Software defined networking: Technologies and solutions toward an open network eco－system

LIU FangMing (刘方明)[1]*, GUO Jian[1] & LIU JiangChuan[2]

[1] *School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China*;

[2] *Simon Fraser University, Canada*

Received January 16, 2017; accepted February 15, 2017

Abstract Software-defined networking (SDN), a new networking paradigm decoupling the software control logic from the data forwarding hardware, promises to enable simpler management, more flexible resource usage and faster deployment of network services. It opens network functionality, application programmability, and control－to－data communication interfaces that used to be closed in conventional network devices, offering endless opportunities but also challenges for both existing players and newcomers in the market. Through a comprehensive and comparative exploratory of SDN state－of－the－art techniques, standardization activities and realistic applications, this article unveils historic and technical insights into the innovations that SDN offers toward an emerging open network eco－system. We closely examine the critical challenges and opportunities when the networking industry is reshaped by SDN. We further shed light on future development directions of SDN in broad application scenarios, ranging from cloud datacenters, network operating systems, and advanced wireless networking.

Keywords Software defined networking; Open network eco－system; Cloud computing; Datacenters; Network operating Systems

doi: 10.16262/j.cnki.1005－0841.2017.01.001

1 Introduction

Analogical to the explosive invention and popularization of PCs that separates operating systems (OSes) from the underlying hardware, *Software-Defined Networking* (SDN) enables an open programmable network norm that separates the data plane from the control plane in conventional network devices [1]. In the SDN architecture, the hardware is only responsible for simple packet forwarding while the decision of forwarding is controlled by a logically centralized controller, which enables network programmability through accessible interfaces for network operators (See Fig. 1). With simple and general interfaces, network operators are able to program the control logic, globally monitor network status and enable automatic response to network events.

Openness lies in the heart of SDN, evidently from the brand names of such key emerging SDN associated organizations as *Open Networking Foundation* (ONF) [2] and representative projects as *OpenFlow* [3]. Decoupling the software control logic from data forwarding hardware, the long-run mission of SDN is to build an open networking architecture that gathers together the intelligence from both existing players and startups in the computer networking market toward a new eco-system with endless opportunities. It facilitates software design with open programmable interfaces and standardizes hardware manufacturing with open architectures. For SDN users, the open programmability of SDN can dramatically reduce the

* Corresponding author (Email: fmliu@hust. edu. cn, Web: http://grid. hust. edu. cn/fmliu/)

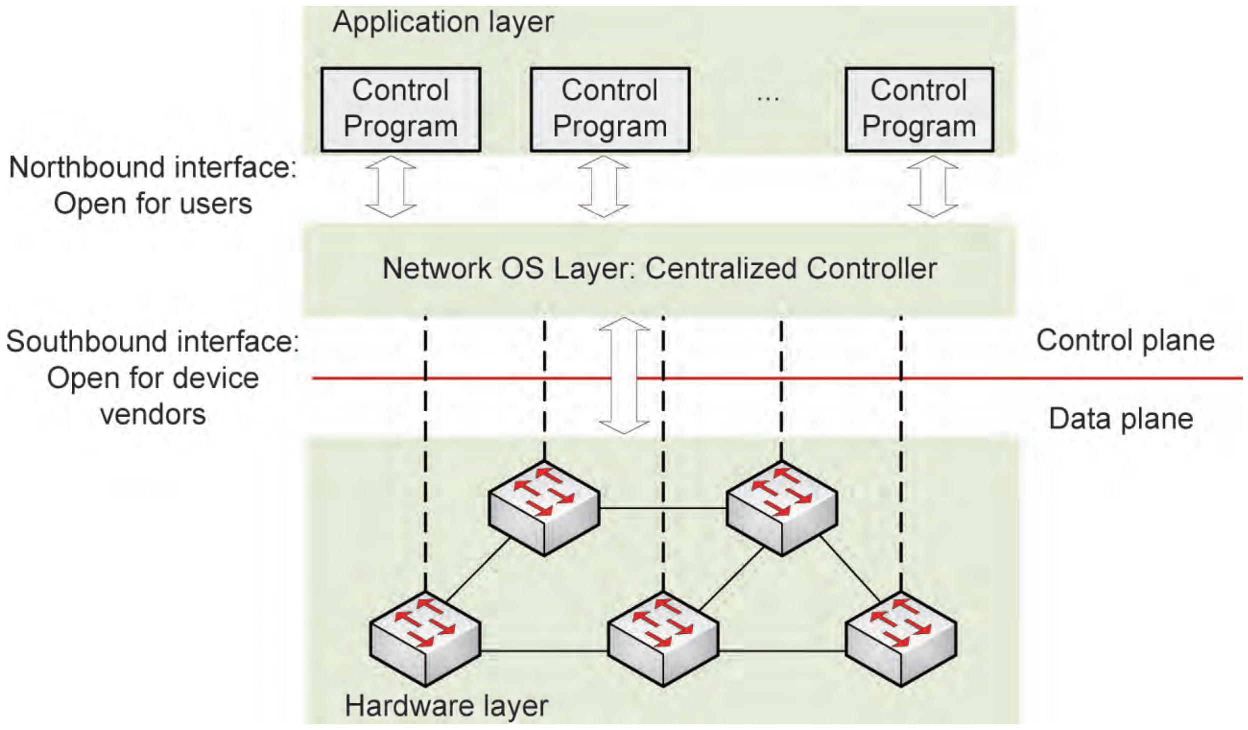

Application layer
Control Program
Control Program
...
Control Program
Northbound interface: Open for users
Network OS Layer: Centralized Controller
Southbound interface: Open for device vendors
Control plane
Data plane
Hardware layer

Figure 1 Abstraction of SDN architecture: three layers and two interfaces.

cost of network development and management where such complex network services as cloud datacenters and enterprise networking are needed. The visibility of packet forwarding also produces chances to design new protocols and reveal networking principles for research and education.

This obviously opens a Pandora's box that has long been controlled by the network hardware manufacturers. For decades, the public network was made as a black box to users by sealing both software and hardware as well as both data and control into network devices, providing limited accessibility and configurability, not to mention programmability. Today, SDN is drawing exploding public attention from both academia and industry. According to an SDN central report [4], the Venture Capital investment in SDN has increased 50 times over the last three years, and a report from the IDC [5] further predicts that the SDN market size will grow to 3.7 billion dollars by 2016. Yet there remain numerous questions to be answered after opening the Pandora's box: Which aspects of the network architecture can be or should be opened by SDN? What kind of open functionality can SDN realize? Which players will benefit from such open opportunities and innovations, or influenced by the challenges and monopolistic competition market? To name but a few.

This article aims to explore the potentials and design spaces of SDN from diverse aspects, seeking answers to these questions and also raising issues toward future research and development. Different from early surveys that focus on programmable networks and their development toward SDN [6—8] or the security challenges in SDN [9], we investigate both technical and industrial impact of SDN from the perspective of an open network eco-system. We survey the roadmap and standardization activities of SDN, examining its rapid evolution from a concept of programmable networks to an eco-system of cooperation and competition. From a *technical perspective*, we closely dissect the architectural design choices and compare the state-of-art SDN solutions, including both its data plane of simple forwarding hardware and its control plane towards a networking operating system, and more importantly, their interplays via open communication interfaces. Specifically, we discuss the development from earlier SDN versions based on the single match table in switch, physical centralized controller and low-level APIs, to advanced versions based on multi-stage matching switch, distributed multiple controller instances and high-level easy-to-use APIs. From an *application perspective*, we illustrate how SDN can be widely applied in both wired and wireless networks, ranging from wide-area traffic engineering for geo-distributed datacenter networks, intra-datacenter network management, switch control and optimization, to wireless network re-design. Furthermore, we envision the future development trend in SDN under broader scenarios, such as improving the scalability and reliability of control platform and enhancing the simplicity and generality of programmable interface.

