A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation

Fei Hu, Qi Hao, and Ke Bao

***Abstract*—Software-defined network (SDN) has become one of the most important architectures for the management of large-scale complex networks, which may require repolicing or reconfigurations from time to time. SDN achieves easy repolicing by decoupling the control plane from data plane. Thus, the network routers/switches just simply forward packets by following the flow table rules set by the control plane. Currently, OpenFlow is the most popular SDN protocol/standard and has a set of design specifications. Although SDN/OpenFlow is a relatively new area, it has attracted much attention from both academia and industry. In this paper, we will conduct a comprehensive survey of the important topics in SDN/OpenFlow implementation, including the basic concept, applications, language abstraction, controller, virtualization, quality of service, security, and its integration with wireless and optical networks. We will compare the pros and cons of different schemes and discuss the future research trends in this exciting area. This survey can help both industry and academia R&D people to understand the latest progress of SDN/OpenFlow designs.**

***Index Terms*—Software-defined network (SDN), OpenFlow, network virtualization, QoS, security.**

I. Introduction

A. Motivations

CONVENTIONAL networks utilize special algorithms implemented on dedicated devices (hardware components) to control and monitor the data flow in the network, managing routing paths and determining how different devices are interconnected in the network. In general these routing algorithms and sets of rules are implemented in dedicated hardware components such as Application Specific Integrated Circuits (ASICs) [1]. ASICs are designed for performing specific operations. Packet forwarding is a simple example. In a conventional network, upon the reception of a packet by a routing device, it uses a set of rules embedded in its firmware to find the destination device as well as the routing path for that packet. Generally data packets that are supposed to be delivered to the same destination are handled in similar manner. This operation takes place in inexpensive routing devices. More expensive routing devices can treat different packet types in different manners based on their nature and contents. For example, a Cisco router allows the users to mark out the priorities of different flows through customized local router programming. Thus we can manage the queue sizes in each router directly. Such a customized local router setup allows more efficient traffic congestion and prioritization control.

A problem posed by this methodology is the limitation of the current network devices under high network traffic, which poses severe limitations on network performance. Issues such as the increasing demand for scalability, security, reliability and network speed, can severely hinder the performance of the current network devices due to the ever increasing network traffic. Current network devices lack the flexibility to deal with different packet types with various contents because of the underlying hardwired implementation of routing rules [2]. Moreover, the networks, which make up the backbone of the Internet, need to be able to adapt to changes without being hugely labor intensive in terms of hardware or software adjustments. However, traditional network operations cannot be easily re-reprogrammed or re-tasked [3].

A possible solution to this problem is the implementation of the data handling rules as software modules rather than embedding them in hardware. This method enables the network administrators to have more control over the network traffic and therefore has a great potential to greatly improve the performance of the network in terms of efficient use of network resources. Such an idea is defined in an innovative technology, called Software-Defined Networking (SDN) [4]. Its concept was originally proposed by Nicira Networks based on their earlier development at UCB, Stanford, CMU, Princeton [1]. The goal of SDN is to provide open, user-controlled management of the forwarding hardware in a network. SDN exploits the ability to split the data plane from the control plane in routers and switches [5]. The control plane can send commands down to the data planes of the hardware (routers or switches) [6]. This paradigm provides a view of the entire network, and helps to make changes globally without a device-centric configuration on each hardware unit [7]. Note that the control panel could consist of one or multiple controllers, depending on the scale of the network. If using multiple controllers, they can form a peer-to-peer high-speed, reliable distributed network control. In any case, all switches in the data plane should obtain the consistent view of the data delivery. The switches in the data plane just simply deliver data among them by checking the flow tables that are controlled by the controller(s) in the control panel. This greatly simplifies the switches' tasks since they do not need to perform control functions.

The concept of SDN is not entirely new. As a matter of fact, a few decades ago people could use special infrastructure

Manuscript received September 29, 2013; revised January 30, 2014 and April 2, 2014; accepted May 15, 2014. Date of publication May 22, 2014; date of current version November 18, 2014. *(Corresponding author: Q. Hao.)*

F. Hu and K. Bao are with the Department of Electrical and Computer Engineering, The University of Alabama, Tuscaloosa, AL 35487 USA (e-mail: fei@eng.ua.edu; kbao@crimson.ua.edu).

Q. Hao is with the Department of Electrical Engineering, The South University of Science and Technology of China, Shenzhen, Guandong 518055, China (e-mail: hao.q@sustc.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/COMST.2014.2326417

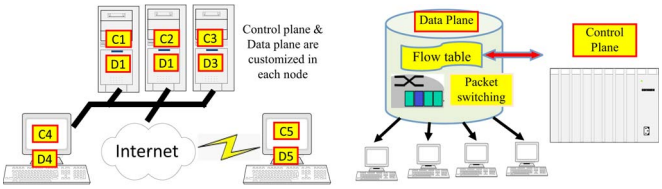


Fig. 1. Comparison of traditional network (left) and SDN (right).

(such as cloud computing hardware) to decouple the network operating system (similar to the control functions in SDN control plane) from computing-intensive applications (similar to the data delivery in data plane). Today cloud computing enables the networked computation and storage without using local resources. Such a decoupling of control and data plays a critical role in large-scale, high-speed computing system.

SDN results in improved network performance in terms of network management, control and data handling. SDN is a potential solution to the problems faced by conventional network (Fig. 1 [3]–[5]) and is gaining more acceptance in applications such as cloud computing. It can be used in data centers and workload optimized systems [8]. By using SDN, the administrators have the ability to control the data flow as well as to alter the characteristics of the switching devices (or routing devices) in the network from a central location, with control application implemented as software module without the need of dealing with each device individually [10]. This gives the network administrators the ability to arbitrarily change routing tables (routing paths) in network routing devices. It also allows an extra layer of control over the network data since the administrator can assign high/low priorities to certain data packets or allow/block certain packets flowing through the network [1]–[3].

From cloud computing perspective, SDN provides great benefits. First, it makes cloud provider more easily deploy different vendors' devices. Traditionally the big cloud providers (such as Google, Amazon, etc.), have to purchase the high-performance switchers/routers from the same vendor in order to easily re-configure the routing parameters (such as routing table update period). Different vendors' routers have their own pros and cons. However, it is a headache to customize each router since each vendor may have its own language syntax. Now SDN allows a cloud provider to fast re-policy the routing or resource distribution issues as long as each vendor's routers follow the SDN standard. Second, it enables a cloud user to more efficiently use the cloud resources or conduct scientific experiments by creating virtual flow slices. The OpenFlow protocol is compatible to GENI standard, and this enables a user to arbitrarily create slices/slivers without being aware of the physical network infrastructure. No matter the infrastructure is wireless or wired system, and no matter how the cloud provider deploys different storage units in various locations, the concept of virtual flow in a SDN makes data flow transparently route through all cloud devices.

SDN is less expensive due to universal, data-forwarding switching devices that follow certain standards, and provides more control over network traffic flow as compared to the conventional network devices. Major advantages of SDNs include [11]–[15], [17]–[19].

1) Intelligence and Speed: SDNs have the ability to optimize the distribution of the workload via powerful control panel. This results in high speed transmissions and makes more efficient use of the resources.

