing: Load balancing is a widely used technique to achieve better resource usage. A common practice of load balancing in data centers is deploying front-end load balancers to direct each client's request to a particular server replica to increase throughput, reduce response time, and avoid overloading of network. Dedicated load balancers, however, are usually very expensive. SDN enables an alternative approach. In the following paragraphs, we first discuss algorithms to balance load using packet forwarding rules, and then present uses cases in various scenarios.

Wang *et al.* make initial efforts to build a model and develop algorithms for load balancing using packet forwarding rules of SDN [124]. They assume uniform traffic from all clients with different IP addresses and propose to use a binary tree to arrange IP prefixes. The traffic is then divided using wild card rules, so that a server replica will handle a traffic whose volume is in proportion to processing ability of the server replica. Although the assumption may not be true in most cases, this work establishes a basis for future research on load balancing leveraging packet forwarding rules of SDN. Besides computing a path for each traffic flow proactively to achieve balanced load, another methodology is to migrate traffic from heavy loaded switching devices to lightly loaded ones reactively [182].

While algorithm development is essential, others make efforts to deploy load balancing with SDN in various scenarios. Different services or tenants may need their own dedicated and specialized load balancing algorithm implementations and do not want to affect each other, even if some load

balancing implementations break down. On this consideration, Koerner *et al.* introduce differentiated load balancing algorithms for different types of traffic, for example, web traffic and email traffic, to achieve dedicated and specialized balancing algorithm implementations depending on requirements of services and workloads [183]. In another situation, a front-end load balancer needs to direct every request and can become the bottleneck of a data center. To solve this problem, Handigol *et al.* present Plug-n-Serve (now called Aster*x [53]), which balances load over an arbitrary unstructured network using an implementation of SDN [172]. It directly controls paths taken by new HTTP requests to minimize average response time of web services.

2) Cross-Layer Design: A cross-layer approach is a highly touted technique to enhance integration of entities at different layers in a layered architecture, for example, the OSI reference model, by allowing entities at different layers to exchange information among each other. As SDN offers a platform for applications to easily access network status information, cross-layer approaches can be easily developed on this platform. In the following paragraphs, we present use cases to provide guaranteed QoS and improved application performance leveraging the cross-layer design technique.

Many network applications require a certain level of QoS support. Utilizing QoS information for appropriate network resource reservation represents one effective cross-layer approach to achieve guaranteed QoS. For example, Ferguson *et al.* demonstrate better QoS for video conferencing by retrieving schedule of available bandwidth from an SDN controller. Then, the earliest time at which a video call or alternatively an audio call can be made with guaranteed quality is calculated [55]. As another example, Jeong *et al.* present QoS-aware Network Operating System (QNOX) to offer QoS guaranteed services, such as QoS-aware virtual network embedding and end-to-end network QoS assessment [56]. For a request demanding a virtual network with QoS requirements on bandwidth of virtual link and delay between virtual nodes, QNOX will make QoS-aware mapping for the virtual network on the substrate network. QNOX also monitors end-to-end QoS offering measured results to help make operational changes.

Similarly, adaptive routing can also improves applications' performance. For example, Wang *et al.* propose a cross-layer approach to configure underlying network at runtime based on big data application dynamics [184], taking advantages of high reconfigurability of SDN switching devices as well as high speed and reconfigurability of optical switches. They use Hadoop as an example and design Hadoop job scheduling strategies to accommodate dynamic network configuration on a hybrid network with Ethernet and optical switches. Their preliminary analysis reports improved application performance and network utilization with relatively low configuration overhead.

B. Boundless Roaming

Smartphones and tablets are becoming dominating devices in the Internet access. These mobile devices access the Internet wirelessly. To ensure continuous connectivity while these devices move from one location to another, connections may be handed over from one base station to another, or even from one wireless network to another. Seamlessness handover is critical for applications to provide uninterrupted services. Handover in the current literature is often limited to networks of a single carrier with the same technology. In SDN, networks of different carriers with different technologies could have a common unified control plane. This enables boundless mobility with seamless wireless connection handover between different technologies and carriers, as shown in Fig. 2.

Various handover schemes have been developed based on SDN. For example, Yap *et al.* propose handover algorithms between Wi-Fi and WiMAX networks including Hoolock, which exploits multiple interfaces on a device, and n-casting, which duplicates traffic across n distinct paths [173]–[175]. These handover schemes can be easily implemented with SDN to reduce packet loss and improve TCP throughput during handover. Another use case is Odin, which is a prototype SDN framework for enterprise WLANs [176]. Odin allocates a unique Basic Service Set Identification (BSSID) for each client connected. Handover is performed by removing the BSSID from one physical wireless Access Point (AP) and spawning it to another. Odin exhibits low delay in re-association, no throughput degradation and minimum impact on HTTP download in either a single or multiple handovers.

C. Network Maintenance

Configuration errors are common causes of network failures. It is reported that more than 60% of network downtime is due to human configuration errors [185]. What makes it worse is that existing network tools that deal with individual diagnosis such as ping, traceroute, tcpdump, and NetFlow, fail to provide an automated and comprehensive network maintenance solution. As a comparison, centralized and automated management and consistent policy enforcement, inherent in SDN networks, help reduce configuration errors. Moreover, with a global view and central control of configuration, SDN offers opportunities to design comprehensive network diagnosis and prognosis mechanisms for automated network maintenance, as described in the following paragraphs.

Network diagnosis tools like *ndb* [186] and OFRewind [177] are crucial to detect causes of network failure. Inspired by *gdb*, *ndb* is a network debugger providing backtrace of network events. Every time a packet visits a switching device, a small "postcard" is sent back to a controller. The controller will then build a backtrace for network debugging. Taking a similar strategy, OFRewind constantly records network events appearing in a network. A novel feature of OFRewind is that it replays recorded events later to troubleshoot the network.

A key benefit of SDN-based prognosis mechanisms is that central control of an SDN implementation can directly resolve network failures with shorter routing convergence time [180]. For example, Sharma *et al.* propose a fast restoration mechanism for SDN [187]. After detection of failure, the controller calculates new forwarding paths for affected paths and updates packet forwarding rules immediately without waiting for old forwarding rules to expire.

D. Network Security

Network security is a notable part of cyber security and is gaining attentions. Traditional network security practices deploy firewalls and proxy servers to protect a physical network. Due to the heterogeneity in network applications, ensuring exclusive accesses by legitimate network applications involves implementation of a network-wide policy and tedious configuration of firewalls, proxy servers, and other devices. In this aspect, SDN offers a convenient platform to centralize, merge and check policies and configurations to make sure that the implementation meets required protection thus preventing security breaches proactively.

Moreover, SDN provides better ways to detect and defend attacks reactively. Ability to collect network status of SDN allows analysis of traffic patterns for potential security threats. Attacks, such as low-rate burst attacks and Distributed Denial-of-Service (DDoS) attacks, can be detected just by analyzing traffic pattern [71], [188]. At the same time, SDN provides programmatic control over traffic flows. Consequently, traffic of interest can be explicitly directed to Intrusion Prevention Systems (IPSs) for Deep Packet Inspection (DPI) [189], [190]. If attacks are detected, SDN can install packet forwarding rules to switching devices to block the attack traffic from entering and propagating in a network [55], [119]. Centralized control of SDN permits dynamically quarantine of compromised hosts and authentication of legitimate hosts based on information obtained through requesting end hosts [191], requesting a Remote Authentication Dial In User Service (RADIUS) [192] server for users' authentication information [193], [194], tainting traffic [195], [196] or system scanning during registration [197].

Finally, SDN is further capable of providing direct and fine-grained control over networks, and gives opportunities to implement novel security protection strategies. For example, Jafarian *et al.* develop Moving Target Defense (MTD) based on efficient control of SDN. A virtual IP is associated with each host for data transmission and is randomly mutated with high unpredictability and rate while a real IP of the host is static. Controllers will specify translation between the virtual IP and real IP while maintaining configuration integrity [178]. As another example, Mendonca *et al.* present AnonyFlow, an in-network endpoint anonymization service designed to provide privacy to users. AnonyFlow performs translation between AnonID, Network IP and Machine IP using an implementation of SDN [179].

E. Network Virtualization

Network virtualization is a popular technique to allow multiple heterogeneous network architectures to cohabit on a shared infrastructure [198] and plays a significant role in the IaaS model. A common practice of network virtualization is to slice a physical network into multiple virtual instances and assign them to different users, controllers, or SDN applications, as illustrated in Fig. 5. Conventional virtualization methods using tunnels and VLAN or MPLS tags require tedious configurations on all the involved network devices. As a comparison, SDN offers a platform allowing configuration of all switching devices in a network from a controller, for example, libNetVirt [199], [200]. With this platform, different strategies can be developed at the application layer to automate configuration for network slicing.

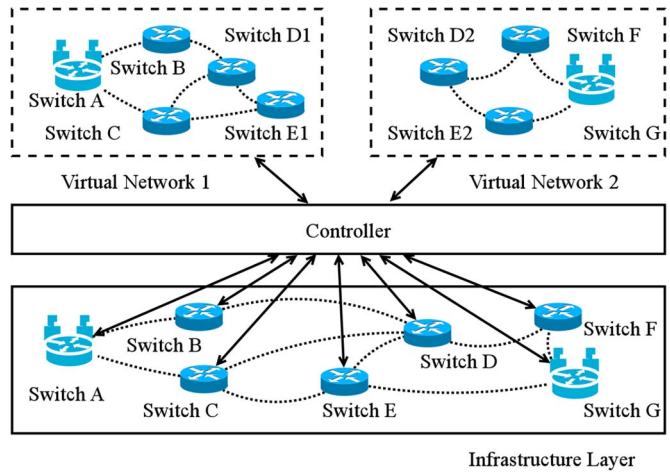

Fig. 5. Network virtualization: Multiple virtual networks can be created on the same physical network, sharing infrastructure resources. An SDN application can only oversee and user resources of its own virtual network.

In SDN research, FlowVisor is one leading example that provides functions to slice the network resources, including bandwidth, topology, flow space, switching device CPU, forwarding table space, and control channel [201], [202]. FlowVisor is located between guest controllers and switching devices acting as a transparent proxy to filter control messages such that a guest controller can only see and manipulate its own virtual network. FlowVisor is a useful tool to create virtual networks from a physical network for research experimentations [203] and to share a physical network with various users with clear isolation [204]. In another approach, Gutz *et al.* introduce a network virtualization approach by providing isolation at a language level [121]. In this approach, taking a collection of slice definitions and their associated applications as an input, a list of packet forwarding rules is generated for each switching device to create an appropriate virtual network for each application.