2 The emergence of SDN

In conventional computer networks, a large collection of such network devices as switches and routers perform data forwarding according to complex network protocols, which are generally "born" there when manufactured by their hardware vendors. These network elements have very limited flexibility, each working as a "black box" once being deployed, so is the whole network system. Given that the software implementation is tightly coupled with the hardware infrastructure, such a system can hardly evolve to accommodate new network protocols or quality of service mechanisms. A typical example is the IPv4 to IPv6 transition in the Internet. Despite that the IPv6 protocol has been officially standardized for over 15 years and there is an urgent need for its deployment, it carries only 2.4% Internet traffic as of today [10]. There have been numerous transition plans being proposed, but the limited configurable interfaces of the existing network devices bring significant challenges for a network administrator and operator to easily deploy and manage such new services. Network operators have to learn lots of low-level configuration commands as well as understand the underlying network protocols, so as to meet high-level network service requirements with these complex commands. Moreover, as network events (e.g., congestion, IP changes) occur frequently, operators must manually re-configure the policies in response to the changing network status.

2.1 Programmable networking: Early attempts

Driven by the ability to rapidly create and deploy new control and management tools in response to users' network service demands, the idea of programmable networks has been discussed in both academia and industry for decades along with the development of traditional IP networks. The first two well-known schools of thought on programmable networks were raised in the 1990s—*Active Networking* (AN) by the DARPA [7] and *Open Signaling* by the Open Signaling Working Group (OPENSIG) [10].

The AN community advocated dynamic runtime support in network devices for new services, so as to enable network customization under the architecture of existing IP networks. The packet behavior, even the configuration of flows or switches, can be controlled by executable programs comprised in "active packets". The runtime deployment of services realizes highly dynamic software control in networks. It however imposes great complexity to the programming model for code mobility and runtime support in resource-constrained network devices.

The OPENSIG community suggested taking a telecommunications approach to make the network programmable by opening programmable network interfaces in switches and routers. The IEEE Project 1520 [12] also took an attempt to standardize software interfaces for programming of network devices, including ATM switches, circuit switches and IP routers.

These innovations have tried to create a paradigm shift for networks aiming at application layer programmability for new service deployment. Although being less successful, these early ideas of separating the forwarding hardware and the control software have greatly motivated the later development of SDN.

2.2 Decoupling software from hardware

Different from the earlier solutions, SDN refactors the relationship between network devices and the software control logic toward a fully decoupled architecture. The key argument is that the decision logic should not be implicitly or explicitly hardwired in protocols distributed among network devices. The origin of SDN can be traced back to the *4D project* [13], which was established under the Stanford Clean Slate Program [14] with ONRC support. The 4D project takes a clean slate approach to re-design network control and management, by providing network operators with a direct interface to configure network elements. Two other projects, SANE [15] and Ethane [16], both targeting enterprise networking, also made key contributions toward the SDN era. SANE leverages *logically-centralized server* to make all routing and access control decisions. In Ethane, the admittance and routing are managed by the cooperation between extremely simple *flow-based switches* and a *centralized controller*.

SDN took off when the initial version of OpenFlow [3] was released, which clearly defines the communications interface between the control and forwarding layers of a software-defined network architecture. Although originated from academia, the SDN's power of decoupling and openness has quickly attracted attention from industry, particularly user-driven organizations and cloud providers such as Google, and Microsoft. The real-world demands on network control in the cloud datacenters pushes forward the pervasive deployment and standardization of SDN, which has since become one of the most actively growing fields in the networking community.

3 SDN architecture: From data plane to control plane

Figure 1 shows a generic architecture of software-defined networking, which consists of *three layers* and *two open communication interfaces*. At a glance, the architecture is separated into two planes: (i) a *data plane* that consists of simple forwarding switches, being responsible for data forwarding; (ii) a *control plane* consisting of the controlling software, to be programmed with open APIs. Different from current switches running complex protocols among multiple devices, the forwarding hardware in SDN data plane simply forwards packets according to the rules that are set up by upper-layer controllers through software-to-hardware communication interfaces (referred to as *southbound*). The logically centralized controller functions as an OS over the networks with a set of application-layer programmable interfaces (referred to as *northbound*) and transforms the service requirements into instructions for the data plane devices.

To understand the advantages of such decoupling, we can compare it with the relationship between computer hardware and software. The hardware (e. g. , CPU versus switches) receives and executes a set of standard instructions (e. g. , X86 instructions vs. OpenFlow protocols) and the software runs on a platform (OS versus controller) via open APIs. Such a layered architecture enables device manufacturers to focus on making fast forwarding hardware whereas software engineers to design easy-to-use program API and tools, thereby making it easier to deploy new protocols, network services, complex middlebox and network virtualization.

3.1 Data plane: Simple forwarding hardware

A switch serves as the corepacket forwarding hardware in the SDN data plane. Given one or multiple flow-based tables, the switch performs actions simply according to the rules defined in the table entries. As such, all or part of the control logic can be offloaded to remote controllers by installing rules with open

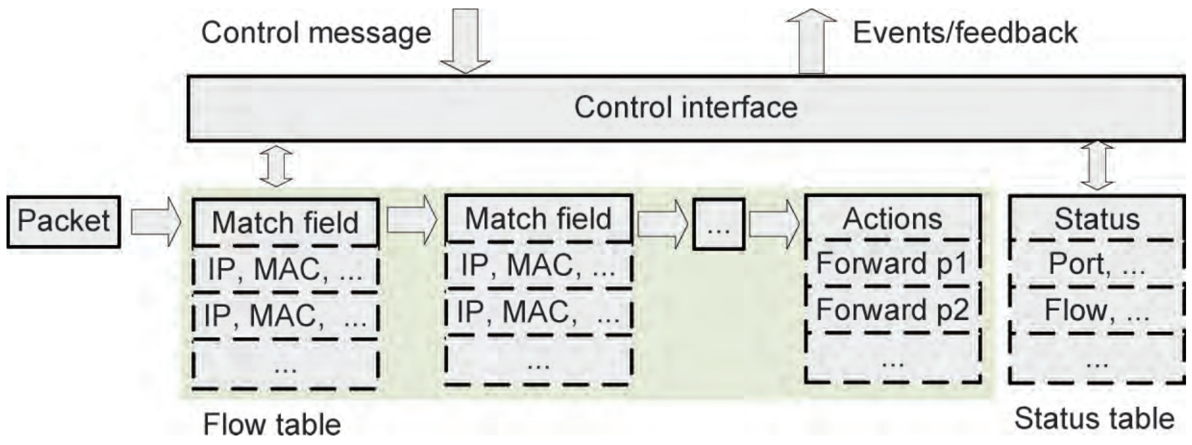


Figure 2 SDN switch abstraction.

protocols. As illustrated in Fig. 2, the design space of flow based switch involves three components: (i) the control interface, through which the controller configures the switch and receives events from the switch; (ii) the state tables, which maintain counters to record the state of each flow entry, port, queue, etc.; and (iii) the flow tables, each entry of which consists of a match field describing the state (e.g., port, VLAN, IP, and MAC) and a corresponding behavior (e.g., send out/drop packets).