2) Easy Network Management: The administrators have a remote control over the network and can change the network characteristics such as services and connectivity based on the workload patterns. This enables administrators to have more efficient and instant access to the configuration modifications.

3) Multi-Tenancy: The concept of the SDN can be expanded across multiple partitions of the networks such as the data centers and data clouds. For example, in cloud applications, multiple data center tenants need to deploy their applications in virtual machines (VMs) across multiple sites. Cloud operators need to make sure that all tenants have good cross-site performance isolation for tenant specific traffic optimization. Existing cloud architectures do not support joint intra-tenant and inter-tenant network control ability. SDN can use decoupled control/data planes and resource visualization to well support cross-tenant data center optimization [133].

4) Virtual Application Networks: Virtual application networks use the virtualization of network resources (such as traffic queues in each router, distributed storage units, etc.) to hide the low-level physical details from the user's applications. Thus a user can seamlessly utilize the global resources in a network for distributed applications without direct management of the resource separation and migration issues across multiple data sites. Virtual application networks can be implemented by the network administrators by using the distributed overlay virtual network (DOVE) which helps with transparency, automation and better mobility of the network loads that have been virtualized [2], [5]. As a matter of fact, a large chunk of SDN is along the rational of virtualization. Virtualization can hide all lower level physical network details and allow the users to re-policy the network tasks easily. Virtualization has been used in many special networks. Within the context of wireless sensor networks (WSNs), there was a laudable European initiative called VITRO, which has worked precisely on this. The concept of virtual WSN [131] separates the applications from the sensor deployment details. Thus we can run multiple logic sensing applications over the same set of physical sensors. This makes the same WSN serve multiple applications.

B. SDN Implementation: Big Picture

Here, we briefly summarize the SDN design aspects. In Sections II–VIII, we will provide the details of each design aspect. Since SDN's control plane enables software-based re-policing, its re-programming should also follow general software design principle [37]. Here, we first briefly review the software design cycle. The design of a software module typically follows 3 steps: (1) design; (2) coding and compiling; and (3) unitary tests. SW debuggers are critical tools. (e.g., gdb [38]). A next usability level is provided by the integrated development environment (IDEs) such as Eclipse [39]. As a promising software design principle, component-based software engineering (CBSE) [40] has been proposed in the 4WARD project [41]. The Open Services Gateway initiative (OSGi) [42] has also

been used for a full life cycle of software design. The Agile SW development methodology proposed in [43] has been used to provide better feedback between different stages than conventional waterfall methodologies [44].

Regarding controllers, examples include Nox [48] (written in C), POX [49] (in Python), Trema [50], floodlight [51] (in Jave), etc. NOX [48] was the first OpenFlow controller implementation. It is written in C++. An extension of NOX is implemented in POX [49]. NOX can run in Windows, Linux, Mac OS, and other platforms. A Java-based controller implementation is called Beacon [52]. Its extension is Floodlight controller [53]. It can virtualize the SDN control via the OpenStack [54] architecture. Trema controller is now shipped with OpenFlow network emulator based on Wireshark [55].

Before practical OpenFlow design, there are some good simulating tools for initial proof-of-concept, such as NS-2 [56] with OpenFlow Software Implementation Distribution (OF-SID) [57]. Recently, Mininet [15] has become a powerful emulation tool.

SDN/OpenFlow programming languages have been studied in some projects. For example, FML [58] enables easy SDN network policy definitions. Procera [58] defines controller policies and behaviors. The Frenetic language [59] allows the programs written for one platform to work in other platforms.

SDN/OpenFlow debuggers have been used to trace the controller's program execution status. ndb [60] mimics GNU debugger gdb [38] and uses breakpoints and back-traces to monitor the network behaviors. Tremashark [61] plugs Wireshark [55] into Treama [50]. It is now evolving to another powerful debugging tool called OFRewind [62]. FlowCheck [63] can check the updating status of flow tables. A more comprehensive tool called NICE [64], has generated a preliminary version [65], and can be used to analyze the codes and packet flows. Through the above tools, OpenFlow testbeds are able to be established worldwide such as GENI [66] in the USA, Ofelia [67] in the European Union and JGN [68] in Japan.

C. OpenFlow: A Popular Protocol/Standard of SDN

A number of protocol standards exist on the use of SDN in real applications. One of the most popular protocol standards is called OpenFlow [8]–[10], [16], [20]. OpenFlow is a protocol that enables the implementation of the SDN concept in both hardware and software. An important feature of OpenFlow is that scientists can utilize the existing hardware to design new protocols and analyze their performance. Now it is becoming part of commercially available routers and switches as well.

As a standard SDN protocol, OpenFlow was proposed by Stanford. Regarding testbeds of OpenFlow, many designs have been proposed for OpenFlow protocols. They use open source codes to control universal SDN controllers and switches. Regarding switches, OpenVSwitch (OVS) [45] is one of the most popular, software-driven OpenFlow switch. Its kernel is written in Linux 3.3 and its firmware including Pica8 [46] and Indigo [47] is also available.

OpenFlow is flow-oriented protocol and has switches and ports abstraction to control the flow [21]–[27]. In SDN, there is a software named controller which manages the collection

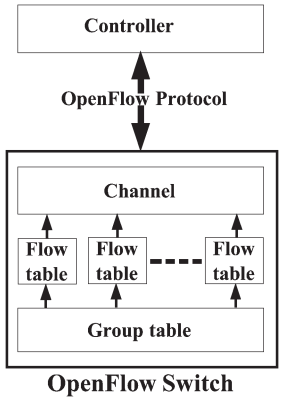


Fig. 2. OpenFlow model.

of switches for traffic control. The controller communicates with the OpenFlow switch and manages the switch through the OpenFlow protocol. An OpenFlow switch can have multiple flow tables, a group table, and an OpenFlow channel (Fig. 2 [22]–[26]). Each flow table contains flow entries and communicates with the controller, and the group table can configure the flow entries. OpenFlow switches connect to each other via the OpenFlow ports.

Initially the data path of the OpenFlow routing devices has an empty routing table with some fields (such as source IP address, QoS type, etc.). This table contains several packet fields such as the destination of different ports (receiving or transmission), as well as an action field which contains the code for different actions, such as packet forwarding or reception, etc. This table can be populated based on the incoming data packets. When a new packet is received which has no matching entry in the data flow table, it is forwarded to the controller to be processed. The controller is responsible for packet handling decisions, for example, a packet is either dropped, or a new entry is added into the data flow table on how to deal with this and similar packets received in the future [27], [28].

SDN has the capability of programming multiple switches simultaneously; but it is still a distributed system and, therefore, suffers from conventional complexities such as dropping packets, delaying of the control packets etc. Current platforms for SDN, such as NOX and Beacon, enable programming; but it is still hard to program them in a low level. With new protocols (such as OpenFlow) becoming more standard in industry, SDN is becoming easier to implement. The control plane generates the routing table while the data plane, utilizing the table to determine where the packets should be sent to [3]. Many companies utilize OpenFlow protocols within their data center networks to simplify operations. OpenFlow and SDN allow data centers and researchers to easily abstract and manage the large network.

The OpenFlow architecture typically includes the following 3 important components [8]–[10], [29].