F. Green Networking

Green networking has become important in network design and deployment for economic and environmental benefits. Different approaches have been considered to achieve green networking, including, but not limited to, energy-aware data link adaptation, energy-aware traffic proxying, energy-aware infrastructure and energy-aware application, as suggested in [205].

It turns out that SDN switching devices may not directly offer benefits in energy reduction in network operation [206]. However, SDN could offer significant promises in supporting minimization of network-wide energy consumption. As a proof, Heller *et al.* demonstrate energy-aware data link adaptation with SDN [54]. They propose a mechanism to determine minimum data links and switching devices for a data center network based on traffic loads and dynamically power down redundant links and switching devices for energy efficient operations.

Ex. 1011
Juniper Networks, Inc. / Page 14 of 25

TABLE III
EXPERIMENT SETUP OF SDN APPLICATIONS

Name	Standard	Method	Scale	Controller	Switch
Plug-n-Serve [53], [172]	OpenFlow	real testbed	about 10 switches	NOX-based	commercial switches from Cisco, HP and NEC, and NetFPGA-based
PANE [55]	OpenFlow	Mininet simulation		Nettle-based	Open vSwitch
QNOX [56]	ForCES	virtual testbed	6 PC servers hosting 114 FEs	build on CE	Linux based switches on VMs
OpenRoads [173]–[175]	OpenFlow	real testbed	5 Ethernet switches, 30 Wi-Fi APs, and 1 WiMAX basestation	NOX	switches from NEC and HP, WiMAX basestation built by NEC, and WiFi APs based on ALIX PCEngine boxes
Odin [176]	OpenFlow	real testbed	a client, a server, and 2 APs	NOX	Open vSwitch on Atheros AR9280 wireless Card
OFRewind [177]	OpenFlow	real testbed	10 PCs, 5 switches	NOX	switches from 3 vendors
OF-RHM [178]	OpenFlow	Mininet simulation	2 virtual switches	NOX	Open vSwitch based
AnonyFlow [179]	OpenFlow	real testbed	4 switches	NOX-based	2 commercial switches and 2 NetFPGA based switches
ElasticTree [54]	OpenFlow	real testbed	3 physical switches	NOX	switches from NEC and HP

G. SDN for Cloud Computing

Cloud computing is changing the way people do computing and business. It provisions computing and storage resources on demand and charges on usage with server and network virtualization. SDN provides opportunities to extend the service provisioning model of IaaS beyond computing and storage resources to include a rich set of accompanying network services for more flexible and efficient cloud computing [207].

Data center networks for cloud computing have a few key requirements, including scalability for large scale deployment, location independence for dynamic resource provision, QoS differentiation for different tenants, and network visibility and fine-grained control [208]. As aforementioned in previous subsection, SDN can fully meet these requirements. In the following paragraphs, we discuss two unique issues for cloud computing, namely virtual switching and VM migration.

Virtual switching is used for communication among VMs in the same host. Conventional virtual switching provided with hypervisors, however, does not provide sufficient visibility and control. As a solution, Open vSwitch provides virtual edge switching for VMs with visibility and control leveraging the idea of SDN [74], [75]. These vSwitches also report network status to and receive packet forwarding rule from SDN controllers, just like SDN switching devices. However, vSwitches offer limited storage and processing resources compared with physical switches. Moshref *et al.* present a virtual Cloud Rule Information Base (vCRIB) to overcome these resource constraints [209]. vCRIB automatically finds the optimal rule placement among physical and virtual switches while minimizing traffic overhead and adapting quickly to cloud dynamics such as traffic changes and VM migrations.

VM migration is widely used in data centers for statistical multiplexing or dynamic communication pattern changing to achieve higher bandwidth for tightly coupled hosts [210]. Conventional VM migration is often limited to a single broadcast domain, since IP addresses should be preserved across broadcast domains. However, Address Resolution Protocol (ARP) messages cannot go beyond a broadcast domain. Mobile IP and Locator/Identifier Separation Protocol (LISP) based solutions [211]–[213] can fix this issue and have inspired SDN solutions. Specifically, the OpenDaylight [214] SDN controller has added a LISP-based mapping service. SDN can preserve VM connectivity during intra and inter data center VM migration [182], [215]–[221] by inserting proper packet forwarding rules in switching devices to direct traffic to the new VM location. As an example of inter data centers VM migration, Mann *et al.* present CrossRoads to migrate VM across data centers seamlessly using an implementation of SDN [218]. CrossRoads extends the idea of location independence based on pseudo addresses proposed in cloud networking to work with controllers governing their own data center network. Cross subnet ARP resolution is used to distinguish an IP address outside a subnet after inter data center migration. The resolution will then direct packets with this IP address to their destination in an external data center. As a result, VM connections can be maintained during inter data center migration.

VI. CASE IN POINT: OPENFLOW

OpenFlow is first proposed by McKeown *et al.* with an objective to enable easy network experiments in a campus network [222] and is currently used in most SDN practices as shown in Table III. Early phase experiments using OpenFlow mainly aim at creating a separate software controllable network focusing on controlling forwarding of packets. Later, researches discovered that the implementation of a separate software controllable network using OpenFlow is actually a practical enabler of the so called "Software-Defined Networking".

OpenFlow takes advantages of the fact that most modern Ethernet switches and routers contain flow tables for essential networking functions, such as routing, subnetting, firewall protection, and statistical analysis of data streams. In an OpenFlow switch, each entry in the flow table has three parts, "header" for matching received packets, "action" to define what to do for the matched packets, and "statistics" of the matched traffic flow. As illustrated in Fig. 6, the OpenFlow protocol offers convenient flow table manipulation services for a controller to insert, delete, modify, and lookup the flow table entries through a secure TCP channel remotely.

OpenFlow may be seen as just a protocol specification used in switching devices and controllers interfacing. Its idea of

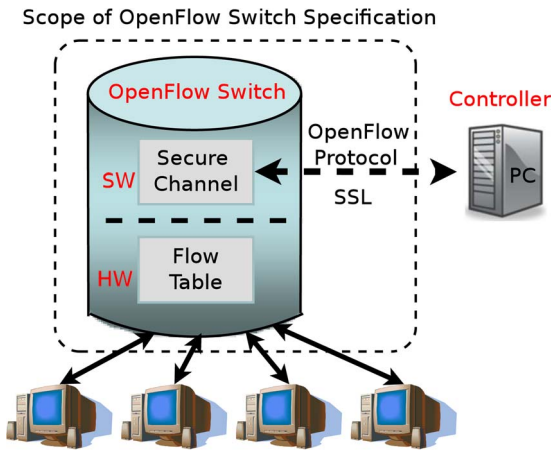

Fig. 6. OpenFlow Switch. The flow table is controlled by a remote controller via the secure channel [222].

creating a separate network solely for network control manifests the key concept of SDN and lays foundation for network programmability and logically centralized control. In the development of SDN and OpenFlow, their concepts and design approaches go hand in hand with each other. On one hand, many concepts in SDN are based on the design of OpenFlow. On the other hand, as the concept of SDN becomes clearer and more mature, then it influences the future development of OpenFlow. In other words, OpenFlow defines initial concept of SDN and SDN governs future development of OpenFlow. In the following subsections, we first present standardization process and deployment cases of OpenFlow. Then we introduce some widely used OpenFlow software projects, and lastly compare OpenFlow with ForCES, which is also a protocol to separate the control and data planes [37]–[40].

A. Standardization and Deployment

The OpenFlow specification is continuously evolving with new features in every new release. With more and more vendors releasing their OpenFlow-enabled products and solutions, more software projects being developed on OpenFlow and more organizations deploying OpenFlow-enabled networks, a complete and well-functioned ecosystem is being built around OpenFlow.

The OpenFlow Switch Consortium [223] released the first OpenFlow reference implementation, version 0.1.0, with source code on November 30, 2007. It then published the OpenFlow version 1.0 on December 31, 2009, which added multiple queues per output port for minimum bandwidth guarantees. The next version 1.1 was released on February 28, 2011, which introduced multiple tables pipeline processing. After that, the role of standardizing OpenFlow specification was moved to ONF [29]. In December 2011, the ONF board approved OpenFlow version 1.2 and published it in February 2012, which added support for IPv6. Later on April 19, 2012, ONF further approved OpenFlow version 1.3 and on June 25, a rectified version was released with OF-Config 1.1, which is a protocol to configure and manage OpenFlow switches and controllers.

Along with the process of OpenFlow standardization, many OpenFlow switches and controllers surface. The first OpenFlow switches are built using open network hardware, for example, NetFPGA [80], [81]. After the first appearance, vendors, including NEC, IBM, Juniper, HP, Cisco, Huawei, Pica8/Pronto, Centec, Dell/Force10, Extreme, Mellanox, NoviFlow, Arista, Brocade, NetGear and many more, start to release their OpenFlow switches. For controllers, there are commercial controllers, such as ProgrammableFlow Controller from NEC and Big Network Controller from Big Switch, as well as open source controllers, such as OMNI [224], Trema [225], Ryu [226], Floodlight [227], NOX [228], [229], and OpenDaylight [214], as listed in Table IV. Other networking related projects also start to utilize OpenFlow, for example, OpenStack Quantum for cloud computing networks.

With so many OpenFlow switches and controllers available, OpenFlow networks have been and are being deployed for both research and production purposes. The first large scale OpenFlow network is deployed by the Stanford OpenRoads project [230]. The Stanford OpenRoads deployment consists of five 48-port 1GE OpenFlow Ethernet switches, 30 Wi-Fi APs and 1 WiMAX base station. In this testbed, SSIDs are used to identify network slices. A DHCP server is used to allocate IP addresses to mobile clients. OpenRoads also has a logging system, data graphing tools and real-time visualization to monitor the system. These tools are carefully designed to be complementary yet independent, and they are reusable for other deployments. The OpenRoads deployment opens the road to deploy OpenFlow in campus networks with both wired and wireless connections. Other than research purpose deployments in campus networks, OpenFlow has also been deployed in production networks. Google has deployed a software defined WAN called "B4" connecting Google's data centers across the planet for three years [231]. B4 is a hybrid approach with simultaneous support of existing routing protocols and novel OpenFlow SDN approach. For the data plane, Google builds its own OpenFlow enabled switches from multiple merchant silicon switch chips. To support legacy routing protocols like BGP and IS-IS, B4 runs an open source Quagga [34] stack on the control plane. Routing protocol packets are redirected to Quagga from the data plane. Routing table updates of Quagga are then translated to flow table updates on the switches. With techniques like centralized traffic engineering to share bandwidth among competing applications with possibility to use multiple paths, remarkably, B4 shows near 100% link utilization. B4 supports existing routing protocols and OpenFlow, thus offers a less disruptive and more economical solution for OpenFlow deployment in production networks for businesses. OpenFlow networks have expanded rapidly, since its first deployment. Now, OpenFlow products are permeating into laboratories, classrooms [232], testbed networks [58], [59], [233]–[237], data centers, and service provider networks [238].