In early designs, notably OpenFlow v1.0, a *Single Match Table* (SMT) was used, where a switch matches packet headers against entries in a single flow table. It is easy for switch design and programming. The entries, however, need to store every matching field defined in the table, and the rules do not scale well given the limited size of a single TCAM. *Multiple Match Tables* (MMT) extends the simple SMT by using multiple smaller match tables and processing matching through a multi-stage pipeline. It has been incorporated into the latest OpenFlow specification, allowing flexible implementation of the match tables based on network requirements.

Nevertheless, once the switch is deployed, it is difficult to reconfigure the match tables and add new forwarding behaviors beyond the existing standards. As a refinement, RMT (*Reconfigurable Match Tables*) has been suggested [17], which allows the number, widths and depths of match tables as well as all header fields to be modified without changing the underlying hardware.

The following example shows how the packets are processed in the data plane in a microscopic view. OpenFlow specifies that, when a packet arrives at a port, the switch extracts the packet header and matches it against the match field [18]. If matching successfully, the switch will handle the packet according to the action defined in the matched table entry and update the corresponding counters. Otherwise, it will generate an event through the control interface to notify the controller to setup rules for this packet. The performance of data plane mainly depends on the forwarding rate and delay, flow setup rate and delay and northbound channel bandwidth. As the number of switches increases, the interaction between switches and the centralized controller will bring significant overhead to the controller. To alleviate this bottleneck, recent solutions advocate to offload part of the control logic to switches. For example, DIFANE [19] keeps all traffic, including unmatched packets, in the data plane by directing packets through intermediate switches that store the necessary rules. Deep packet buffer for bursty traffic may also be implemented by using co-processing CPU in the switches to handle both control plane and data plane traffic [20].

3.2 Control plane: Towards a networking operating system

As shown in Fig. 3, the SDN control plane aims to enable flexible support/control for switch hardware, maintain such basic network functions as building network topology, managing device status, and routing, and provide easy programmable interfaces and robust runtime environments for network services. With the promise of being a networking operating system, a series of critical issues are to be answered in its design space: (i) What kind of API should be offered and how to translate an application layer function into a set of instruction messages for switches? (ii) How should the service threads and northbound messages be scheduled? (iii) What type of programming language and runtime support are needed?

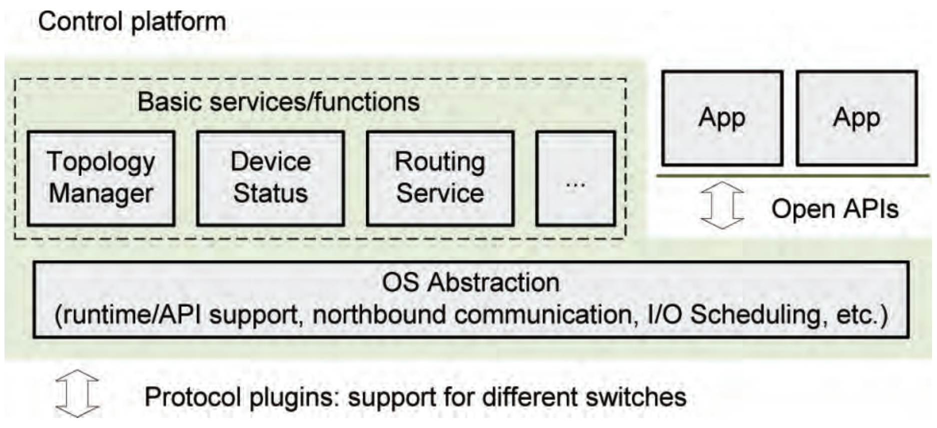

Control platform
Basic services/functions
Topology Manager
Device Status
Routing Service
...
App
App
Open APIs
OS Abstraction
(runtime/API support, northbound communication, I/O Scheduling, etc.)
Protocol plugins: support for different switches

Figure 3 Abstraction of SDN controller.

As the SDN controller is logically centralized, a straightforward solution is to deploy a single instance controller. NOX [21], the first open source control platform based on OpenFlow, leverages a physical centralized PC server to run network management applications written as centralized programs. To make the controller more OS-like, Beacon [23] further provides build-in applications for commonly needed functionalities and allows starting and stopping both existing and new applications.

The single instance solution however has obvious scaling limitations. By using multi-threaded I/O scheduling, Beacon achieves linear performance scaling. The rapid development of OpenFlow toward larger scales drives the proposition of distributed solutions with multiple instances. A representative is Onix [24], a distributed control platform that divides the network into multiple subsets, each being managed by an instance as an aggregated single node when exposed to other controllers. Different from Onix, Kandoo [25] uses a root controller to maintain the network-wide states and applies multiple local controllers to manage local control applications. By distinguishing the responsibilities of the controllers, Kandoo reduces control channel consumption by an order of magnitude compared to single-instance OpenFlow networks.

Today's open source and commercial controllers are evolving toward multi-instance solutions with higher performance and scalability, better reliability and easy-to-use APIs. Efficient status management using databases (e. g., *Network Information Base* [24] for Onix) is introduced to enhance the performance. Dynamic controller assignment can improve the utilization of controllers and balance the workloads of multiple instances [22]. Multi-path protocols have also been used to address connectivity failures, and the traditional system recovery tools can be used to handle the controller failures [24], thus increasing the reliability of the system. In Table 1, we summarize the state-of-the-art controller or controller platforms, highlighting their control models, unique features, and limitations.

Table 1 Typical projects on SDN controller platform

Projects	Control model	Owner	Deployment language	Features/Limitations
NOX [21]	Single instance	Nicira	Python/C++	Low-level API
POX [26]		Murphy McCauley (UCB)	Python	Low-level API, low performance
Beacon [23]		David Erickson (Stanford)	Java	Runtime Modularity, basic build-in services
MUL [27]		Kulcloud	C	C based, multi-threaded
SNAC [28]		Nicira	C++	Web-based policy manager
Maestro [29]		Zheng Cai (Rice)	Java	Linear performance scalability, limited by CPU
Floodlight [30]		Big Switch Networks	Java	Open source, working with OpenFlow, tested and supported
Onix [24]	Multiple instances	Nicira	C++	First distributed controller
Big Networks Controller [31]		Big Switch Networks	Java	Distributed controller built on Floodlight
OpenDaylight [32]		Community, Linux Foundation Collaborative	Java	Community-driven, open source
OpenContrail [33]		Juniper Networks	C C++	Standards-based protocols, providing all the necessary components for network virtualization
Cisco ONE [34]		Cisco	N/A	Application-centric approach, centralized visibility
VMware NSX [35]		VMware	N/A	Complete network virtualization

3.3 Bridging data and control planes

The southbound interface connects the software in the control plane and forwarding devices in the data plane, aiming at providing secure and efficient communication protocols which can transmit a controller's request for rule modification or status information, and the switch's synchronous feedback or asynchronous event messages. The protocols defined between the data plane and the control plane create a standard for designing upward-compatible switches, which can open the hardware industry and broaden the deployment of general switch hardware.