1) Switches: OpenFlow defines an open source protocol to monitor/change the flow tables in different switches and routers. An OpenFlow switch has at least three components: a) flow table(s), each with an action field associated with each flow entry, b) a communication channel, which provides link for the transmission of commands and packets between

a controller and the switch, c) the OpenFlow protocol, which enables an OpenFlow controller able to communicate with any router/switch.

2) Controllers: A controller can update (revise, add, or delete) flow-entries from the flow table on behalf of the user's experiments. A static (versus dynamic) controller can be a simple software unit running on a computer to statically (versus dynamically) establish packet path between a group of test computers during a scientific experiment.

3) Flow-entries: Each flow-entry includes at least a simple action (network operation) for that flow item. Most OpenFlow switches support the following three actions: (a) sending this flow's packets to a port, (b) encapsulating this flow's packets and sending to a controller, and (c) dropping this flow's packets.

OpenFlow has gone through many standard iterations, and it is currently on version 1.3; however only version 1.0 is available for practical software and hardware design. The second and subsequent versions of OpenFlow changed the match structures so that the number and bit count of each header field could be specified. Thus new protocols would be easier to implement. In [21] a special controller is used to separate control bits from data bits, which allows for the network infrastructure to be shared more easily. A server is often utilized for the controller portion of OpenFlow architecture.

Currently, several projects are ongoing that utilize OpenFlow in both Europe and Japan [27], [28]. In Europe, eight islands are currently interconnected using OpenFlow. In Japan, there are plans to create a network compatible with the one in Europe, as well as a testbed that is much more widespread.

The existing OpenFlow standard assumes centralized control, that is, a single-point controller can manage all flow tables in different switches. This concept works very well in a small-scale, cable-based local area network. When OpenFlow was proposed, it was tested in a wired campus network. However, if many switches are deployed in a large area, it is difficult to use a single-point control. Especially when wireless media have to be used to connect long-distance devices, a central control becomes difficult since wireless signals fade away quickly for a long distance. Single control also has single-point failure issue. To solve the above issue, we can use distributed controllers in different locations. Each controller only manages the local switches. However, all controllers keep highly reliable communications for consistent view of the global status. As an example, HyperFlow [132] uses a logically centralized but physically distributed control panel to achieve a synchronized view of the entire SDN.

D. Beyond OpenFlow: Other SDN Standards

Besides OpenFlow (the most popular SDN protocol/standard), there exist other SDN implementations. For instance, IEEE P1520 standards have defined Programmable Network Interfaces [143]. It can be seen as an initial model of SDN, since it also has network programming abstractions.

ForCES (Forwarding and Control Element Separation) [144] is another standard defined by IETF. It consists of a series of RFCs for the coverage of different aspects on how to manage control and data forwarding elements. It proposes the models to separate IP control and data forwarding, Transport Mapping layer for the forwarding and control elements, logical function block library for such a separation, etc. However, ForCES does not have widespread adoption due to its lack of clear language abstraction definition and controller-switcher communication rules.

Note that ForCES has a key difference from OpenFlow: ForCES defines networking and data forwarding elements and their communication specifications. However, it does not change the essential network architecture. OpenFlow changes the architecture since it requires the routers/switches have very simply data forwarding function and the routing control functions should be removed to the upper level controllers. Therefore, OpenFlow cannot run in traditional routers that do not support OpenFlow standards, while ForCES can run in traditional devices since it just adds networking/forwarding elements.

SoftRouter [145] defines clearly the dynamic binding procedure between the network elements located in control plane (software-based) and data plane. In this standard, the network can be described in two different views, i.e., physical view and routing view. In the physical view, the network is made up of nodes Internetworked by media links. The nodes could be a forwarding element (FE) or a control element (CE). The FE is a common router without local sophisticated control logic. The CE is used to control FE. A CE is a general server. The routing view of a network reflects the network topology based on the concept of network element (NE). An NE is a logical grouping of network interfaces/ports and the corresponding CEs that control those ports. SoftRouter includes a few protocols: Discovery protocol (to establish a binding between FE and CE), FE/CE control protocol, and CE/CE protocol.

E. 1.5 SDN Applications

In this section we will provide some application examples on using SDN and OpenFlow.

1) Internet Research: Updating the Internet brings many challenges as it is constantly being used; it is difficult to test new ideas and strategies to solve the problems found in an existing network. SDN technologies provide a means for testing ideas for a future Internet without changing the current network [30]. Since SDN allows the control and data traffic to be separated with an OpenFlow switch, it is easier to separate hardware from software. This separation allows for experimenting with new addressing schemes so that new Internet architecture schemes can be tested.

Usually, it is difficult to experiment with new types of networks. Since new types of networks often utilize different addressing schemes and include other non-standard protocols, these changes are difficult to incorporate into existing networks. OpenFlow allows for routers, switches, and access points from many different companies to utilize the separation of the control and data planes. The devices simply forward data packets based on defined rules from the controller. If a data packet arrives and the device does not have a rule for it, the device forwards the packet to the controller that determines what to do with the packet, and if necessary, it sends a new rule to the device so that it can handle future data packets in the same manner [21].

2) Rural Connections: SDN simplifies complex data center and enterprise networks; it can further be utilized to simplify rural Wi-Fi networks. The main issues with rural environments include sparse populations, small profit margins and resource constraints, and others. SDN is beneficial because it separates the construction of the network and the configuration of the network by placing the control/management functionality into the central controller. This separation enables the rural infrastructure deployment business (which must be done locally in rural areas) and the Internet Service Provider (ISP) business (which is typically done remotely in cities) to be completely separated, i.e., those two businesses are operated by different entities [31], [32]. Therefore, SDN makes the management of rural networks much more convenient than traditional network architecture where the local network devices need customized control (which means the control of rural devices must be done in rural areas).

3) Date Centers Upgrading: Data centers are an integral part of many companies [33]. For example, Google has a large number of data centers so they can quickly provide data when requested. Similarly, many other companies utilize data centers to provide data to clients in a quick and efficient manner, but data centers are expensive to maintain. OpenFlow allows companies to save money in setting up and configuring networks since it allows switches to be managed from a central location [34].

Oftentimes, data center networks utilize proprietary architectures and topologies, which creates issues when merging different networks together; however there is often a need to merge two divergent networks. SDN brings a solution to this issue. In [33] the authors propose that a network infrastructure service based on OpenFlow be utilized to connect data center networks. They further state that these interconnected data center networks could solve problems with small latency by moving workload to underutilized networks. If a network is busy at a certain time of day, the workload might be able to be completed sooner in a network of a different time zone or in a network that is more energy efficient.

In [34] a data center model is created with a large number of nodes to test performance, throughput and bandwidth. The model included 192 nodes with 4 regular switches and 2 core switches with an OpenFlow controller. There was a firewall between the core switches, OpenFlow controller and the router. The authors also utilized an application called Mininet to prototype their network and test the performance. Mininet allows researchers to customize a SDN using OpenFlow protocols. Further, they utilized several tools to analyze their network setup including Iperf, Ping, PingAll, PingPair, and CBench. These tools allow people to check the possible bandwidth, connectivity, and the speed in which flows can be changed, respectively. Wireshark was also used to view traffic in the network.