B. OpenFlow Software Projects

Many OpenFlow software projects exist today. Among them NOX controller [228], [229] and Mininet simulator [239] are the most frequently used tools in SDN related researches.

NOX is the first OpenFlow controller. It allows applications to be implemented based on a centralized network view using

TABLE IV
OPENFLOW CONTROLLERS

Name	Platform	Organization	Features	Site
NOX/POX	C++/Python	Stanford	the first OpenFlow controller	http://www.noxrepo.org/
Beacon	Java	Stanford	support both event-based and threaded operation, provide Web UI	https://openflow.stanford.edu/display/Beacon/Home
Floodlight	Java	Big Switch	developer-friendly with plenty documents	http://floodlight.openflowhub.org/
Ryu	Python	NTT laboratories OSRG group	with OpenStack support	http://osrg.github.io/ryu/
Maestro	Java	Rice University	Multi-threaded support	http://code.google.com/p/maestro-platform/
OMNI	Pyhton/Java	Universidade Federal do Rio de Janeiro	web interface on NOX	http://www.gta.ufrj.br/omni/
McNettle	Haskell	Yale University	high-level declarative expressive language, multi-core optimization	http://haskell.cs.yale.edu/nettle/mcnettle/
Trema	Ruby/C		virtual network DSL	http://trema.github.io/trema/
OpenDaylight Controller	Java	Linux Foundation	provide REST API and web GUI	http://www.opendaylight.org/

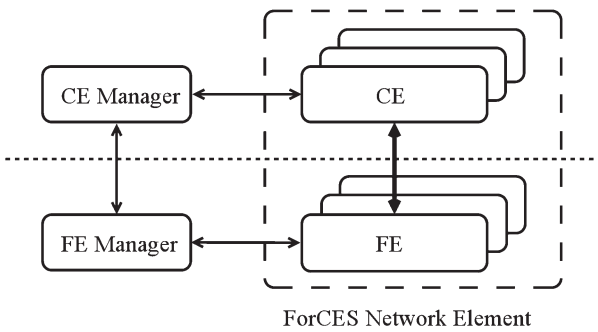

Fig. 7. IETF ForCES Architecture. A ForCES network element consists two kinds of components, namely control elements (CEs) and forwarding elements (FEs). CE Manager and FE Manager, which reside outside of the ForCES NE, provide configuration to the corresponding CE or FE in the pre-association phase.

high level names as opposed to distributed algorithms over low-level addresses. Applications are written in either Python or C++ and are loaded dynamically. Core infrastructure and speed-critical functions of NOX are implemented in C++.

Mininet is a network simulator for rapidly prototyping of a large OpenFlow network. A virtual network is created according to specified links, hosts, switching devices and controllers. A Command-Line Interface (CLI) is provided to interact with the virtual network, for example, checking connectivity between two hosts using ping. Since Mininet provides a simulated environment for experimentation, new ideas can be developed and tested in Mininet before deployed onto a real environment in a straightforward way. Usage of lightweight OS-level virtualization features, including processes and network namespaces, to create virtual networks allows Mininet to scale to hundreds of nodes in a single computer.

C. OpenFlow and ForCES

OpenFlow aims at separating the control plane from the data plane. Another well known effort to separate the control plane from the data plane and standardize information exchange between the control and data planes is ForCES [37]–[40] proposed by IETF. As shown in Fig. 7, a ForCES Network Element (NE) consists of multiple Forwarding Elements (FEs) and multiple Control Elements (CEs). FE provides per-packet processing and is instructed by CEs on how to process packets.

TABLE V
COMPARISONS BETWEEN OPENFLOW AND FORCES

	OpenFlow	ForCES
Goal	implementation of SDN	separation of forwarding and control planes
Architecture	both network devices and architecture are changed	network architecture remains unchanged
Forwarding Model	flow table	Logical Functional Block
Protocol Interface	OpenFlow Protocol and OF-Config	ForCES PL and TML

ForCES uses ForCES Protocol Layer (ForCES PL) to define the protocol between FEs and CEs, and ForCES Protocol Transport Mapping Layer (ForCES TML) to transport the PL messages. Thus, ForCES allows coexistence of multiple TMLs from various vendors and interoperability is guaranteed as long as both endpoints support the same TML. Packet forwarding in FE is based on abstraction of Logical Functional Blocks (LFBs) [240], each of which has a single specific function of processing packets. In the following paragraphs, we present a comprehensive comparison between OpenFlow and ForCES in terms of their goals, architecture, forwarding model and protocol interface, as summarized in Table V [241], [242]:

- *Goal*: ForCES is not designed with a long-term vision to implement SDN. The goal of ForCES is separating the data plane from the control plane, while OpenFlow is designed for SDN;
- *Architecture*: ForCES NEs are functionally equivalent to conventional routers and they still run routing protocols just as conventional routers. Unlike ForCES, OpenFlow switches will be controlled by a controller and can run without any routing protocols;
- *Forwarding Model*: OpenFlow forwarding model provides programmability that is restricted to predefined capabilities of OpenFlow switches. For example, only predefined actions can be chosen to process a packet with OpenFlow. ForCES forwarding engine is more flexible, since function of each LFB and LFB topology can be dynamically specified. Thus, new packet processing actions can be created;
- *Protocol Interface*: In terms of protocol messages, ForCES supports more features than OpenFlow, such as message batching, execution mode selection, and command pipelining.

TABLE VI
SDN RELATED RESEARCHES

Layer		Issue	Approach and Reference	Related Area
Infrastructure	Switching Device	Storage	reduce memory usage [65], efficient usage of memory [66], combine different storage hardwares [67], [68]	Integrated Circuits, Embedded Systems, Hardware Testing
		Processing	combine different processing hardwares [69]–[71]	
		Hardware Platform	software implementation, firmware implementation, implementation on open hardware platform [32], [76]–[78]	
		Performance Evaluation	[83], [84]	
	Transmission Media	Wireless Radio	OpenRadio [87]	SDR [85], Mobile Ad Hoc
		Optical Fibers	unified control plane [94]–[103], [107], split control plane [105], [106], [108], [109]	GMPLS, ROADMs [92]
Control	Southbound Data Stream	High Level Language	FML [55], [111], [118], [119], Frenetic [112], [113], [120]–[122], Nettle [114]–[117]	Programming Language, Formal Methods, Compilers
		Rule Update	[126]–[128]	Distributed System
		Policy and Rule Validation	FlowChecker [145], NICE [147]–[150], VeriFlow [152]–[154]	Formal methods
	Northbound Data Stream	Network Status Collection	OpenTM [131]	Network Measurement
		Network Status Synchronization	HyperFlow [142], Onix [143]	Distributed System, Database [243]
	Performance	Increase Processing Ability	parallelism [156]–[158]	Algorithm Analysis, Software Engineering
		Decrease Request Frequency	"authority switches" [163], DevoFlow [164], Kandoo [165]	
Application		Adaptive Routing	load balancing [53], [124], [172], [183], cross-layer design [55], [56], [184], multicast [246], [246]–[250]	Routing [244], [245], Traffic Engineering, NDN [61], [251]
		Boundless Mobility	OpenRoads [173]–[175], Odin [176]	Wireless Network,
		Consolidate Security	attack detection [71], [188], anonymization [179], MTD [178], access control [195]–[197]	Computer Security, Function Outsourcing [189],
		Network Virtualization	FlowVisor [201], [202], language level virtualization [121]	Network Virtualization, Web Caching [252],
		Ease Maintenance	debugger [177], [186], restoration [187]	
		Green Networking	ElasticTree [54]	Green Networking,
		Cloud Computing	Open vSwitch [74], [75], VM migration [182], [215]–[220]	Cloud [207], [253], [254]

In summary, ForCES provides more flexible forwarding model and richer protocol features. However, because of the disruptive business model brought by the LFB forwarding model and lack of open source support, ForCES is not so widely adopted as OpenFlow. OpenFlow can still learn a lot from both merits and shortcomings of ForCES for further success.

VII. CONCLUSION, DESIGN GUIDELINES AND FUTURE RESEARCH

To conclude, we first present a brief summary of the whole article. Then we list design principles that have been adopted in SDN related researches. Finally, we point out a few SDN related open issues that need future research efforts.

A. Summary and Conclusion

Recent developments in ICT domain, for example, mobile, multimedia, cloud, and big data, are demanding for more convenient Internet access, more bandwidth from users, as well as more dynamic management from service providers. SDN is considered as a promising solution to meet these demands. In this paper, we have presented the concept of SDN and highlighted benefits of SDN in offering enhanced configuration, improved performance, and encouraged innovation. Moreover, we have provided a literature survey of recent SDN researches in the infrastructure layer, the control layer, and the application layer, as summarized in Table VI. Finally, we have introduced OpenFlow, the de facto SDN implementation.

B. Design Guidelines

The success of SDN requires improvements and developments at all the three layers, including the infrastructure layer, the control layer, and the application layer. It needs collaboration of different organizations including vendors, academia, and communities, and interdisciplinary knowledge covering both hardware and software. In the following paragraphs, we outline a few design guidelines for further development and future research in SDN:

- An SDN switching device is relatively simple with a separate control plane. The SDN switching device can be easier to manufacture and its cost will be cheaper using merchant silicon. However, issues on switching device hardware design are still open. Specifically, SDN switching devices need more memory space and higher processing speed with an economically viable cost. Integration of various new hardware technologies is necessary.