The OpenFlow protocol [3] defines the first standard communication interface between the control and forwarding layers of an SDN architecture. The OpenFlow channel is encrypted using TLS (*Transport Layer Security*), and supports three message types, namely, controller-to-switch, asynchronous, and symmetric. The controller-to-switch messages are initiated by the controller and used to modify or inspect the state of the switch. Asynchronous messages are initiated by the switch and used to update the controller of network events and changes to the switch state. Symmetric messages are initiated by either the switch or the controller and then sent without solicitation

As part of the controller platform, the northbound interface, supported by the runtime environment of a controller, enables programmability by exposing open functionality for network managers. Such open application layer interfaces refine the roles of platform providers and application providers, and creates an open environment for all kinds of network applications. Unlike the southbound interface, there is no widely accepted standard and each platform has their own APIs. There are however two consensuses on designing a good northbound interface: (i) API integrality, which allows users to flexibly request/manage network resources and query about the state of network, and (ii) easy-of-use in providing high-level functional abstraction for programmers

NOX [21] is the first platform that implements network services on OpenFlow by providing low-level interfaces. To make programming easier, Frenetic [36] further combines a high-level network query language for reading network state and a network policy management library for specifying packet-forwarding policy. Other open source controllers (e.g., Floodlight, OpenDaylight) have also released the development guidelines with detailed reference to each functionality.

4 SDN ecosystem and applications

As SDN decouples the software from hardware and provides open programmable interfaces, the technical monopoly of traditional network device vendors would be broken. As a consequence, the standardization of SDN has drawn significant attention of different organizations from all parts of the eco-system, consisting of the research community (e.g., ONS), the SDN adopters (e.g., Google, and Microsoft), the network device manufacturers (e.g., Cisco, and ASRITA) and the open source communities (e.g., OpenDayLight). While these participators collaborate to form the future of SDN, each of them has unique goals and interests on SDN standardization.

Fig. 4 lists the key players of the SDN eco-system, each of which has its respective contributions or solutions. The research community hosts the annual *Open Networking Summit* (ONS) to gather the leaders and innovators from all parts of the SDN ecosystem to share novel SDN ideas and solutions, and to shed light on the future development. The primary SDN users, such as Google and Facebook, have established a user-driven organization, *Open Networking Foundation* (ONF) [2], to define industrial SDN standards, particularly those based on OpenFlow. These solutions/standards have been implemented by hardware manufacturers, through releasing commercial infrastructures with programmable interfaces. To summarize, the major SDN projects are listed in Table 1.

At the same time, the world's top telecom operators have launched an industry-specification group, namely, *Network Functions Virtualization* (NFV) [37], to standardize software-based network functions and services. Differing from SDN that describes a network architecture, NFV emphasizes on implementing network functions that can run on industrial-standard server hardware. NFV can thus be viewed as a complementation to SDN.

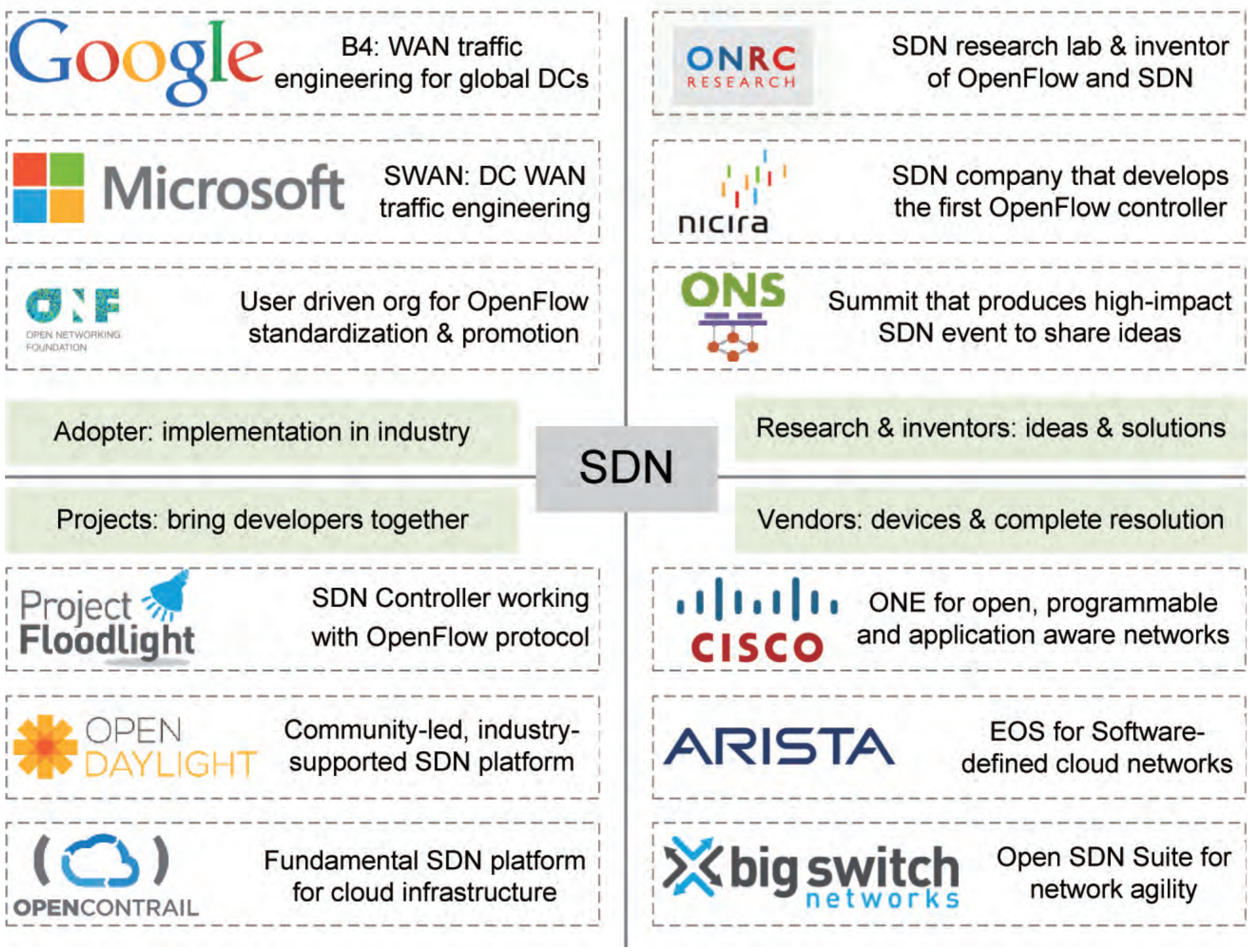

B4: WAN traffic engineering for global DCs
SDN research lab & inventor of OpenFlow and SDN
SWAN: DC WAN traffic engineering
SDN company that develops the first OpenFlow controller
User driven org for OpenFlow standardization & promotion
Summit that produces high-impact SDN event to share ideas
Adopter: implementation in industry
Research & inventors: ideas & solutions
SDN
Projects: bring developers together
Vendors: devices & complete resolution
SDN Controller working with OpenFlow protocol
ONE for open, programmable and application aware networks
Community-led, industry-supported SDN platform
EOS for Software-defined cloud networks
Fundamental SDN platform for cloud infrastructure
Open SDN Suite for network agility

Figure 4 SDN opens the networking eco-system.