4) Mobile Device Offloading: Privacy is important for business applications because people often work on data that needs to be kept secure. Some data can be sent among only a few people while other data does not require the same level of security. As an example, in [35] the authors utilized an Enterprise-Centric Offloading System (ECOS) to address these concerns. ECOS was designed to offload data to idle computers while ensuring that applications with additional security requirements are only offloaded on approved machines. Performance was also taken into consideration for different users and applications [35]. SDN is utilized to control the network and select resources. The resources selected must be able to meet the security requirements. The controller will determine if such a device is available for offloading that meets the security requirements while maintaining energy savings. If no such device exists, data is not allowed to be offloaded from the mobile device. If energy savings is not necessary, then any resource with enough capacity is utilized if available. OpenFlow switches are utilized so that the controller can regulate the flows. ECOS was able to offload while taking into account security requirements without an overly complex scheme.

5) Wireless Virtual Machines: Applications running on wireless virtual machines in businesses are becoming increasingly common. These virtual machines allow the companies to be more flexible and have lower operational costs. In order to extract the full potential from a virtual machine, there are needs for making them more portable. The main issue is how to maintain the virtual machine's IP address in the process. The current methods of handling virtual machines were not efficient. The solutions proposed in [36] include using a mobile IP or dynamic DNS. The main issue with both solutions is that someone has to manually reconfigure the network settings after removing the virtual machine. This limits businesses and data centers from easily porting their virtual machines to new locations.

An application named CrossRoads was developed by [36] in order to solve the mobility issue for virtual machines. CrossRoads is designed to allow mobility of both live and offline virtual machines. CrossRoads has three main purposes. The first purpose is to be able to take care of traffic from data centers as well as external users. The second purpose is to make use of OpenFlow with the assumption that each data center utilizes an OpenFlow controller. The third purpose is to make use of pseudo addresses for IP and MAC addresses in order to have the addresses remain constant when porting while allowing the real IP to change accordingly.

The basic implementation of their software was to create rules for finding the virtual machines in different networks. The CrossRoads controller would keep track of the real IP and MAC addresses for the controllers in each data center as well as the virtual machines in its own network. When a request is sent for an application running on a particular virtual machine, a request is broadcasted to the controllers. If the controller receives a request for a virtual machine that is not in its table, then it broadcasts the request to the other controllers; the controller who has the virtual machine's real IP address then sends out the pseudo MAC address to the original controller, and the original controller can update its table in case it gets another request in the near future.

Comparisons: SDN has been shown to be a valuable resource in many different types of applications. SDN allows users to quickly adapt networks to new situations as well as test new protocols. Table I shows the differences among some typical SDN applications. As one can see, OpenFlow was utilized in most of the applications for its versatility. Data centers continue to become an important part of the Internet and many

TABLE I
A COMPARISON OF DIFFERENT SDN APPLICATIONS

Applications	Use OpenFlow	Net. Traffic amount	For Data Center	Network Scalability	Mobile Platform	QoS support	For Cloud	Allow Hardware Change
Internet Research [30]	Yes	Depends on applications	Yes	Excellent	Yes	All QoS metrics	No	Yes
Rural Networking [31]	No	Low	No	Poor	No	Low throughput	No	No
Changing Silicon [32]	No	Medium	Yes	N/P	No	Low data rate	No	Yes
Data Centers [33]	Yes	High	Yes	Very Good	No	Latency is a concern	No	No
Cloud [34]	Yes	High	Yes	Excellent	Yes	Typically Real-time	Yes	No
Mobile Apps [35]	Yes	Low	No	Good	Yes	Long Delay	No	No
Virtual Machines [36]	Yes	Depends	Yes	Very Good	Yes	Real-time	Yes	No

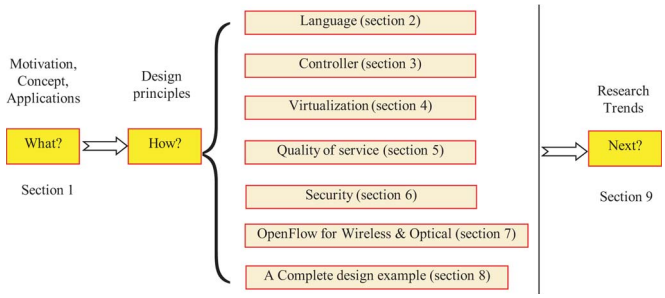


Fig. 3. Organization of this survey.

large companies. The column mobile applications refers to cell phones, tablets, and other non-traditional media formats rather than laptops and other typical computing platforms. A few of the applications utilize the cloud. Hardware changes are difficult to implement in conventional networks. This is mainly because they require a system to be shut down during upgrade. But SDN provides conveniences for such upgrades due to its separation of data and control planes.

F. Road Map

Fig. 3 shows the organization of this paper. After the concept is explained (Section I), Sections II–VIII will survey the most important aspects in SDN/OpenFlow design. Since SDN aims to enable easy re-policing, the network programming is a must (Section II). SDN simplifies all switches as data forwarders only and leave complex control in controllers (Section III). Due to the dynamic network resources deployment, it is critical to provide the users an accurate network resource management via the virtualization tools (Section IV). Then we move to the important SDN performance issue—QoS (Section V). We will explain different schemes that can support the QoS requirements. Any network has threats and attacks. SDN is not an exception. Section VI will explain the security and fault tolerance aspects in SDN designs. Then we introduce the ideas of implementing SDN/OpenFlow in two most important network types—wireless and optical networks (Section VII). Section VIII introduces a SDN design example. To help the readers understand unsolved challenging research issues, we will point out the next-step research directions in this exciting field (Section IX). Finally, Section X concludes the entire paper.

The reason of covering the three aspects (QoS, security, and wireless/optical) besides the basic SDN issues (Sections II–IV)

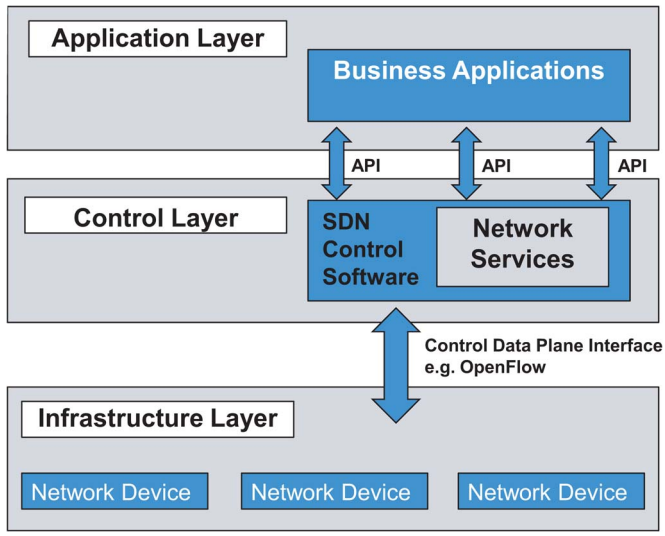


Fig. 4. Programming of the SDN and language abstraction.

is due to the following factors: First, for any new network architecture, the first concern is its performance, which mainly includes the end-to-end delay, throughput, jitter, etc. Therefore, it is critical to evaluate its QoS support capabilities. This is the reason that we use an individual section (Section V) to cover SDN's QoS support issues; Second, security is always a top concern for a user before he or she uses a new network model. There are many new attacks raised for any new network architecture. Therefore, we will use another section (Section VI) to cover SDN security considerations; Finally, today two most typical network media are wireless transmissions and optical fiber. SDN eventually needs to face the design challenges when used for those cases. Therefore, in Section VII we discuss SDN extensions in wireless and optical links.