TABLE VII
CURRENT RESEARCH PROJECTS

Name	Research Area	Organization	Site Address
ONRC	Modern SDN Stack, OpenRadio and Software Defined Cellular, Wireless Networking, Redesigning the Network Stack for Mobile Devices, OpenFlow Optimized Switch ASIC Design, SDN for Home Networks, Header Space Analysis, Load-Balancing as a Network Primitive, Unified Control Architecture for Packet and Circuit Networks, An SDN Approach to MPLS Traffic Engineering and Virtual Private Networks.	Stanford, Berkerly	http://onrc.net/
PANE	Participatory Networking (PANE) is a prototype OpenFlow controller integrating end-users, hosts and applications in network management by supporting resource requests, traffic hints, and status queries.	Brown University	http://pane.cs.brown.edu/
Frenetic	A programming language for OpenFlow networks.	Cornell, Princeton	http://www.frenetic-lang.org/
BISmark	OpenWRT-based platform for performing measurements of ISP performance, as well as traffic inside the home.	Georgia Tech, University of Napoli Federico II	http://projectbismark.net/
RouteFlow	virtualized IP routing services over OpenFlow enabled hardware.	CPqD	https://sites.google.com/site/routeflow/
InCNTRE SDNLab	SDN testing tools, methodologies, and procedures.	Indiana University	http://incntre.iu.edu/SDNlab
FP7 SDN related Programs	Federated E-infrastructure Dedicated to European Researchers Innovating in Computing network Architectures (FEDERICA), Split Architecture Carrier Grade Networks (SPARC), OpenFlow in Europe Linking Infrastructure and Applications (OFELIA), Future Internet testbeds experimentation between Brazil and Europe (FIBRE), OpenFlow Experiment in Real-time Internet Edutainment (OFERTIE)	mainly organizations in Europe	http://ec.europa.eu/research/fp7/

- SDN is originally developed for IP based networks on campuses. Therefore, scaling ubiquitous coverage of SDN requires unification and integration with advanced technologies in wireless and optical transmission (i.e., SDR and GMPLS, respectively). These technologies give opportunities for an SDN controller to have a widespread control over all the network parameters and behavior. Integrated with these technologies, SDN will obtain more appropriate control of the infrastructure to achieve more efficient infrastructure resource utilization.
- To enhance advantages of decoupling the control plane from the data plane, a high level expressive and comprehensive interface to access and control switching devices should be provided to further ease network configuration and management. Skills from various computer science areas, such as programming language theory, formal methods, and distributed systems, should be applied to enable automated generation from high level language described policies to low level rules without conflicts and to guarantee consistency during rule update procedure.
- Network measurement techniques are also useful for network status collection. For large scale networks, multiple controllers are necessary. Synchronization algorithms studied in distributed systems and database systems can be adopted to synchronize collected network status among multiple controllers.
- An SDN controller has to handle a massive amount of interaction events with its associated switching devices. To guarantee efficiency of network operations, methods in software optimization and algorithm analysis can be used to improve controller's performance, and properly designed architecture can help decrease request frequency.
- SDN provides a platform to implement various SDN applications. SDN applications can access a global network view and cross-layer information to make better network operation decisions. SDN controllers ensure that these decisions are carried out properly at the infrastructure layer. Still, participation of software developers is required to turn innovative ideas to solutions that can bring economic, social, and environmental benefits.

C. Future Research

Many challenges in SDN still need further research attention and many organizations have started research projects in various aspects of SDN as shown in Table VII. In the following paragraphs, we list the open issues covering the whole lifecycle of SDN from standardization, implementation, to deployment:

- *Standardization of SDN*: Being the de facto implementation of the SDN concept, OpenFlow is by no means the only SDN implementation. IETF has also released an SDN framework [255]. The ETSI Industry Specification Group (ISG) for Network Functions Virtualization (NFV) has been formed recently to promote NFV, which is highly complementary to SDN [256]. Upon maturity of SDN implementations, a comprehensive comparison among all potential SDN implementations should be conducted. In the control layer, we have witnessed many projects aiming at similar problems using similar approaches. However, a dominating solution has not yet arisen. Fragmentation in controller functions and APIs provided by the controllers may be a potential barrier for commercial development of SDN. Besides, OpenFlow specification evolves rapidly and allows different interpretations. Therefore, different implementations may behave inconsistently and cause unpredictable disorders [257].
- *Implementation of SDN*: The current SDN approach of decoupling control plane completely from data plane suggests a total removal of any onboard routing protocols from switching devices. This approach may be too idealistic, which may prevent SDN from widespread adaptation. An immediate and direct transform to idealistic SDN will

need a lot risky investment to replace all the conventional network devices. In the transition phase from conventional switching devices to fully decoupled SDN switching devices, semi-decoupled switching devices, which run routing protocols and can also be remotely controlled are helpful to smooth the evolution to idealistic SDN. Some other design drawbacks in the current SDN implementation do exist. For example, in OpenFlow, a long header is used for matching to allow more granular control, i.e., twelve tuples in OpenFlow 1.0 and more in OpenFlow 1.1. This may require more space in rule storage and more time in lookup. Besides, widely used proxy design between switching devices and controller has already shown a negative effect on performance [152], [204].

- *Deployment of SDN*: Further study of SDN in carrier networks with carrier-grade requirements [258], wireless mesh networks with fast client mobility [259], and wireless sensor networks which require high reliability and reachability [260], [261] is also needed for wide deployment of SDN.

Finally, we note that these issues pointed out here are not meant to be exhaustive. However, many more issues remain to be attended in this active and promising area of SDN.

REFERENCES

[1] *IDC Predictions 2013: Competing on the 3rd Platform*, IDC, Framingham, MA, USA, Nov. 2012, White Paper. [Online]. Available: http://www.idc.com/research/Predictions13/downloadable/238044.pdf

[2] P. Mell and T. Grance, "The NIST definition of cloud computing (draft)," NIST Special Publication, vol. 800-145, p. 7, 2011.

[3] J. Gantz and D. Reinsel, "Extracting value from chaos," IDC, Framingham, MA, USA, White Paper, Jun. 2011. [Online]. Available: http://www.emc.com/collateral/analyst-reports/idc-extracting-value-from-chaos-ar.pdf

[4] J. Manyika *et al.*, "Big data: The next frontier for innovation, competition, and productivity," McKinsey Global Inst., Mumbai, India, pp. 1–137, 2011.

[5] P. Cesar and D. Geerts, "Past, present, and future of social TV: A categorization," in *Proc. IEEE CCNC*, 2011, pp. 347–351.

[6] Y. Jin, X. Liu, Y. Wen, and J. Cai, "Inter-screen interaction for session recognition and transfer based on cloud centric media network," in *Proc. IEEE ISCAS*, 2013, pp. 877–880.

[7] Y. Jin, Y. Wen, G. Shi, G. Wang, and A. Vasilakos, "CoDaaS: An experimental cloud-centric content delivery platform for user-generated contents," in *Proc. Int. Conf. Comput. Netw. Commun.*, 2012, pp. 934–938.

[8] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[9] "Cisco visual networking index: Global mobile data traffic forecast update, 2013–2018," San Jose, CA, USA, White Paper, Feb. 2014.

[10] Facebook Timeline. [Online]. Available: http://newsroom.fb.com/Timeline

[11] "Cisco visual networking index: Forecast and methodology, 2011–2016," San Jose, CA, USA, White Paper, May 2012. [Online]. Available: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf

[12] H. Kim, T. Benson, A. Akella, and N. Feamster, "The evolution of network configuration: A tale of two campuses," in *Proc. ACM SIGCOMM Conf. Internet Meas. Conf.*, 2011, pp. 499–514.

[13] "What's Behind Network Downtime?" Sunnyvale, CA, USA, May 2008, White Paper. [Online]. Available: https://www-935.ibm.com/services/sg/gts/pdf/200249.pdf

[14] H. Xie, Y. Yang, A. Krishnamurthy, Y. Liu, and A. Silberschatz, "P4p: Provider portal for applications," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 351–362, Aug. 2008.

[15] T.-Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari, "Confused, timid, and unstable: Picking a video streaming rate is hard," in *Proc. ACM Conf. Internet Meas. Conf.*, 2012, pp. 225–238.

[16] X. Chen, Z. M. Mao, and J. Van Der Merwe, "ShadowNet: A platform for rapid and safe network evolution," in *Proc. Conf. USENIX Annu. Tech. Conf.*, 2009, p. 3.

[17] R. Perlman, "Rbridges: Transparent routing," in *Proc. 23rd Annu. Joint Conf. IEEE INFOCOM*, 2004, vol. 2, pp. 1211–1218.

[18] R. Perlman, D. Eastlake, III, S. Gai, D. Dutt, and A. Ghanwani, Routing bridges (RBridges): Base Protocol Specification, Jul. 2011, RFC 6325. [Online]. Available: http://tools.ietf.org/rfc/rfc6325.txt

[19] J. Pan, S. Paul, and R. Jain, "A survey of the research on future Internet architectures," *IEEE Commun. Mag.*, vol. 49, no. 7, pp. 26–36, Jul. 2011.

[20] L. Zhang *et al.*, "Named Data Networking (ndn) project," Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC, Palo Alto, CA, USA, 2010.

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

[22] L. Popa, A. Ghodsi, and I. Stoica, "HTTP as the narrow waist of the future internet," in *Proc. 9th ACM SIGCOMM Workshop Hotnets-IX*, 2010, pp. 6:1–6:6.

[23] "Software-defined networking: The new norm for networks," Palo Alto, CA, USA, White Paper, Apr. 2012. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/white-papers/wp-sdn-newnorm.pdf

[24] T. Nadeau and P. Pan, Software Driven Networks Problem Statement, Oct. 2011, Internet Draft. [Online]. Available: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf

[25] Open Networking Summit. [Online]. Available: http://www.opennetsummit.org/archives/oct11/site/tutorials.html

[26] Hot Topics in Software Defined Networking (HotSDN). [Online]. Available: http://conferences.sigcomm.org/sigcomm/2012/hotsdn.php

[27] European Workshop on Software Defined Networks. [Online]. Available: http://www.ewsdn.eu/previous/ewsdn12.html

[28] "Software defined networking: A new paradigm for virtual, dynamic, flexible networking," Hopewell Junction, NY, USA, White Paper, Oct. 2012. [Online]. Available: http://ict.unimap.edu.my/images/doc/SDN%20IBM%20WhitePaper.pdf

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

[30] J. Smith *et al.*, "Switchware: Accelerating network evolution," CIS Dept., Univ. Pennsylvania, Philadelphia, PA, USA, White Paper, 1996.