With its open control interfaces, SDN can be applied in a variety of scenarios, especially for networks that need easy management, complex middleboxes, global monitoring and centralized control (see Table 2). We now take a closer look at these typical applications in different scenarios and examine how SDN benefits the networking systems.

Table 2 Representative applications of SDN

Name	Scenario and application	Advantages	Solutions	Limitations
B4 [38]	Datacenters, WAN traffic engineering	Achieving near 100% links utilization	SDN for centralized traffic engineering	Need hardware programming, bottlenecks in transmitting protocol packets
SWAN [39]	Datacenters, WAN traffic engineering	Carry 60% more traffic	Centralized controller to maintain the whole network topology and OpenFlow switches to deal with forwarding	Limited functional interfaces

(*To be continued on the next page*)

(*Continued*)

Name	Scenario and application	Advantages	Solutions	Limitations
DevoFlow [40]	Datacenter network management, flow scheduling	Offloading overhead for controller, 10—53 times fewer flow table entries and 10—42 times fewer control messages	Using rule cloning and local actions for switch forwarding rules and use sampling, triggers and reports to collect network statistics	Lack of implementation for DevoFlow switches and hardware verification
Hedera [41]	Datacenter network management, flow scheduling	Achieve 96% optimal bisection bandwidth and up to 113% better performance than the static load-balancing methods	3 main components: 1) elephant detection, 2) demand estimation, 3) placement heuristics	Relying on traffic demand estimation for N×N host to host
Meridian [42]	Clouds datacenters, network management	Supports for a service-level model for application networking in cloud	3 main logic layers: 1) network model and APIs, 2) network orchestration, 3) interfaces to underlying network devices	Only for applications like web services, no consideration for multiple virtual networks, scalability problem
NetGraph [43]	Datacenter network management	In large graphs, queries can be answered in 100 ms and consume memory less than 100 M	Complement edge weight updates, insertions and deletions for the proposed TEDI algorithm	Overhead of frequently update path information with algorithm of O(n^2) time complexity
OpenSketch [44]	Traffic measurement	The error rate is 0.04% with 600 kb memory in switch, far less than early solutions. Easily programmable	A simple three-stage pipeline in the data plane and a measurement library in the control plane	More memory consumption to achieve the same error rate compared with streaming algorithm
SIMPLE [45]	Traffic engineering, middlebox management	Achieving 6 times load balancing improvement compared with today's deployments (reducing the maximum load while distributing the traffic load more evenly), and 95% accuracy for modified packets forwarding	Using SDN-style control for middlebox-specific traffic steering. Efficient data plane, unified resource management strategy, rules for dynamic packet modifications	Not well supported for the middlebox policy changes. The controller's performance overhead, latency and correctness for inferring new rules remains to be improved.
OpenRadio [46]	Wireless network	Enabling operators to upgrade and optimize the network completely in software	A software abstraction layer that exposes a modular and declarative interface to program the PHY and MAC layers	No software defined RAN controller.
SoftRAN [47]	Wireless Network	Effective load balancing, interference management and high throughput for wireless local geographical network	Abstracting the RAN as a virtual big-base station comprised of a central controller and radio elements	Overhead scales with the devices in the network.
ElasticTree [48]	Datacenter, green computing	Save up to 50% of network energy for datacenter workloads	Dynamically adjusting the set of active network elements	Only for tree-based topologies, relying on prior knowledge of input traffic.

4.1 Switch control and optimization

With the ability of controlling every single forwarding rule in the switches, SDN can make traffic engineering much easier and more flexible. For instance, *middleboxes* (e.g., network firewalls, and Network Address Translation) have become a critical component in today's networks due to the fact that they provide crucial performance and network security. As existing middleboxes need manual configurations or configuration scripts under specific situations, SDN can simplify middleboxes management with automated programs. For example, SIMPLE [45] is an SDN-based policy enforcement layer that enables network operators to deploy logical middlebox routing policies and automatically translates them into the data plane forwarding rules. SIMPLE provides unified resource management to balance the middlebox load in response to traffic changes and can automatically deal with dynamic packet modifications.

It is worth noting that measurement APIs are not well supported in state-of-the-art SDNs. Flow-based measurements like NetFlow [54] and sFlow [55] consume significant amounts of resources. On the other hand, sketch-based streaming algorithms lack generality to support a wide variety of measurement tasks. To tradeoff generality and efficiency, OpenSketch [44] introduces a three-stage pipeline to support diverse sketches for the generality in the data plane, and uses a resource allocator to divide each switch's memory according to the number of tasks. This well balances the memory consumption and the accuracy.

4.2 WAN traffic engineering for geo-distributed datacenters

Inter-Datacenter (Inter-DC) traffic, which is at the terabits/sec level, imposes remarkable challenges to wide area networks (WANs) [38]. For consideration of transmission reliability, today's cloud providers usually deploy 2-3x over-provisioning bandwidth resource for Inter-DC traffic [38]. However, as the resource allocation using MPLS TE (*Multiprotocol Label Switching Traffic Engineering*) lacks coordination among geo-distributed services in different datacenters, the average utilization of WAN links, which is at most 40%—60% [39], have extremely poor efficiency.

SDN offers elegant solutions to improve network efficiency and reduce the over-provisioning cost, which have been advocated by major cloud providers. SWAN [39], a WAN traffic engineering platform developed by Microsoft, leverages centralized controller to maintain the global view of the network topology, and applies OpenFlow switches to deal with forwarding state and rules updates. By frequently updating the flow schedules in the data plane to match the traffic demand, SWAN can carry 60% more traffic than the earlier solutions.

Another SDN solution, B4 [38], comes from Google's datacenters across the planet. It constructs a private WAN to achieve fault tolerance, cost efficiency and required control that can hardly be realized by traditional WAN architectures. The B4 architecture consists of three logical layers: a *global controller* and *site controller* for each datacenter to support centralized traffic engineering, and *customized switches* to integrate the OpenFlow functionality. B4 can dynamically allocate bandwidth or shift application demands to handle competing demands during resource constraint and link failure, driving links to near full utilization as well as providing full control on edge servers and networks.

4.3 Intra-datacenter network management

Intra-datacenter networks are shared among a large number of applications/VMs. It is critical to provide global and flexible management tools for datacenter operators. Compared to traditional networks with only limited configurable interfaces, SDN can provide easier network management, in two critical aspects: (i) *centralized visibility*, which reduces the cost of setting up every single switch; (ii) *abundant programmable interfaces*, which enables automatic monitoring and control in a network service.

The former focuses on underlying flow scheduling for different traffic patterns in datacenters. Compared with a WAN, mice flows are common in datacenter networks [53], and therefore it needs much more data-and-control plane communication and statistic collecting is not cost-effective withOpenFlow. For this particular traffic pattern, DevoFlow [40] introduces rule cloning and local actions to offload the control overhead to a switch, and extends OpenFlow with new threshold-based triggers to improve the efficiency of statistic collection. Hedera [41] focuses on flow scheduling in datacenter networks. To avoid congestion resulting from existing Equal-Cost Multi-path Routing (ECMP), it seamlessly integrates three components: *elephant detection* for finding large flows using statistics collected by underlying SDN switches; *demand estimation* designed for finding flows' overall fair bandwidth allocation; and *placement heuristic* to find near-optimal placement solutions.