II. LANGUAGE ABSTRACTIONS FOR SDN

A. Language Abstractions

In SDN the control function consists of two parts, i.e., the controller with the program and the set of rules implemented on the routing/switching devices (Fig. 4). This has an implication of making the programmer not worry about the low-level details in the switch hardware. The SDN programmers can just write the specification that captures the intended forwarding behavior of the network instead of writing programs dealing with the low-level details such as the events and the

forwarding rules of the network. This enables the interactions between the controllers and switches. A compiler transforms these specifications into code segments for both controllers and switches. As an example, a SDN programming tool called NetCore [69] allows descriptions of the network rules and policies which cannot be implemented directly on the switches. Another important fact about NetCore is that it has a clear formal set of rules that provide a basis for reasoning about program execution status.

Here, we introduce two important language abstractions in SDN programming.

1) Network Query Abstractions: In SDNs each switch stores counters for different forwarding rules. They are for the counts of the total number of packets and data segments processed using those rules. For traffic monitoring the controller has the ability to check different counters associated with different forwarding rules. This enables the programmers to monitor the fine details of implementation on the switches. This is a tedious job and makes the program complicated. Therefore, an added level of abstraction will help the programmers. To support applications whose correct operation involves a monitoring component, Frenetic [70] includes an embedded query language that provides effective abstractions for reading network state. This language is similar to SQL and includes segments for selecting, filtering, splitting, merging and aggregating the streams of packets. Another special feature of this language is that it enables the queries to be composed with forwarding policies. A compiler produces the control messages needed to query and tabulate the counters on switches.

2) Consistent Update Abstractions: Since SDNs are event-driven networks, the programs in SDNs need to update the data forwarding policy from time to time because of the changes in the network topology, failures in the communication links, etc. An ideal solution is the automatic update of all the SDN switches in one time; but in reality it is not easy to implement. One good solution is to allow certain level of abstraction, and then send these changes from one node to another. An example is the per-packet consistency which ensures that each packet just uses the same, latest policy (instead of a combination of both the old and new policy). This preserves all features that can be represented by individual packets and the paths they take through the SDN. Those properties subsume important structural invariants such as basic connectivity and free-of-loop, and link access control policies. Per-flow consistency ensures that a group of related packets are processed with the same flow policy. Frenetic provides an ideal platform for exploring such abstractions, as the compiler can be used to perform the tedious bookkeeping for implementing network policy updates [70].

B. Language Abstraction Tools: Frenetic Project

SDN requires efficient language abstraction tools to achieve network re-programming. As an example, the Frenetic project aims to provide simple and higher level of abstraction with three purposes, i.e., (i) Monitoring of data traffic, (ii) Managing (creating and composition) packet forwarding policies, (iii) Ensuring the consistency when updating those policies [71]. By providing these abstractions the network programming becomes easy and efficient without a need of worrying about the low-level programming details.

Frenetic project utilizes a language that supports an application-level query scheme for subscribing to a data stream. It collects information about the state of the SDN, including traffic statistics and topology changes. The run-time system is responsible for managing the polling switch counters, gathering statistics, and reacting to the events. In the Frenetic project the specification of the packet forwarding rules in the network is defined by the use of a high-level policy language which can easily define the rules and is convenient to programmers. Different modules can be responsible for different operations such as the routing, discovery of the topology of the network, workload balancing, and access control, etc. This modular design is used to register each module's task with the run time system which is responsible for composing, automatic compilation and optimization of the programmer's requested tasks. To update the global configuration of the network, Frenetic project provides a higher level of abstraction. This feature enables the programmers to configure the network without going physically to each routing device for installing or changing packet forwarding rules. Usually, such a process is very tedious and is prone to errors. The run-time system makes sure that during the updating process only one set of rules is applied to them, i.e., either the old policy or the new one but not both of the rules. This makes sure that there is no violations for the important invariants such as connectivity, control parameters of the loops and the access control when the Open-Flow switches from one policy to another [71].

To illustrate Frenetic language syntax, here we use an example. In MAC learning applications, an Ethernet switch performs interface query to find a suitable output port to deliver the frames. Frenetic SQL (Structure Query Language) is as follows:

Select (packets) $*$
GroupBy ([srcmac]) $*$
SplitWhen ([inport]) $*$
Limit (1)

Here *Select(packets)* is used to receive actual packets (instead of traffic statistics). The *GroupBy([srcmac])* divides the packets into groups based on a header field called *sercmac*. Such a field makes sure that we receive all packets with the same MAC address. *SplitWhen([inport])* means that we only receive the packets that appear in a new ingress port on the switch. Limit(1) means that the program just wants to receive the first packet in order to update the flow table in data plane.

In a nut shell, Frenetic language project is an aggregation of simple yet powerful modules that provide an added level of abstraction to the programmer for controlling the routing devices. This added layer of abstraction runs on the compiler and the run time system, and is vital for the efficient code execution.

C. Language Abstraction Tool: FlowVisor

The virtualization layer helps in the development and operation of the SDN slice on the top of shared network infrastructures. A potential solution is the concept of Auto-Slice [73]. It provides the manufacturer with the ability to redesign the SDN for different applications while the operator

intervention is minimized. Simultaneously the programmers have the ability to build the programmable network pieces which enable the development of different services based on the SDN working principles.

Flow Visor is considered to be a fundamental building block for SDN virtualization and is used to partition the data flow tables in switches using the OpenFlow protocol by dividing it into the so-called flow spaces. Thus switches can be manipulated concurrently by several software controllers. Nevertheless, the instantiation of an entire SDN topology is non-trivial, as it involves numerous operations, such as mapping virtual SDN (vSDN) topologies, installing auxiliary flow entries for tunneling and enforcing flow table isolation. Such operations need a lot of management recourses.

The goal is to develop a virtualization layer which is called SDN hypervisor. It enables the automation of the deployment process and the operation of the vSDN topologies with the minimum interaction of the administrator. vSDNs focuses on the scalability aspects of the hypervisor design of the network. In [74] an example is presented in which a network infrastructure is assumed to provide vSDN topologies to several tenants. The vSDN of each tenant takes care of a number of things such as the bandwidth of the link, its location and the switching speed (capacity), etc. The assumption is that every tenant uses switches that follow OpenFlow protocol standards with a flow table partitioned into a number of segments. The proposed distributed hypervisor architecture has the capability of handling a large amount of data flow tables for several clients. There are two very important modules in the hypervisor: Management Module (MM) and Multiple Controller Proxies (CPX). These modules are designed in such a manner that it distributes the load control over all the tenants.

The goal of the MM portion is to optimize global parameters. The transport control message translation is used to enable the tenants to have the access to the packet processing set of rules within a specific SDN layer without having to disturb the simultaneous users. Upon the reception of a request, MM inquires the vSDN about the resources available in the network with every SDN domain and then accordingly assigns a set of logical resources to each CPX.