[31] D. Alexander *et al.*, "The SwitchWare active network architecture," *IEEE Netw.*, vol. 12, no. 3, pp. 29–36, May/Jun. 1998.

[32] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. Kaashoek, "The click modular router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, Aug. 2000.

[33] M. Handley, O. Hodson, and E. Kohler, "XORP: An open platform for network research," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 53–57, Jan. 2003.

[34] Quagga Routing Software Suite. [Online]. Available: http://www.nongnu.org/quagga/

[35] The BIRD Internet Routing Daemon. [Online]. Available: http://bird.network.cz/

[36] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. van der Merwe, "The case for separating routing from routers," in *Proc. ACM SIGCOMM Workshop FDNA*, 2004, pp. 5–12.

[37] L. Yang, R. Dantu, T. Anderson, and R. Gopal, Forwarding and Control Element Separation (ForCES) Framework, Apr. 2004, RFC 3746. [Online]. Available: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf

[38] T. Lakshman, T. Nandagopal, R. Ramjee, K. Sabnani, and T. Woo, "The softrouter architecture," in *Proc. ACM SIGCOMM Workshop Hot Topics Netw.*, 2004, pp. 1–6.

[39] W. Wang *et al.*, "Design and implementation of an open programmable router compliant to IETF ForCES specifications," in *Proc. 6th ICN*, 2007, pp. 1–6.

[40] A. Doria *et al.*, Forwarding and Control Element Separation (ForCES) Protocol Specification, Mar. 2010, RFC 5810. [Online]. Available: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf

[41] J. Rexford *et al.*, "Network-wide decision making: Toward a wafer-thin control plane," in *Proc. HotNets*, 2004, pp. 59–64.

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

[43] H. Yan *et al.*, "Tesseract: A 4D network control plane," in *Proc. 4th USENIX Conf. NSDI*, 2007, p. 27.

[44] A. Farrel, J.-P. Vasseur, and J. Ash, A Path Computation Element (PCE)-Based Architecture, Aug. 2006, RFC 4655. [Online]. Available: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf

[45] M. Casado *et al.*, "SANE: A protection architecture for enterprise networks," in *Proc. 15th Conf. USENIX-SS*, Berkeley, CA, USA, 2006, vol. 15, p. 10.

[46] J. Luo, J. Pettit, M. Casado, J. Lockwood, and N. McKeown, "Prototyping Fast, Simple, Secure Switches for Etha," in *Proc. 15th Annu. IEEE Symp. HOTI*, 2007, pp. 73–82.

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

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

[49] S. Shenker, M. Casado, T. Koponen, and N. McKeown, "The future of networking, and the past of protocols," presented at the Open Networking Summit, Stanford, CA, USA, 2011. [Online]. Available: http://www.opennetsummit.org/archives/apr12/site/talks/shenker-tue.pdf

[50] A. Gember, P. Prabhu, Z. Ghadiyali, and A. Akella, "Toward software-defined middlebox networking," in *Proc. 11th ACM Workshop Hot Topics Netw.*, 2012, pp. 7–12.

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

[52] M. Ghobadi, S. Yeganeh, and Y. Ganjali, "Rethinking end-to-end congestion control in software-defined networks," in *Proc. 11th ACM Workshop Hot Topics Netw.*, 2012, pp. 61–66.

[53] N. Handigol, S. Seetharaman, M. Flajslik, R. Johari, and N. McKeown, "Aster* x: Load-balancing as a network primitive," in *Proc. 9th GENI Eng. Conf. (Plenary)*, 2010, pp. 1–2.

[54] B. Heller *et al.*, "ElasticTree: Saving energy in data center networks," in *Proc. 7th USENIX Conf. NSDI*, 2010, p. 17.

[55] A. Ferguson, A. Guha, J. Place, R. Fonseca, and S. Krishnamurthi, "Participatory networking," in *Proc. Hot-ICE*, San Jose, CA, USA, 2012, p. 2.

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

[57] T. Koponen *et al.*, "Architecting for innovation," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 3, pp. 24–36, Jul. 2011.

[58] PlanetLab. [Online]. Available: http://www.planet-lab.org/

[59] Global Environment for Network Innovations (GENI). [Online]. Available: http://www.geni.net/

[60] A. Sharafat, S. Das, G. Parulkar, and N. McKeown, "MPLS-TE and MPLS VPNS with OpenFlow," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 452–453, Aug. 2011.

[61] N. Blefari-Melazzi, A. Detti *et al.*, "An OpenFlow-based testbed for information centric networking," in *Proc. Future Netw. Mobile Summit*, 2012, pp. 4–6.

[62] H. Yin *et al.*, SDNi: A Message Exchange Protocol for Software Defined Networks (SDNS) across Multiple Domains, Jun. 2012, Internet draft. [Online]. Available: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf

[63] H. Xie *et al.*, "Software-defined networking efforts debuted at IETF 84," *IETF J.*, Oct. 2012. [Online]. Available: http://www.internetsociety.org/fr/node/45708

[64] R. Birke *et al.*, "Partition/aggregate in commodity 10G ethernet software-defined networking," in *Proc. IEEE 13th Int. Conf. HPSR*, 2012, pp. 7–14.

[65] E. Karpilovsky, M. Caesar, J. Rexford, A. Shaikh, and J. van der Merwe, "Practical network-wide compression of IP routing tables," *IEEE Trans. Netw. Serv. Manage.*, vol. 9, no. 4, pp. 446–458, Dec. 2012.

[66] T. Pan, X. Guo, C. Zhang, W. Meng, and B. Liu, "ALFE: A replacement policy to cache elephant flows in the presence of mice flooding," in *Proc. IEEE ICC*, Jun. 2012, pp. 2961–2965.

[67] O. Ferkouss *et al.*, "A 100 Gig network processor platform for OpenFlow," in *Proc. 7th Int. CNSM*, 2011, pp. 1–4.

[68] J. C. Mogul and P. Congdon, "Hey, you darned counters!: Get off my ASIC!" in *Proc. 1st Workshop HotSDN*, 2012, pp. 25–30.

[69] V. Tanyingyong, M. Hidell, and P. Sjodin, "Improving PC-based OpenFlow switching performance," in *Proc. ACM/IEEE Symp. ANCS*, 2010, pp. 1–2.

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

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

[72] Pantou: OpenFlow 1.0 for OpenWRT. [Online]. Available: http://www.openflow.org/wk/index.php/Pantou_:_OpenFlow_1.0_for_OpenWRT

[73] Y. Mundada, R. Sherwood, and N. Feamster, "An OpenFlow switch element for click," presented at the Symposium Click Modular Router, Ghent, Belgium, 2009.

[74] B. Pfaff *et al.*, "Extending networking into the virtualization layer," in *Proc. HotNets*, New York, NY, USA, 2009. [Online]. Available: http://conferences.sigcomm.org/hotnets/2009/papers/hotnets2009-final143.pdf

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

[76] J. Lockwood *et al.*, "NetFPGA-an open platform for gigabit-rate network switching and routing," in *Proc. IEEE Int. Conf. MSE*, 2007, pp. 160–161.

[77] M. Anwer, M. Motiwala, M. Tariq, and N. Feamster, "SwitchBlade: A platform for rapid deployment of network protocols on programmable hardware," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 183–194, Oct. 2010.

[78] G. Lu *et al.*, "Serverswitch: A programmable and high performance platform for data center networks," in *Proc. NSDI*, 2011, p. 2.

[79] A. Rostami, T. Jungel, A. Koepsel, H. Woesner, and A. Wolisz, "ORAN: OpenFlow routers for academic networks," in *Proc. IEEE 13th Int. Conf. HPSR*, 2012, pp. 216–222.

[80] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown, "Implementing an OpenFlow switch on the NetFPGA platform," in *Proc. 4th ACM/IEEE Symp. ANCS*, 2008, pp. 1–9.

[81] J. Kempf *et al.*, "OpenFlow MPLS and the open source label switched router," in *Proc. 23rd ITC*, 2011, pp. 8–14.

[82] Indigo—Open Source OpenFlow Switches. [Online]. Available: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf

[83] A. Bianco, R. Birke, L. Giraudo, and M. Palacin, "Openflow switching: Data plane performance," in *Proc. IEEE ICC*, 2010, pp. 1–5.

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

[85] T. Ulversoy, "Software defined radio: Challenges and opportunities," *IEEE Commun. Surveys Tuts.*, vol. 12, no. 4, pp. 531–550, May 2010.

[86] E. Mannie, Generalized Multi-Protocol Label Switching (GMPLS) Architecture, Oct. 2004, RFC 3945. [Online]. Available: http://tools.ietf.org/rfc/rfc6325.txt

[87] M. Bansal, J. Mehlman, S. Katti, and P. Levis, "OpenRadio: A programmable wireless dataplane," in *Proc. 1st Workshop HotSDN*, 2012, pp. 109–114.

[88] R. Murty, J. Padhye, A. Wolman, and M. Welsh, "Dyson: An architecture for extensible wireless LANs," in *Proc. USENIXATC*, 2010, p. 15.

[89] Joint Program Executive Office (JPEO) for the Joint Tactical Radio System (JTRS), Software Communications Architecture Specification, San Diego, CA, USA 2012. [Online]. Available: http://jpeojtrs.mil/sca/Documents/SCAv4_0/SCA_4.0_20120228_ScaSpecification.pdf

[90] G. Jianxin, Y. Xiaohui, G. Jun, and L. Quan, "The software communication architecture specification: Evolution and trends," in *Proc. PACIIA*, 2009, vol. 2, pp. 341–344.

[91] Wireless Innovation Forum (Previously the SDRForum). [Online]. Available: http://www.wirelessinnovation.org/

[92] J. Elbers and A. Autenrieth, "From static to software-defined optical networks," in *Proc. 16th Int. Conf. ONDM*, 2012, pp. 1–4.

[93] A. Giorgetti, F. Cugini, F. Paolucci, and P. Castoldi, "OpenFlow and PCE Architectures in Wavelength Switched Optical Networks," in *Proc. 16th Int. Conf. ONDM*, 2012, pp. 1–6.

[94] S. Das, G. Parulkar, and N. McKeown, "Simple unified control for packet and circuit networks," in *Proc. IEEE/LEOSST Meet.*, 2009, pp. 147–148.