The latter focuses on providing upper-layer control interfaces over the network. It has been argued that a service-level network model and policy abstractions are integral parts of cloud applications [42]. To this end, an SDN controller platform, Meridian [42], is proposed, integrating three logic layers, namely, an *abstract API layer*, a *network orchestration layer*, and a *network driver layer*. Another solution, NetGraph [43], is a shared graph library that supports network management operations (e.g., network-aware VM placement, and real time network monitoring) in dynamic cloud network topology. It provides scalable and fast graph query with open APIs for receiving topology updates and dynamically computing graph queries. With programmable interfaces, cloud providers are able to deploy their bandwidth allocation policies in SDN data centers, so as to provide predictable network performance for applications/VMs in clouds [49—52].

Energy efficiency of large-scale datacenters has drawn critical attention in recent years, too. With SDN interfaces for dynamically allocating network resources, network operators are able to flexibly use network devices according to the traffic load. This complements the existing works that mainly focus on low-powered hardware manufacturing or dynamic right-sizing for servers. An example is ElasticTree [48], which reduces datacenter energy cost by focusing on the networking elements. It computes the network subset that can satisfy the demand of current data center workload, generates SDN routing rules, and shuts down the idle networking elements to create such a power-saving network subset. For a wide array of traffic patterns, ElasticTree can save energy ranging from 25% to 62%.

5 Conclusion and future directions of SDN

So far we have surveyed the state-of-the-art of software-defined networking (SDN), including architecture design, typical applications and industrial open source/commercial projects. We have summarized the issues, challenges and trends of the SDN architecture design space for control plane, data plane and communication interfaces, and explored the representative SDN application scenarios with cutting-edge solutions.

SDN has shown great potentials in improving network management/measurement for diverse networks. Besides the use in industry, SDN also offers an open platform for the education of networking architecture and protocols itself. A typical computer system course involves many inside projects; yet this can hardly be done for networking courses nowadays, given that the existing network devices cannot be easily programmed for self-designed network services or protocols. This could be overcome by SDN, which enables the design of new networking functions by students with a customizable control plane and visible packet forwarding in the data plane. It will certainly offer students the first-hand experiences, preparing them better for related research or product development in their career.

Despite its significance in the networking or even the whole IT sector, SDN is still in its infancy and both opportunities and challenges are arising. We now examine the critical challenges in designing and implementing SDN and highlight potential directions toward the future of SDN.

5.1 Opening SDN architecture

Targeting at a new networking eco-system of good performance and ease-of-use functionality, scalability, reliability, simplicity and generality remain great challenges for the SDN development and deployment. So far there lacks a universally accepted SDN platform and very few productive SDN applications have been deployed in the real world.

Scalability and reliability of control platform. An import challenge in implementing an SDN platform is to make it scalable with the growth of network devices, while maintaining it logically centralized. To improve the controller's scalability, newer solutions (e.g., Onix [24] and Kandoo [25]) are changing towards physically distributed controllers, since one single server can handle limited flows at a time, and it also suffers from single point failures. Multiple controller instances potentially achieve better scalability and reliability; yet the associated complexity can be higher, and such new problems as controller placement, task allocation and scheduling, and communications among the controllers are to be addressed [56].

Simplicity and generality of programmable interface. As SDN promises to enable easy programmability for network managers, it is necessary to design a simple and general southbound interface supporting the deployment of network applications on various SDN forwarding hardware. We have seen growing interest in designing high-level languages for SDNs [57, 60]. To overcome the low developer productivity, high-level API and policy abstraction have been used to make application programming simple [42]. The runtime ability for starting and terminating applications have also been supported to make controller OS-like. Still, as compared conventional programming languages that have been closely examined in the past decades, these new SDN-based languages are still in early stages, whose foundations are yet to be built, verified, and optimized theoretical and practically.

5.2 Opening network functionality with SDN

Cloud datacenters. Different from the Internet, data center networks are a fabric of homogeneous servers, connected by high throughput network devices in specific topologies. Such a design well accommodates centralized global control and monitoring. On the other hand, datacenter networks are shared among different users or applications, which need complex access control, large numbers of VLANs, strict performance isolation and accurate monitoring. These can hardly be implemented with traditional network hardware with limited programmability. Google and Microsoft have both successfully deployed SDN-based global datacenter WANs [38, 39], which significantly improve link utilization between datacenters. For intra-datacenter networks, SDN can be deployed for flow scheduling, traffic monitoring, network configuration and performance isolation. Existing works (e. g., [38, 41, 42]) have focused on dynamic flow scheduling for high throughput and load balancing. Datacenter monitoring [61] and network performance isolation for VMs remain critical issues to be examined in the future research.

Enterprise networks and campus networks. Enterprise networks, campus networks or ISP networks are highly dynamic environments with mass of events and complex access control. Their users frequently join or leave the network, demanding flexible VLAN configuration for user authentication and traffic monitoring. With high level interfaces on configuring forwarding rules, SDN can improve management and security for these networks. A representative real-world example is Procera [62, 61], which supports the campus network configuration with a high-level language. Another platform, SIMPLE [45], simplifies middlebox traffic steering by providing configurable interfaces. These existing solutions are designed for specific contexts; more SDN practices and general implementations are to be explored to improve the management in these networks.

5.3 Reshaping wireless network design

It has been widely accepted that the future Internet is moving towards wireless mobile networking, and today's shipments of such wireless mobile terminals as laptops, smartphones, and tablets are quickly surpassing that of the fixed PCs. As wireless protocols evolve rapidly, operators and vendors need to continuously update the software and network devices to handle new traffic class from new wireless mobile applications. In the past, the protocol definition was closely coupled with the hardware, so that protocol changes may require replacing the base stations, which leads to excessively higher expense in today's rapidly changing and dense networks.

To overcome this drawback, OpenRadio [46] presents a novel design for a programmable wireless data plane, whose core component is a software abstraction layer that exposes a modular and declarative interface to program the PHY and MAC layers. By decoupling wireless protocol definitions from the hardware and decomposing the wireless protocols into separate processing and decision plane component, OpenRadio makes base stations remotely programmable to enable vendors and operators to upgrade and optimize the network completely in software.

Another fundamental problem in today's radio access network is how to best utilize and manage limited spectrum to provide wide-area wireless connectivity to mobile devices. The performance of existing distributed coordination algorithms does not scale well, as they need to work with a large number of base stations, especially in terms of latency [47]. Based on SDN, SoftRAN [47] fundamentally reshapes the radio access layer, abstracting the radio access networking in a local geography as a virtual big-base station comprised of a central controller (a logically centralized entity) and radio elements (the base stations). With the knowledge of the entire network state, the controller can recognize the edge users easily and find the best allocation method for load balancing and utility optimization.