As a next step each CPX initializes the allocated segment of the topology by installing flow entries in its domain, which unambiguously bind traffic to a specific logical context using tagging. As the clients are required to be isolated from each other, every CPX is responsible to do a policy control on the data flow table access and make sure that all the entries in these tables are mapped into segments that are not overlapping. CPX is responsible for controlling the routing switches. Also the CPX takes care of all the data communication between the client controller and the forwarding plane.

A new entry into the switch has to follow certain steps (Ididnotseemanysteps). First, the proxy creates a control message for addition of new entry into the switch flow table in such a manner that all references (addresses) to memories are replaced by the corresponding physical entries, and corresponding traffic controlling actions are added into the packet. The Proxy is responsible for maintaining the status of each virtual node in a given SDN. As a result the CPX has the ability to independently transfer virtual resources within its domain to optimize inter-domain resource allocation.

If there are a number of clients in the network, a large number of flow tables are needed in the memory of a routing switch. The task of CPX is to make sure that all the flow tables are virtually isolated, all packet processing takes place in a correct order, and all the actions are carried out in case a connected group of virtual nodes is being mapped to the same routing device.

In the OpenFlow routing devices, there is a problem on the scalability of the platform due to the large flow table size. There could be a large number of entries in the flow table. To deal with such situation, an auxiliary software data paths (ASD) is used in the substrate network [74]. For every SDN domain, an ASD is assigned. The server has enough memory to store all the logical flow tables which are needed by the corresponding ASD compared to the limited space on the OpenFlow routing devices. Although the software-based data path has some advantages, there is still a huge gap between the OpenFlow protocol and the actual hardware components. To overcome these limitations, the Zipf property of the aggregate traffic [75], i.e., the small fraction of flows, is responsible for the traffic forwarding. In this technique ASDs are used for handling heavy data traffic while only a very small amount of high volume traffic is cached in the dedicated routing devices.

Language example of FlowVisor: Here, we provide an example on how FlowVisor creates a slice.

```
# Topology
Example_topo = nxtopo.NXTopo ( )
Example_topo.add_switch (name = "A", ports [1,2,3,4])
Example_topo.add_switch (name = "B", ports [1,2,3,4])
Example_topo.add_link (("A", 4), ("B", 4))
# Mappings
P_map = "A": "S2", "B": "S3"
Q_map = identity_port_map (Example_topo, P_map)
Maps = (P_map, Q_map)
# predicates
Preds = \
([(p, header ("srcport", 80))
For p in Example_topo.edge_ports ("A") +
[(p, header ("dstport", 80))
For p in Exam_topo.edge_ports ("B")])
# slice constructor
Slice = Slice (Example_topo, phys_topo, maps, preds)
```

In the above example, we first define a network topology called Example_topo, which has two switches: A and B. The switches have 3 edge ports each. Then we define the switch→ port mappings. Switch A maps to S2, and B maps to S3. Then we associate a predicate with each edge port. The predicates can map traffic (web only) to the slice. The last line officially creates a slice [138].

III. Controller

The control plane can be managed by a central controller or multiple ones. It gives a global view of the SDN status to upper application layer. In this section, we look into the

architecture and performance of controller in software defined networks.

A. Types of Controllers

While SDN is suitable for some deployment environments (such as homes [76], [77], data centers [78], and the enterprise [79]), delegating control to a remote system has raised a number of questions on control-plane scaling implications of such an approach. Two of the most often voiced concerns are: (a) how fast the controller can respond to data path requests; and (b) how many data path requests it can handle per second. For software controller, there are four publicly-available OpenFlow controllers: NOX, NOX-MT, Beacon, and Maestro [80].

A typical OpenFlow controller is NOX-MT [80]. NOX [48] whose measured performance motivated several recent proposals on improving control plane efficiency has a very low flow setup throughput and large flow setup latency. Fortunately, this is not an intrinsic limitation of the SDN control plane: NOX is not optimized for performance and is single-threaded.

NOX-MT is a slightly modified multi-threaded successor of NOX. With simple tweaks we are able to significantly improve NOX's throughput and response time. The techniques used to optimize NOX are quite well-known: I/O batching to minimize the overhead of I/O, porting the I/O handling harness to Boost Asynchronous I/O (ASIO) library (which simplifies multi-threaded operation), and using a fast multiprocessor-aware malloc implementation that scales well in a multi-core machine.

Despite these modifications, NOX-MT is far from perfect. It does not address many of NOX's performance deficiencies, including but not limited to: heavy use of dynamic memory allocation and redundant memory copies on a per-request basis, and using locking while robust wait-free alternatives exist. Addressing these issues would significantly improve NOX's performance. However, they require fundamental changes to the NOX code base. NOX-MT was the first effort in enhancing controller performance. The SDN controllers can be optimized to be very fast.

B. Methods to Enhance Controller's Performance

We can make OpenFlow network more scalable by designing a multi-level controller architecture. With carefully deployed controllers, we can avoid throughput bottleneck in real networks. For example, in [81] authors have measured the flow rate in a HP ProCurve (model # 5406zl) switch, which is over 250 flows per second. In the meantime, in [82] authors reported that for a data center with over 1000 servers, it could face a flow arrival rate of 100 k flows/second, and in [83] they reported a peak rate of 10 M flows per second for an 100-switch network. The above example shows that current switches cannot handle the application flow rate demands. Therefore, we need to invent an efficient protocol which can minimize the switch-to-controller communications.

The data plane should be made simple. Currently OpenFlow assigns routing tasks to the central controller for flow setup. And the low-level switches have to communicate with the controller very frequently in order to obtain the instructions on how to handle incoming packets. This strategy can consume the controller's processing power and also congest switch-controller links. Eventually they cause a serious bottleneck in terms of the scalability of OpenFlow.

However, recent measurements of some deployment environments suggest that these numbers are far from sufficient. This causes relatively poor controller performance and high network demands to address perceived architectural inefficiencies. But there has been no in-depth study on the performance of a traditional SDN controller. Most results were gathered from systems that were not optimized for throughput performance. To underscore this point, researchers were able to improve the performance of NOX, an open source controller for OpenFlow networks, by more than 30 times in throughput [84].

In most SDN designs the central controller(s) can perform all the programming tasks. This model certainly brings the scalability issue to the control plane. A better control plane should be able to make the packet handling rate scalable with the number of CPUs. It is better to always have the network status in packet level available to the controllers. Study from Tootoonchian *et al.* [84] implements a Glasgow Haskell Compiler (GHC) based runtime system. It can allocate/deallocate memory units, schedule different event handlers, and reduce the interrupts or system calls in order to decrease the runtime system load. They have showed the possibility of using a single controller to communicate with 5000 switches, and achieving the flow rate of up to 14 M per second! The switch-controller communication delay is less than 10 ms in the worst case. In [79] a partition/aggregate scheme is used to handle TCP congestion issue.

C. Advanced Controller Design

Here, we introduce an advanced method for high-speed control functions in control plane. In [140], a mechanism called Control-Message Quenching (CMQ) is proposed to reduce the flow setup delay and improve the SDN throughput among switches/routers. There are huge number of flows that need to be handled by the controllers. The inability of OpenFlow to process so many flows' policy management is due to the inefficient design of control-data plane interfaces. Especially, there exist frequent switch-controller communications: the switches have to consult the controller frequently for instructions on how to handle new incoming packets.