[95] S. Das, G. Parulkar, and N. McKeown, "Unifying packet and circuit switched networks," in *Proc. IEEE GLOBECOM Workshops*, 2009, pp. 1–6.

[96] S. Das *et al.*, "Packet and circuit network convergence with OpenFlow," in *Pro. OFC/NFOEC*, 2010, pp. 1–3.

[97] S. Das *et al.*, "Application-aware aggregation and traffic engineering in a converged packet-circuit network," in *Proc. OFC/NFOEC*, 2011, pp. 1–3.

[98] L. Liu, T. Tsuritani, I. Morita, H. Guo, and J. Wu, "OpenFlow-based wavelength path control in transparent optical networks: A proof-of-concept demonstration," in *Proc. 37th ECOC*, 2011, pp. 1–3.

[99] 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. Exp.*, vol. 19, no. 27, pp. 26 578–26 593, Dec. 2011.

[100] L. Liu *et al.*, "First field trial of an OpenFlow-based unified control plane for multi-layer multi-granularity optical networks," presented at the Optical Fiber Communication Conference, Los Angeles, CA, USA, 2012, Paper PDP5D.2.

[101] L. Liu *et al.*, "First field trial of an OpenFlow-based unified control plane for multi-layer multi-granularity optical networks," in *Proc. OFC/NFOEC*, 2012, pp. 1–3.

[102] L. Liu, T. Tsuritani, and I. Morita, "Experimental demonstration of OpenFlow/GMPLS interworking control plane for IP/DWDM multi-layer optical networks," in *Proc. 14th ICTON*, 2012, pp. 1–4.

[103] L. Liu, T. Tsuritani, and I. Morita, "From GMPLS to PCE/GMPLS to OpenFlow: How much benefit can we get from the technical evolution of control plane in optical networks?" in *Proc. 14th ICTON*, 2012, pp. 1–4.

[104] F. Farias, J. Salvatti, E. Cerqueira, and A. Abelem, "A proposal management of the legacy network environment using OpenFlow control plane," in *Proc. IEEE NOMS*, 2012, pp. 1143–1150.

[105] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian, "Fabric: A retrospective on evolving SDN," in *Proc. 1st Workshop HotSDN*, 2012, pp. 85–90.

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

[107] V. Gudla *et al.*, "Experimental demonstration of OpenFlow control of packet and circuit switches," presented at the Optical Fiber Communication Conference, San Diego, CA, USA, 2010, Paper OTuG2.

[108] D. Simeonidou, R. Nejabati, and S. Azodolmolky, "Enabling the future optical Internet with OpenFlow: A paradigm shift in providing intelligent optical network services," in *Proc. 13th ICTON*, 2011, pp. 1–4.

[109] S. Azodolmolky *et al.*, "Integrated OpenFlow-GMPLS control plane: An overlay model for software defined packet over optical networks," *Opt. Exp.*, vol. 19, no. 26, pp. B421–B428, Dec. 2011.

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

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

[112] N. Foster *et al.*, "Frenetic: A high-level language for OpenFlow networks," in *Proc. Workshop PRESTO*, 2010, pp. 6:1–6:6.

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

[114] A. Voellmy and P. Hudak, "Nettle: A language for configuring routing networks," in *Proc. IFIP TC 2 Working Conf. DSL*, 2009, pp. 211–235.

[115] A. Voellmy *et al.*, "Don't configure the network, program it! Domain-specific programming languages for network systems," Defense Tech. Inf. Center, Fort Belvoir, VA, USA, DTIC Doc. Tech. Rep., 2010.

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

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

[118] T. Hinrichs, N. Gude, M. Casado, J. Mitchell, and S. Shenker, "Expressing and enforcing flow-based network security policies," Univ. Chicago, Chicago, IL, USA, Tech. Rep, 2008.

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

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

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

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

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

[124] R. Wang, D. Butnariu, and J. Rexford, "OpenFlow-based server load balancing gone wild," in *Proc. 11th USENIX Conf. Hot-ICE Netw. Serv.*, 2011, p. 12.

[125] S. Ghorbani and M. Caesar, "Walk the line: Consistent network updates with bandwidth guarantees," in *Proc. 1st Workshop HotSDN*, 2012, pp. 67–72.

[126] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *Proc. ACM SIGCOMM Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2012, pp. 323–334.

[127] M. Reitblatt, N. Foster, J. Rexford, and D. Walker, "Consistent updates for software-defined networks: Change you can believe in!" in *Proc. 10th ACM Workshop HotNets-X*, 2011, pp. 7:1–7:6.

[128] R. McGeer, "A safe, efficient update protocol for OpenFlow networks," in *Proc. 1st Workshop HotSDN*, 2012, pp. 61–66.

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

[130] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot, "Traffic matrix estimation: Existing techniques and new directions," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, pp. 161–174, Oct. 2002.

[131] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: Traffic matrix estimator for OpenFlow networks," in *Passive and Active Measurement*. Berlin, Germany: Springer-Verlag, 2010, pp. 201–210.

[132] A. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *Proc. IEEE INFOCOM*, 2011, pp. 1629–1637.

[133] L. Jose, M. Yu, and J. Rexford, "Online measurement of large traffic aggregates on commodity switches," in *Proc. 11th USENIX Conf. Hot-ICE Netw. Serv.*, 2011, p. 13.

[134] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," in *Proc. 28th Annu. ACM STOC*, 1996, pp. 20–29.

[135] G. Cormode and M. Hadjieleftheriou, "Finding frequent items in data streams," *Proc. VLDB Endow.*, vol. 1, no. 2, pp. 1530–1541, Aug. 2008.

[136] A. Kumar, M. Sung, J. J. Xu, and J. Wang, "Data streaming algorithms for efficient and accurate estimation of flow size distribution," *SIGMETRICS Perform. Eval. Rev.*, vol. 32, no. 1, pp. 177–188, Jun. 2004.

[137] A. Kumar, M. Sung, J. J. Xu, and J. Wang, "Data streaming algorithms for efficient and accurate estimation of flow size distribution," in *Proc. Joint SIGMETRICS Int. Conf. Meas. Model. Comput. Syst.*, 2004, pp. 177–188.

[138] C. Estan, G. Varghese, and M. Fisk, "Bitmap algorithms for counting active flows on high-speed links," *IEEE/ACM Trans. Netw.*, vol. 14, no. 5, pp. 925–937, Oct. 2006.

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

[140] P. Fonseca, R. Bennesby, E. Mota, and A. Passito, "A replication component for resilient OpenFlow-based networking," in *Proc. IEEE NOMS*, 2012, pp. 933–939.

[141] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized?: State distribution trade-offs in software defined networks," in *Proc. 1st Workshop HotSDN*, 2012, pp. 1–6.

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

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

[144] P. Porras *et al.*, "A security enforcement kernel for OpenFlow networks," in *Proc. 1st Workshop HotSDN*, 2012, pp. 121–126.

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

[146] E. Al-Shaer, W. Marrero, A. El-Atawy, and K. ElBadawi, "Network configuration in a box: Towards end-to-end verification of network reachability and security," in *Proc. 17th IEEE ICNP*, 2009, pp. 123–132.

[147] P. Perešíni and M. Canini, "Is your OpenFlow application correct?" in *Proc. ACM CoNEXT Student Workshop*, 2011, pp. 18:1–18:2.

[148] M. Canini, D. Kostic, J. Rexford, and D. Venzano, "Automating the testing of OpenFlow applications," presented at the 1st International Workshop Rigorous Protocol Engineering, Portland, OR, USA, 2011.

[149] M. Canini, D. Venzano, P. Perešíni, D. Kostić, and J. Rexford, "A nice way to test OpenFlow applications," in *Proc. 9th USENIX Conf. NSDI*, 2012, p. 10.

[150] M. Kuźniar, M. Canini, and D. Kostić, "OFTEN testing OpenFlow networks," in *Proc. 1st EWSDN*, Oct. 2012, pp. 54–60.