5.4 Network OS: Who will be the next winner?

A *Networking Operating System* refers to software as an operating system that operates the network/link layer functions (e.g., MPLS) or provides network services (e.g., VPN). Early network OSes, like NetW are whose popularity can be traced back to the 1990s, ran on servers in a LAN and providednetwork services based on existing protocols. Targeting traditional enterprise services (e.g., file sharing, print), they can hardly be employed to operate large-scale cloud datacenters or to support network programmability. Another type of Network OSe, including JUNOS, and Cisco IOS, is embedded in a router to operate the functions in layer 3 of the OSI model. These embedded Network OSes with customizable functions emerged in year 2000 and have since led the trend of opening network for programmable interfaces. Nevertheless, the embedded software only runs on customized infrastructure of their respective companies in a closed system. Few of them have successfully built an open networking eco-system, for the closed functions can hardly meet the requirements of network services from various generic users.

Looking back into the past six years, however, SDN has successfully brought together the members of the network eco-system by building an open environment, where the fundamental concepts, tools and scientific methods are provided to open computer networks. Given the rapid growth of cloud computing and big data, which urgently demand open and flexible networking, network OSes harnessing SDN will undoubtedly be the next hot spot in the IT industry. Again, analogical to the proliferation of PC OS market in which many famous trademarks (Linux, Microsoft Windows, Apple iOS, etc.) have been 4born in the past decades, we will surely witness the involvement of both IT giants and new startups in this next wave competition of the network OSmarket, and witness the birth of new era's winners and their legends.

Acknowledgments

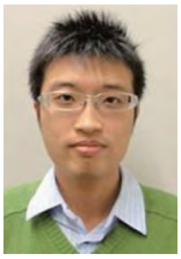
This study was supported in part by a grant from the National Natural Science Foundation of China (NSFC) (Grant Nos. 61370232 and 61520106005).

References

[1] Software—Defined Networking — The New Norm for Networks. ONF White Paper, 2013.
[2] Open Networking Foundation. https://www.opennetworking.org/.
[3] McKeown N, Anderson T, Balakrishnan H, Parulkar G, Peterson L, Rexford J, Shenker S, and Turner J. OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 2008, 38(2):69—74.
[4] Market Report: SDN Market Sizing. Plexxi, Lightspeed Venture Partners and SDNCentral, April, 2013.
[5] SDN Shakes Up the Status Quo in Datacenter Networking. IDC, December 19, 2012. http://www.idc.com/getdoc.jsp?containerId=prUS23888012.
[6] Nunes BAA, Mendonca M, Nguyen XN, Obraczka K, Turletti TA. Survey of Software—Defined Networking: Past, Present, and Future of Programmable Networks. IEEE Communications Surveys and Tutorials.
[7] CampbellAT and et al. A survey of programmable networks. ACM SIGCOMM Computer Communication Review, 1999.
[8] Feamster N, Rexford J, Zegura E. The Road to SDN: An Intellectual History of Programmable Networks.
[9] Scott—Hayward S, O'Callaghan G, Sezer S (2013, November). Sdn Security: A Survey. In Future Networks and Services (SDN4FNS), 2013 IEEE SDN for (pp. 1—7). IEEE
[10] Google IPv6 statistics. http://www.google.com/ipv6/statistics.html.
[11] Campbell A, Katzela I, Miki K, and Vicente J. Open signaling for ATM, Internet and mobile networks. ACM SIGCOMM Computer Communication Review, 1999.
[12] Biswas J, Lazar AA, Huard JF, Lim K, MahjoubS, Pau LF, Suzuki M, Torstensson S, Wang W, Weinstein S. The IEEE P1520 Standards Initiative for Programmable Network Interfaces. IEEE Communications Magazine, 1998, 36(10):64 – 70.

[13] Greenberg A, Hjalmtysson G, Maltz D, Myers A, Rexford J, Xie G, Yan H , Zhan J, Zhang H. A clean slate 4d approach to network control and management. ACM SIGCOMM Computer Communication Review, 2005, 35(5):41—54.
[14] Clean Slate Program. http://cleanslate. stanford. edu/
[15] Casado M, Garfinkel T, Freedman M, Akella A, Boneh D, McKeown N, Shenker S. SANE: A Protection Architecture for Enterprise Networks. In Usenix Security Symposium, 2006.
[16] Casado M, Freedman M, Pettit J, Luo J, McKeown N, Shenker S. Ethane: Taking control of the enterprise. ACM SIGCOMM Computer Communication Review,2007, 37(4):1—12.
[17] Bosshart P, Gibb G, Kim H, Varghese G, McKeown N, Izzard M, Mujica F, Horowitz M, "Forwarding metamorphosis: Fast programmable match—action processing in hardware for SDN," in ACM SIGCOMM, 2013.
[18] Open Flow Switch Specification 1. 4. 0. Open Networking Foundation, 2013.
[19] Yu M, Rexford J, Freedman M, Wang J . Scalable flow—based networking with DIFANE. In ACM SIGCOMM, 2010.
[20] Lu G, Miao R, Xiong Y, Guo C. Using CPU as a Traffic Co—processing Unit in Commodity Switches. In ACM HotSDN, 2012.
[21] Gude N, KoponenT, Pettit J, Pfaff B , Casado M , McKeown N, and Shenker S, "NOX: towards an operating system for networks," in ACM SIGCOMM, 2008.
[22] Wang T, Liu FM , Guo J, Xu H, "Dynamic SDN Controller Assignment in Data Center Networks: Stable Matching with Transfers", in Proc. of IEEE INFOCOM, San Francisco, CA, USA, April, 2016.
[23] Erickson D, "The beacon openflow controller," in ACM HotSDN, 2013.
[24] Koponen T, Casado M, Gude N, Stribling J, Poutievski L, Zhu M, Ramanathan R, Hama T, Shenker S, "Onix: A distributed control platform for large—scale production networks," in USENIX OSDI, 2010.
[25] Hassas Yeganeh S, Ganjali Y, "Kandoo: A framework for efficient and scalable offloading of control applications," in ACM HotSDN, 2012.
[26] POX. http://www. noxrepo. org/pox.
[27] Mul Open flow controller. http://sourceforge. net/projects/mul/.
[28] SNAC. https://github. com/bigswitch/snac/.
[29] Zheng C, Alan LC, Eugene TS. Ng Maestro: A system for scalable OpenFlow control. Technical Report TR10 — 08, Rice University, 2010.
[30] Floodlight. http://www. projectfloodlight. org.
[31] Big Network Controller. http://www. bigswitch. com/products/SDN—Controller.
[32] OpenDaylight. http://www. opendaylight. org/.
[33] OpenContrail. http://opencontrail. org/.
[34] Cisco Open Network Environment. http://www. cisco. com/web/solutions/trends/open_network_environment.
[35] VMware NSX. http://www. vmware. com/products/nsx/.
[36] Foster N, Harrison R, Freedman MJ, Monsanto C, Rexford J, Story A, Walker D. "Frenetic: A network programming language," in ACM SIGPLAN ICFP, 2011.
[37] Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action. OFV white paper, 2012.
[38] Jain S, Kumar A, Mandal S, Ong J, Poutievski L, Singh A, Venkata S, Wanderer J, Zhou J, Zhu M, Zolla J, Hlzle U, Stuart S, Vahdat A. "B4: Experience with a globally—deployed software defined WAN," in ACM SIGCOMM, 2013.
[39] Hong CY, Kandula S, Mahajan R, Zhang M, Gill V, Nanduri M, WattenhoferR. "Achieving high utilization with software—driven WAN," in ACM SIGCOMM, 2013.
[40] Curtis AR, Mogul JC, Tourrilhes J, Yalagandula P, Sharma P, and Banerjee S . "DevoFlow: Scaling flow management for high—performance networks," in ACM SIGCOMM, 2011.
[41] Al—Fares M, Radhakrishnan S, Raghavan B, Huang N, Vahdat A. "Hedera: Dynamic flow scheduling for data center networks," in USENIX NSDI, 2010.
[42] Banikazemi M, Olshefski D, Shaikh A, Tracey J, and Wang G. "Meridian: An SDN platform for cloud network services," IEEE Communications Magazine, Feb 2013.
[43] Raghavendra R, Lobo J, Lee KW. "Dynamic graph query primitives for SDN — based cloud network management," in ACM HotSDN, 2012.
[44] Yu M, Jose L, and Miao R, "Software defined traffic measurement with OpenSketch," in USENIX NSDI, 2013
[45] Qazi Z, Tu C, Chiang L, Miao R, Sekar V, Yu M, "SIMPLE — fying middlebox policy enforcement using SDN," in ACM SIGCOMM, 2013.
[46] Bansal M, Mehlman J, Katti S, and Levis P, "OpenRadio: A programmable wireless dataplane," in ACM HotSDN, 2012.
[47] Gudipati A, Perry D, Li LE, Katti S. "SoftRAN: Software defined radio access network," in ACM HotSDN, 2013.
[48] Heller B, Seetharaman S, Mahadevan P, Yiakoumis Y, Sharma P, Banerjee S, McKeown N, "ElasticTree: Saving energy in data center networks," in USENIX NSDI, 2010.
[49] Guo J, Liu FM, Tang HW, Lian YN, Jin H, Lui JCS, "Falloc: Fair Network Bandwidth Allocation in IaaS Datacenters Via a Bargaining Game Approach", in Proc. of IEEE ICNP, October, Goettingen, Germany, 2013.
[50] Guo J, Liu FM, Huang XM, Lui JCS, Hu M, Gao Q, Jin H, "On Efficient Bandwidth Allocation for Traffic Variability in Datacenters", in Proc. of IEEE INFOCOM, April, Toronto, 2014.
[51] Guo J, Liu FM, Lui J, Jin H, "Fair Network Bandwidth Allocation in IaaS Datacenters via a Cooperative Game Approach", IEEE/ACM Transactions on Networking (ToN), Volume 24, Issue 2, April 2016.
[52] Liu FM , Guo J , Huang XM , Lui J. "eBA: Efficient Bandwidth Guarantee under Traffic Variability in Datacenters", to appear in IEEE/ACM Transactions on Networking (ToN), 2016.
[53] Kandula S, Sengupta S, Greenberg A, Patel P, Chaiken R. "The nature of data center traffic: measurements & analysis," in ACM IMC, 2009.
[54] Cisco NetFlow. http://www. cisco. com/go/netflow. ?
[55] Wang M, Li B, Li Z. sFlow: Towards resource—efficient and agile service federation in service overlay network. In IEEE ICDCS, 2004.
[56] Heller B, Sherwood R, and McKeown N, "The controller placement problem," in ACM HotSDN, 2012.