The basic idea of CMQ is to ask any switch to send only one packet-in message during each RTT (round-trip-time), for each source-destination pair, upon multiple flow table misses. Thus we do not need to bother the controllers each time we receive the packets with the same source/destination. Each switch should maintain a dynamically updated table with all learned, unique source-destination pairs. For each incoming packet that cannot find its source-destination pair, i.e., table-miss occurs, the switch will insert such a new pair into the table, and query the controller. Such a pair table will be maintained periodically in case the network topology changes, which can detected by the control plane.

A problem with existing SDN controller is that the SDN flow tables typically cannot scale well when there are more than 1000 entries [141]. This is mainly because the tables often include wildcards, and thus need ternary content-addressable memory (TCAM), as well as complex, slow data structures. In [141] a scheme called Palette, can decompose a large SDN table into small ones and distribute them to the whole SDN without damaging the policy semantics. It can also reduce the table size by sharing resources among different flows. The graph-theory based on model is used to distribute the small tables to proper switches.

There could exist multiple controllers in the SDN. In [142] a load balancing strategy called BalanceFlow, is proposed to achieve controller load balancing. Through cross-controller communications, a controller is selected as super-controller, which can tune the flow requests received by each controller without introducing much delay. Note that each controller should publish its load information periodically to allow super-controller to partition the loads properly.

IV. Network Virtualization

A. Virtualization Strategies

As technology develops, the modern network becomes larger and more capable of providing all kinds of new services. The cloud computing, and some frameworks such as GENI, FIRE, G-Lab, F-Lab and AKARI, utilize the large-scale experimental facilities from networks. However, resources are always limited and users' demands keep increasing as well. The sharing of network hardware resources among users becomes necessary because it could utilize the existing infrastructure more efficiently and satisfy users' demands. Network virtualization in SDN is a good way to provide different users with infrastructure sharing capabilities [85]. The term OpenFlow often comes with network virtualization these years. The FlowVisor, the controller software, is a middleware between OpenFlow controllers and switches. FlowVisor decomposes the given network into virtual slices, and delegates the control of each slice to a specific controller [86].

Both OpenFlow and FlowVisor have their limitations in terms of network management, flexibility, isolation and QoS. OpenFlow offers common instructions, but lacks standard management tools. FlowVisor only has access to the data plane, so the control plane and network controllers have to be managed by the users of the infrastructure. On the other hand, it can ensure a logical traffic isolation but with a constant level, which means that it lacks flexibility. Facing these challenges, researchers try to establish their own architecture based on OpenFlow or FlowVisor for an improved network virtualization.

FlowVisor can be pre-installed on the commercial hardware, and can provide the network administrator with comprehensive rules to manage the network, rather than adjusting the physical routers and switches. FlowVisor creates slices of network resources and acts as the controlling proxy of each slice to different controllers as shown in Fig. 5. The slices may be switch ports, Ethernet addresses, IP addresses, etc, and they are isolated and cannot control other traffic. It can dynamically

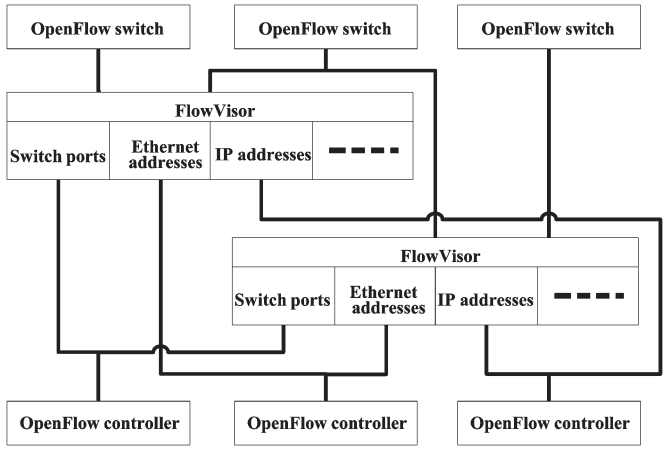


Fig. 5. The FlowVisor acts as proxy and provides slices.

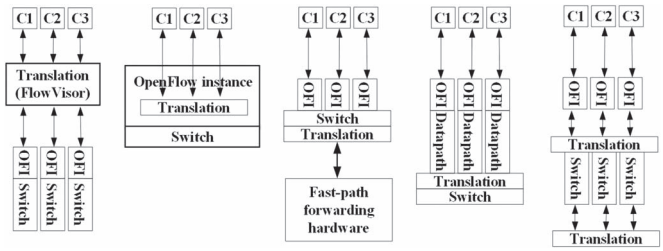


Fig. 6. Various translating functions (C1,C2,C3: different Controllers; OFI—OpenFlow Instance).

manage these slices and distribute them to different OpenFlow controllers, and enables different virtual networks to share the same physical network resources.

B. Virtualization Models

In the context of OpenFlow there are different virtualization models in the view of translation model [87] (Fig. 6). Translation aims to find 1 : 1 mapping relationship between the physical SDN facilities and the virtual resources. The translation unit is located between the application layer and the physical hardware. According to their placements we could classify them into five models.

1) FlowVisor: FlowVisor is the translation unit that delegates a protocol and controls various physical switches or controllers. It has full control of the virtualization tasks.
2) Translation unit: it is in the OpenFlow instance of the switch, and it performs translation among different controllers at the protocol level.
3) Multiple OpenFlow instances running on one switch are connected to one controller. Translation is executed between the data forwarding unit (such as a switch) and an OpenFlow instance.
4) Multiple OpenFlow instances still running on a single switch, but the switch's datapath is partitioned into a few parallel ones, one per instance. It translates by adjusting the ports connected to the different parallel data paths.
5) Multiple translation units are used, and at least one is for virtualization on the switch level, and another one for interconnecting some virtual switches.

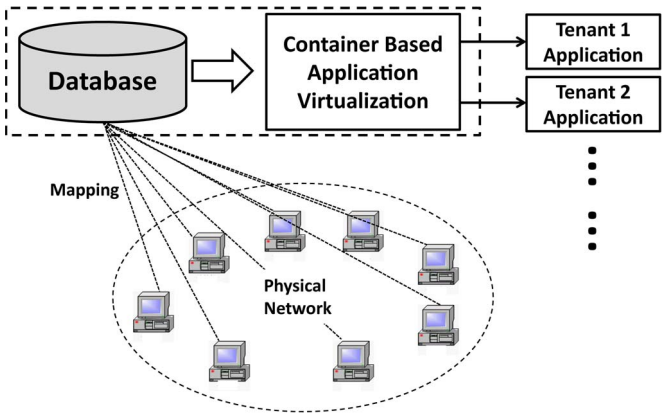


Fig. 7. System design of FlowN.

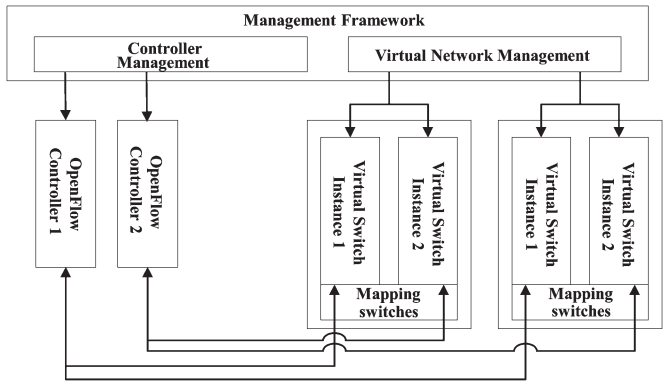


Fig. 8. Integrated OpenFlow management framework.