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

[152] A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey, "VeriFlow: Verifying network-wide invariants in real time," in *Proc. 1st Workshops HotSDN*, 2012, pp. 49–54, New York, NY, USA.
[153] A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey, "Veriflow: Verifying network-wide invariants in real time," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 467–472, Sep. 2012.
[154] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey, "VeriFlow: Verifying network-wide invariants in real time," in *Proc. 10th USENIX Conf. NSDI*, 2013, pp. 15–28.
[155] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proc. 1st Workshop HotSDN*, 2012, pp. 7–12.
[156] Z. Cai, "Maestro: Achieving scalability and coordination in centralized network control plane," Ph.D. dissertation, Rice Univ., Houston, TX, USA, 2011.
[157] Z. Cai, A. L. Cox, and T. E. Ng, "Maestro: A system for scalable OpenFlow control," Rice Univ., Houston, TX, USA, Tech. Rep. TR 10-08, Dec. 2010.
[158] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proc. 2nd USENIX Conf. Hot-ICE Netw. Serv.*, 2012, p. 10.
[159] A. Voellmy and J. Wang, "Scalable software defined network controllers," in *Proc. ACM SIGCOMM Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2012, pp. 289–290.
[160] A. Voellmy, B. Ford, P. Hudak, and Y. R. Yang, "Scaling software-defined network controllers on multicore servers," Comput. Sci., Yale Univ., New Haven, CT, USA, Yale CS TR 1468, 2012.
[161] Beacon. [Online]. Available: https://openflow.stanford.edu/display/Beacon/Home
[162] M. Jarschel *et al.*, "Modeling and performance evaluation of an OpenFlow architecture," in *Proc. 23rd ITCP*, 2011, pp. 1–7.
[163] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," in *Proc. ACM SIGCOMM*, 2010, pp. 351–362.
[164] A. R. Curtis *et al.*, "DevoFlow: Scaling flow management for high-performance networks," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 254–265, Aug. 2011.
[165] S. H. Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proc. 1st Workshop HotSDN*, 2012, pp. 19–24.
[166] Cbench (Controller Benchmarker). [Online]. Available: http://www.openflow.org/wk/index.php/Oflops
[167] M. Jarschel, F. Lehrieder, Z. Magyari, and R. Pries, "A flexible OpenFlow-controller benchmark," in *Proc. EWSDN*, 2012, pp. 48–53.
[168] R. Alimi, R. Penno, and Y. Yang, ALTO Protocol, Feb. 2013, Internet Draft. [Online]. Available: http://tools.ietf.org/id/draft-ietf-alto-protocol-14.txt
[169] V. Gurbani, M. Scharf, T. Lakshman, V. Hilt, and E. Marocco, "Abstracting network state in Software Defined Networks (SDN) for rendezvous services," in *Proc. IEEE ICC*, 2012, pp. 6627–6632.
[170] E. Kissel, G. Fernandes, M. Jaffee, M. Swany, and M. Zhang, "Driving software defined networks with XSP," in *Proc. Workshop SDN/IEEE Int. Conf. Commun.*, 2012, pp. 6616–6621.
[171] E. Keller and J. Rexford, "The "Platform as a service" model for networking," in *Proc. INM/WREN*, 2010, p. 4.
[172] N. Handigol *et al.*, "Plug-n-Serve: Load-balancing web traffic using OpenFlow," in *Proc. ACM SIGCOMM Demo*, Barcelona, Spain, 2009. [Online]. Available: http://conferences.sigcomm.org/sigcomm/2009/demos/sigcomm-pd-2009-final26.pdf
[173] K.-K. Yap *et al.*, "Blueprint for introducing innovation into wireless mobile networks," in *Proc. 2nd ACM SIGCOMM Workshop VISA*, 2010, pp. 25–32.
[174] K.-K. Yap *et al.*, "OpenRoads: Empowering research in mobile networks," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 125–126, Jan. 2010.
[175] K. Yap, S. Katti, G. Parulkar, and N. McKeown, "Delivering capacity for the mobile internet by stitching together networks," in *Proc. ACM Workshop Wireless Students*, 2010, pp. 41–44.
[176] L. Suresh, J. Schulz-Zander, R. Merz, A. Feldmann, and T. Vazao, "Towards programmable enterprise WLANS with Odin," in *Proc. 1st Workshop HotSDN*, 2012, pp. 115–120.
[177] A. Wundsam, D. Levin, S. Seetharaman, and A. Feldmann, "OFRewind: Enabling record and replay troubleshooting for networks," in *Proc. USENIX Annu. Tech. Conf.*, 2011, p. 29.
[178] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "OpenFlow random host mutation: Transparent moving target defense using software defined networking," in *Proc. 1st Workshop HotSDN*, 2012, pp. 127–132.
[179] M. Mendonca, S. Seetharaman, and K. Obraczka, "A flexible in-network IP anonymization service," in *Proc. IEEE ICC*, 2012, pp. 6651–6656.
[180] J. Fu, P. Sjödin, and G. Karlsson, "Intra-domain routing convergence with centralized control," *Comput. Netw.*, vol. 53, no. 18, pp. 2985–2996, Dec. 2009.
[181] K. K. Lakshminarayanan, I. Stoica, S. Shenker, and J. Rexford, "Routing as a service," EECS Dept., Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2006-19, Feb. 2006.
[182] P. Pisa *et al.*, "OpenFlow and Xen-based virtual network migration," in *Communications: Wireless in Developing Countries and Networks of the Future*. Berlin, Germany: Springer-Verlag, 2010, pp. 170–181.
[183] M. Koerner and O. Kao, "Multiple service load-balancing with OpenFlow," in *Proc. 13th IEEE Int. Conf. HPSR*, 2012, pp. 210–214.
[184] G. Wang, T. E. Ng, and A. Shaikh, "Programming your network at run-time for big data applications," in *Proc. 1st Workshop HotSDN*, 2012, pp. 103–108.
[185] Z. Kerravala, "As the value of enterprise networks escalates, so does the need for configuration management," Enterprise Computing & Networking, The Yankee Group Report, Boston, MA, USA, 2004.
[186] N. Handigol, B. Heller, V. Jeyakumar, D. Maziéres, and N. McKeown, "Where is the debugger for my software-defined network?" in *Proc. 1st Workshop HotSDN*, 2012, pp. 55–60.
[187] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Enabling fast failure recovery in OpenFlow networks," in *Proc. 8th Int. Workshop DRCN*, 2011, pp. 164–171.
[188] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS flooding attack detection using NOX/OpenFlow," in *Proc. 35th IEEE Conf. LCN*, 2010, pp. 408–415.
[189] G. Gibb, H. Zeng, and N. McKeown, "Outsourcing network functionality," in *Proc. 1st Workshop HotSDN*, 2012, pp. 73–78.
[190] G. Huang, C. Chuah, S. Raza, and S. Seetharaman, "Dynamic measurement-aware routing in practice," *IEEE Netw.*, vol. 25, no. 3, pp. 29–34, May/Jun. 2011.
[191] J. Naous, R. Stutsman, D. Mazieres, N. McKeown, and N. Zeldovich, "Delegating network security with more information," in *Proc. 1st ACM WREN*, 2009, pp. 19–26.
[192] C. Rigney, A. Rubens, W. Simpson, and S. Willens, Remote Authentication Dial in User Service (RADIUS), Jun. 2000. [Online]. Available: http://tools.ietf.org/rfc/rfc2865.txt
[193] Y. Yamasaki, Y. Miyamoto, J. Yamato, H. Goto, and H. Sone, "Flexible access management system for campus VLAN based on OpenFlow," in *Proc. IEEE/IPSJ 11th Int. SAINT*, 2011, pp. 347–351.
[194] S. Kinoshita, T. Watanabe, J. Yamato, H. Goto, and H. Sone, "Implementation and evaluation of an OpenFlow-based access control system for wireless LAN roaming," in *Proc. 36th Annu. COMPSACW*, 2012, pp. 82–87.
[195] A. Ramachandran, Y. Mundada, M. Tariq, and N. Feamster, "Securing enterprise networks using traffic tainting," Georgia Inst. Technol., Atlanta, GA, USA, Tech. Rep. GTCS-09-15, 2009.
[196] N. Feamster *et al.*, "Decoupling policy from configuration in campus and enterprise networks," in *Proc. 17th IEEE Workshop LANMAN*, 2010, pp. 1–6.
[197] A. Nayak, A. Reimers, N. Feamster, and R. Clark, "Resonance: Dynamic access control for enterprise networks," in *Proc. 1st ACM Workshop Res. Enterprise Netw.*, 2009, pp. 11–18.
[198] N. Chowdhury and R. Boutaba, "Network virtualization: State of the art and research challenges," *IEEE Commun. Mag.*, vol. 47, no. 7, pp. 20–26, Jul. 2009.
[199] D. Turull, M. Hidell, and P. Sjodin, "Using libNetVirt to control the virtual network," in *Proc. IEEE Int. Conf. CLOUDNET*, 2012, pp. 148–152.
[200] D. Turull, M. Hidell, and P. Sjodin, "libNetVirt: The network virtualization library," in *Proc. IEEE ICC*, 2012, pp. 5543–5547.
[201] R. Sherwood *et al.*, "Flowvisor: A network virtualization layer," OpenFlow Switch Consortium, OPENFLOW-TR-2009-1, 2009. [Online]. Available: http://archive.openflow.org/downloads/technicalreports/openflow-tr-2009-1-flowvisor.pdf
[202] R. Sherwood *et al.*, "Can the production network be the testbed?" in *Proc. 9th USENIX Conf. OSDI*, 2010, pp. 1–6.
[203] R. Sherwood *et al.*, "Carving research slices out of your production networks with OpenFlow," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 129–130, Jan. 2010.
[204] Y. Yiakoumis, K.-K. Yap, S. Katti, G. Parulkar, and N. McKeown, "Slicing home networks," in *Proc. 2nd ACM SIGCOMM Workshop HomeNets*, 2011, pp. 1–6.

[205] A. Bianzino, C. Chaudet, D. Rossi, and J. Rougier, "A survey of green networking research," *IEEE Commun. Surveys Tuts.*, vol. 14, no. 1, pp. 3–20, Dec. 2012.

[206] D. Staessens, S. Sharma, D. Colle, M. Pickavet, and P. Demeester, "Software defined networking: Meeting carrier grade requirements," in *Proc. 18th IEEE Workshop LANMAN*, 2011, pp. 1–6.

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

[208] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker, "Applying NOX to the datacenter," in *Proc. HotNets*, New York, NY, USA, 2009. [Online]. Available: http://conferences.sigcomm.org/hotnets/2009/papers/hotnets2009-final103.pdf

[209] M. Moshref, M. Yu, A. Sharma, and R. Govindan, "Scalable rule management for data centers," in *Proc. 10th USENIX Conf. NSDI*, 2013, pp. 157–170.

[210] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: State-of-the-art and research challenges," *J. Internet Serv. Appl.*, vol. 1, no. 1, pp. 7–18, May 2010.

[211] Q. Li, J. Huai, J. Li, T. Wo, and M. Wen, "HyperMIP: Hypervisor controlled mobile IP for virtual machine live migration across networks," in *Proc. 11th IEEE HASE Symp.*, 2008, pp. 80–88.

[212] P. Raad *et al.*, "Achieving sub-second downtimes in internet-wide virtual machine live migrations in LISP networks," in *Proc. IFIP/IEEE Int. Symp. IM*, 2013, pp. 286–293.

[213] M. Coudron, S. Secci, G. Maier, G. Pujolle, and A. Pattavina, "Boosting cloud communications through a crosslayer multipath protocol architecture," in *Proc. IEEE SDN4FNS*, Nov. 2013, pp. 1–8.

[214] OpenDaylight. [Online]. Available: http://www.opendaylight.org/

[215] F. Hao, T. Lakshman, S. Mukherjee, and H. Song, "Enhancing dynamic cloud-based services using network virtualization," in *Proc. 1st ACM Workshop Virtualized Infrastruct. Syst. Archit.*, 2009, pp. 37–44.

[216] B. Boughzala, R. Ben Ali, M. Lemay, Y. Lemieux, and O. Cherkaoui, "OpenFlow supporting inter-domain virtual machine migration," in *Proc. 8th Int. Conf. WOCN*, 2011, pp. 1–7.

[217] J. Matias, E. Jacob, D. Sanchez, and Y. Demchenko, "An OpenFlow based network virtualization framework for the cloud," in *Proc. 3rd Int. Conf. CloudComp Technol. Sci.*, 2011, pp. 672–678.

[218] V. Mann, A. Vishnoi, K. Kannan, and S. Kalyanaraman, "CrossRoads: Seamless VM mobility across data centers through software defined networking," in *Proc. IEEE NOMS*, 2012, pp. 88–96.