[57] Foster N, Freedman MJ, Guha A, Harrison R, Katta NP, Monsanto C, Reich J, Reitblatt M, Rexford J, Schlesinger C, Story A, Walker D. "Languages for software—defined networks," IEEE Communications Magazine, 2013.
[58] Voellmy A, Wang JC, Yang YR , Ford B, Hudak P. "Maple: Simplifying SDN programming using algorithmic policies." In SIGCOMM, 2013.
[59] Ferguson A, Guha A, Liang C, Fonseca R, Krishnamurthi S. "Participatory networking: An API for application control of SDNs," in ACM SIGCOMM, 2013.
[60] Monsanto C, Foster N, Harrison R, Walker D. "A compile and run—time system for network programming languages," in ACM SIGPLAN—SIGACT POPL, 2012.
[61] Moshref M, Yu M, Govindan R. "Resource/Accuracy tradeoffs in software defined measurement," in ACM HotSDN, 2013.
[62] Kim H, Feamster N. "Improving network management with software defined networking," IEEE Communications Magazine, Feb 2013.


Liu Fangming

Liu Fangming (**刘方明**) is a Professor in the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. He received the B. Eng. degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, and the Ph. D. degree from the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong. From 2009 to 2010, he was a visiting scholar at the Department of Electrical and Computer Engineering, University of Toronto, Canada. He is selected into the National Top-Notch Young Talents Program of National High-level Personnel of Special Support Program issued by the Central Organization Department of CPC, and he is named the CHUTIAN Scholar of Hubei Province, China. He is one of the Youth Scientists of the National 973 Basic Research Program Project on Software-defined Networking (SDN)-based Cloud Datacenter Networks, which is one of the largest SDN research projects in China. From 2012 to 2013, he was invited as a StarTrack Visiting Young Faculty in Microsoft Research Asia (MSRA), Beijing. In 2016, he served as an NSFC review panel committee expert for general funding projects, and invited to be an expert for reviewing candidates of CCF Outstanding Doctoral Dissertation Award.

His research interests include Cloud Computing and Data Center, Green Sustainable Computing and Communications, SDN/NFV and Virtualization, Internet Content Distribution and P2P Systems. He is a PI or co-PI of several NSFC grants and key projects on large-scale and energy-efficient datacenters, a PI of Huawei collaboration project on OpenStack based hybrid clouds, as well as involved in a National 863 Project on Inspur cloud server systems as a key member. He has a series of publications on Proceedings of the IEEE (ESI highly cited paper), JSAC, TON, TPDS, TC, ACM/IFIP/USENIX Middleware, INFOCOM, ICDCS, ICNP, ACM NOSSDAV, ACM e-Energy, ACM SIGMETRICS, etc. He received the ACM Wuhan Rising Star Award, and is a co-recipient of two Best Paper Awards from IEEE GLOBECOM 2011 and IEEE IUCC 2012, respectively; as well as a co-recipient of Best Paper Candidates of IEEE/ACM IWQoS 2016 and IC2E 2016. In particular, several key algorithms and system prototypes developed by him and his team have been deployed in real-world systems with a large number of users.

As an IEEE senior member and CCF senior member, he was a Guest Editor for the *IEEE Network Magazine* and *IEEE Systems Journal*, an Associate Editor for the *Frontiers of Computer Science*, and the Editor-in-Chief of *EAI Endorsed Transactions on Collaborative Computing*. He also served as the Poster/Demo Co-Chair of ICNP 2016 and the Publicity Co-Chair of IEEE/ACM IWQoS 2017, INFOCOM 2017 SmartCity Workshop program co-chair, CCF NASAC 2015 program vice-chair, and the TPC for IEEE INFOCOM 2013—2017, ICDCS 2015—2017, ICNP 2014, ACM Multimedia 2014 and 2016, ACM e-Energy 2016—2017, IEEE/ACM IWQoS 2016—2017, etc. He is also an ACM China SIGCOMM Chapter committee member.