[219] Y. Pu, Y. Deng, and A. Nakao, "Cloud rack: Enhanced virtual topology migration approach with open vswitch," in *Proc. ICOIN*, 2011, pp. 160–164.

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

[221] H. Qian, X. Huang, and C. Chen, "Swan: End-to-end orchestration for cloud network and wan," in *Proc. IEEE 2nd Int. Conf. CloudNet*, Nov. 2013, pp. 236–242.

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

[223] OpenFlow Switch Consortium. [Online]. Available: http://www.openflow.org/

[224] D. Mattos *et al.*, "OMNI: OpenFlow management infrastructure," in *Proc. Int. Conf. NOF*, 2011, pp. 52–56.

[225] Trema. [Online]. Available: http://trema.github.com/trema/

[226] Ryu. [Online]. Available: http://osrg.github.com/ryu/

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

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

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

[230] K.-K. Yap *et al.*, "The Stanford OpenRoads deployment," in *Proc. 4th ACM Int. WINTECH*, 2009, pp. 59–66.

[231] S. Jain *et al.*, "4: Experience with a globally-deployed software defined WAN," in *Proc. ACM SIGCOMM Conf.*, 2013, pp. 3–14.

[232] N. Feamster and J. Rexford, "Getting students' hands dirty with clean-slate networking," in *Proc. SIGCOMM Educ. Workshop*, Toronto, ON, Canada, 2011. [Online]. Available: http://edusigcomm.info.ucl.ac.be/pmwiki/uploads/Workshop2011/20110430002/clean-slate.pdf

[233] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A blueprint for introducing disruptive technology into the Internet," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 59–64, Jan. 2003.

[234] Federated E-Infrastructure Dedicated to European Researchers Innovating in Computing Network Architectures (FEDERICA). [Online]. Available: http://www.fp7-federica.eu/

[235] JGN-X (JGN-eXtreme). [Online]. Available: http://www.jgn.nict.go.jp/english/index.html

[236] Future Internet Testbeds Experimentation Between Brazil and Europe (FIBRE). [Online]. Available: http://www.fibre-ict.eu/

[237] R. Riggio, T. Rasheed, and F. Granelli, "Empower: A testbed for network function virtualization research and experimentation," in *Proc. IEEE SDN4FNS*, Nov. 2013, pp. 1–5.

[238] U. Holzle, OpenFlow @ Google, Apr. 2012. [Online]. Available: http://www.opennetsummit.org/archives/apr12/hoelzle-tue-openflow.pdf

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

[240] L. Dong, F. Jia, and W. Wang, "Definition and implementation of logical function blocks compliant to ForCES specification," in *Proc. 15th ICON*, 2007, pp. 531–536.

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

[242] E. Haleplidis, S. Denazis, O. Koufopavlou, J. Salim, and J. Halpern, "Software-defined networking: Experimenting with the control to forwarding plane interface," in *Proc. EWSDN*, 2012, pp. 91–96.

[243] P. Lin, J. Bi, and H. Hu, "ASIC: An architecture for scalable intra-domain control in OpenFlow," in *Proc. 7th Int. Conf. Future Internet Technol.*, 2012, pp. 21–26.

[244] M. R. Nascimento, C. E. Rothenberg, M. R. Salvador, and M. F. Magalhães, "QuagFlow: Partnering Quagga with OpenFlow," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 441–442, Aug. 2010.

[245] M. R. Nascimento *et al.*, "Virtual routers as a service: The RouteFlow approach leveraging software-defined networks," in *Proc. 6th Int. CFI Technol.*, 2011, pp. 34–37.

[246] Y. Nakagawa, K. Hyoudou, and T. Shimizu, "A management method of IP multicast in overlay networks using openflow," in *Proc. 1st Workshop HotSDN*, 2012, pp. 91–96.

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

[248] T.-Y. Huang *et al.*, "PhoneNet: A phone-to-phone network for group communication within an administrative domain," in *Proc. 2nd ACM SIGCOMM Workshop Netw., Syst., Appl. MobiHeld*, 2010, pp. 27–32.

[249] B. Koldehofe, F. Dürr, M. A. Tariq, and K. Rothermel, "The power of software-defined networking: Line-rate content-based routing using OpenFlow," in *Proc. 7th Workshop MW4NG Internet Comput.*, 2012, pp. 3:1–3:6.

[250] D. Kotani, K. Suzuki, and H. Shimonishi, "A design and implementation of OpenFlow controller handling IP multicast with fast tree switching," in *Proc. IEE/IPSJ Int. SAINT*, 2012, pp. 60–67.

[251] R. Ravindran, X. Liu, A. Chakraborti, X. Zhang, and G. Wang, "Towards software defined icn based edge-cloud services," in *Proc. IEEE 2nd Int. Conf. CloudNet*, Nov. 2013, pp. 227–235.

[252] T. Li, N. Van Vorst, R. Rong, and J. Liu, "Simulation studies of OpenFlow-based in-network caching strategies," in *Proc. 15th CNS Symp.*, 2012, pp. 12:1–12:7.

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

[254] J. Rubio-Loyola *et al.*, "Scalable service deployment on software-defined networks," *IEEE Commun. Mag.*, vol. 49, no. 12, pp. 84–93, Dec. 2011.

[255] T. Nadeau and P. Pan, Framework for Software Defined Networks, Oct. 2011, Internet Draft. [Online]. Available: http://tools.ietf.org/id/draft-nadeau-sdn-framework-01.txt

[256] ETSI Industry Specification Group for Network Functions Virtualization (ISG NFV), Network Functions Virtualisation, An Introduction, Benefits, Enablers, Challenges & Call for Action, Oct. 2012, White Paper. [Online]. Available: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf

[257] M. Kuzniar, P. Peresini, M. Canini, D. Venzano, and D. Kostic, "A SOFT way for openflow switch interoperability testing," in *Proc. 8th Int. Conf. Emerging Netw. Exp. Technol.*, 2012, pp. 265–276.

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

[259] P. Dely, A. Kassler, and N. Bayer, "Openflow for wireless mesh networks," in *Proc. 20th ICCCN*, 2011, pp. 1–6.

[260] A. Mahmud and R. Rahmani, "Exploitation of OpenFlow in wireless sensor networks," in *Proc. ICCSNT*, 2011, vol. 1, pp. 594–600.
[261] A. Mahmud, R. Rahmani, and T. Kanter, "Deployment of flow-sensors in internet of things' virtualization via OpenFlow," in *Proc. 3rd FTRA Int. Conf. MUSIC*, 2012, pp. 195–200.

Wenfeng Xia received the B.S. degree in computer science from the University of Science and Technology of China, Hefei, China, in 2011. He is currently working toward the M.S. degree at the School of Computer Science and Technology, USTC. He is currently working as a Project Officer with the School of Computer Engineering, Nanyang Technological University, Singapore. His research interests include computer networks and software engineering.

Yonggang Wen (S'99–M'08–SM'14) received the Ph.D. degree in electrical engineering and computer science (minor in western literature) from Massachusetts Institute of Technology (MIT), Cambridge, MA, USA. He has worked in Cisco to lead product development in content delivery networks, which had a revenue impact of $3 billion globally. He has published over 100 papers in top journals and prestigious conferences. He is currently an Assistant Professor with the School Of Computer Engineering, Nanyang Technological University, Singapore. His research interests include cloud computing, green data centers, big data analytics, multimedia networks, and mobile computing. His latest work in multiscreen cloud social televisions has been featured by global media (more than 1600 news articles from over 29 countries) and recognized with ASEAN ICT Award 2013 (Gold Medal) and IEEE Globecom 2013 Best Paper Award. He serves on the editorial boards of IEEE TRANSACTIONS ON MULTIMEDIA, IEEE ACCESS JOURNAL, and *Elsevier Ad Hoc Networks*.

Chuan Heng Foh (S'00–M'03–SM'09) received the M.Sc. degree from Monash University, Clayton, Vic., Australia, in 1999 and the Ph.D. degree from the University of Melbourne, Parkville, Vic., in 2002. After his Ph.D. studies, he spent six months as a Lecturer with Monash University. From 2002 to 2012, he was an Assistant Professor with Nanyang Technological University, Singapore. He is currently a Senior Lecturer with the University of Surrey, Surrey, U.K. He is the author or coauthor of over 100 refereed papers in international journals and conferences. His research interests include protocol design and performance analysis of various computer networks such as wireless local area and mesh networks, and mobile ad hoc and sensor networks, fifth-generation networks, and data center networks. He actively participates in IEEE conference and workshop organization, including the International Workshop on Cloud Computing Systems, Networks, and Applications (CCSNA), where he is a steering member. He is currently an Associate Editor for IEEE ACCESS, IEEE WIRELESS COMMUNICATIONS, and *International Journal of Communications Systems*. He is also the Chair of the Special Interest Group on Green Data Center and Cloud Computing under the IEEE Technical Committee on Green Communications and Computing.

Dusit Niyato received the B.Eng. degree in computer engineering from King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand, in 1999 and the Ph.D. degree in electrical and computer engineering from the University of Manitoba, Winnipeg, MB, Canada, in 2008. He is currently an Associate Professor with the School of Computer Engineering, Nanyang Technological University, Singapore. His research interests include radio resource management in cognitive radio networks and broadband wireless access networks.

Haiyong Xie received the B.S. degree from the University of Science and Technology of China, Hefei, China, in 1997 and the M.S. and Ph.D. degrees in computer science from Yale University, New Haven, CT, USA in 2005 and 2008, respectively. He is currently the Executive Director of the Cyberspace and Data Science Laboratory, Chinese Academy of Electronics and Information Technology, Beijing, China, and a Professor with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, China. He was the Principal Researcher for Huawei U.S. Research Labs and the P4P Working Group (P4PWG) and Distributed Computing Industry Association. He proposed P4P (proactive provider participation in P2P) to coordinate network providers and peer-to-peer applications in a seminal paper published in ACM SIGCOMM 2008, and led and conducted original research and large-scale tests on P4P. Encouraged by and based upon his research and results on P4P, the P4PWG was formed to promote academic studies and industrial adoptions of P4P, which was later adopted by IETF to form a new Application Layer Traffic Optimization (ALTO) Working Group. His research interest includes content-centric networking, software-defined networking, future Internet architecture, and network traffic engineering.