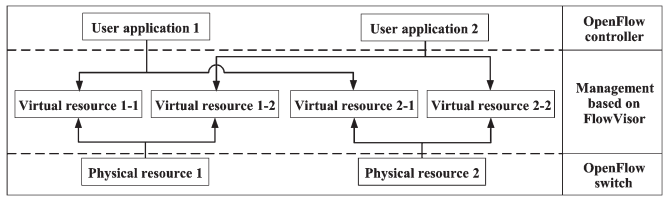


Fig. 9. OpenFlow network virtualization for Cloud computing.

C. Virtualization Architectures

Some systems have been proposed to address the OpenFlow-based network virtualization limitations. These methods can be classified as three types: (1) Improve the OpenFlow controller. OpenFlow controller is a software, and it can be modified by users to satisfy their special demands. (2) Improve the FlowVisor. The FlowVisor itself already has basic management function, and it can be improved to overcome some limitations. (3) To add new abstraction layer upon OpenFlow switch. Researchers add new layers or new components to manage the virtual network. In the following we will focus on some performance requirements for a SDN virtualizer.

1) Flexibility: The flexibility in the network virtualization denotes the scalability and the control level to the network. It usually conflicts with the isolation demand.

In [85] it present a system called FlowN that extends the NOX version 1.0 OpenFlow controller, and embeds a MySQL version 14.14 based database with the virtual-to-physical mappings as shown in Fig. 7. This FlowN is a scalable virtual network and provides tenants a full control of the virtual network tenants can write their own controller application and define arbitrary network topology. With the container based architecture, the controller software that interacts with the physical switches is shared among tenant applications, and so that the resources could be saved when the controller becomes more and more complex these days.

This system is evaluated in two experiments by increasing the number of the nodes: one measures the latency of the packets arriving at the controller, and the other measures the fault time of the link used by multiple tenants. When the number of nodes is large, the system has the similar latency as FlowVisor does but is more flexible; and its fault time could be small even the number of network nodes is large.

In [88] an efficient network virtualization framework is proposed. Its major features include: (1) monitor multiple instances of OpenFlow switches, (2) set up controllers and SDN applications, and (3) achieve QoS performance. It can easily configure the parameters of different switches, and monitor the network topology to see any node changes. It uses OpenNMS as the management tool since it is open source. It has virtual controller management as shown in Fig. 8. The prototype is successfully tested on the testbed consisting of six PCs, one switch and one OpenFlow switch.

A MAC layer network virtualization scheme with new MAC addressing mode is proposed in [89]. Since it uses a centralized MAC addressing, it could overcome the SDN scalability problems. This system efficiently supports Cloud computing and sharing of the infrastructures as shown in Fig. 9.

The virtualization of the LANs could be used to virtualize the network, but it has more complexity and overhead, and is not good at scalability. Thus the virtualization of MAC layer functions could be used, and is realized in [89] by reserving part of the remaining MAC address for the virtual nodes. This system reduces IP and control overhead, but the security issues need to be solved. Details of the system are provided, but the prototype is not tested in experiment.

2) Isolation: In order to ensure all the tenants of the virtual network can share the infrastructure without collision, the isolation problem must be addressed. The isolation may be in different levels or places, just like address space. A research network named EHU-OEF is proposed in [86] (Fig. 10). This network uses L2PNV, which means Layer-2 Prefix-based Network Virtualization, to separate various resource slices and allows users to have multiple virtual networks based on the MAC address settings. L2PNV has made some specific flow rules as well as some customized controller modules. It can also change FlowVisor.

EHU-OEF can well isolate different slices in the flow table, and the flow traffic can be distinguished based on the MAC addresses. Moreover, the NOX controllers use their module ecosystem to easily manage different slices. This solution has the benefit since it can deal with longer MAC header such as in virtual LAN (VLAN) cases. It can also be used to test other non-IP protocols by simply changing the addressing schemes. The EHU-OEF prototype is tested on the platform composed of seven NEC switches (IP8800/S3640), four Linksys WRT54GL,

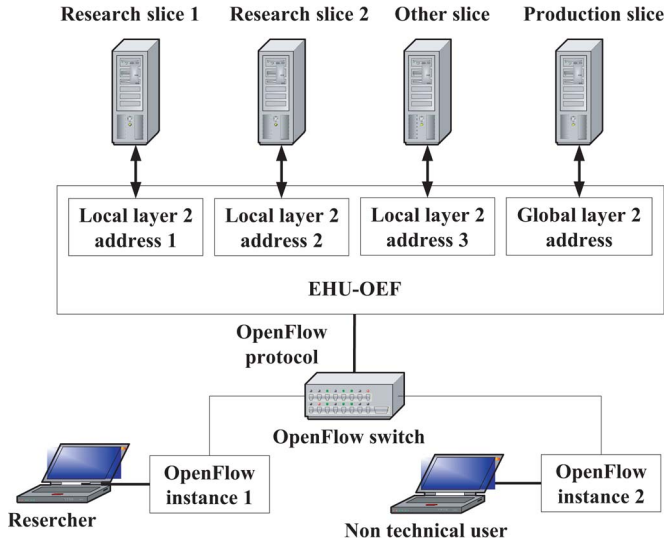


Fig. 10. EHU-OEF: an integrated OpenFlow management framework.

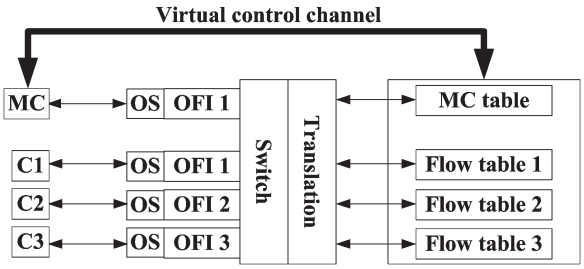


Fig. 11. A Full virtualization system. (MC: master controller; C1, C2, C3: regular controllers; OS: operating system; OFI: OpenFlow instance) [87].

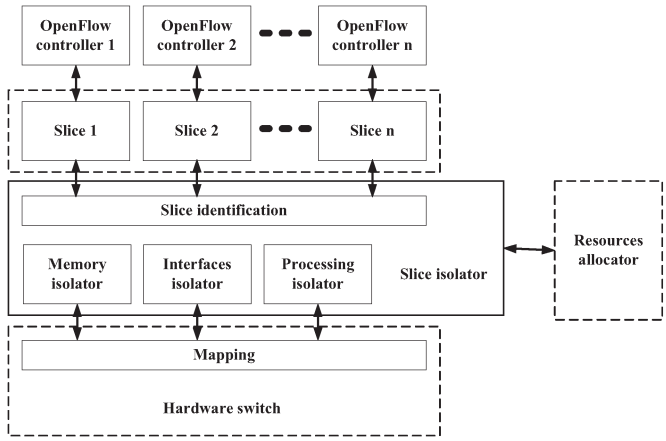


Fig. 12. Network virtualization using Slice Isolator [90].

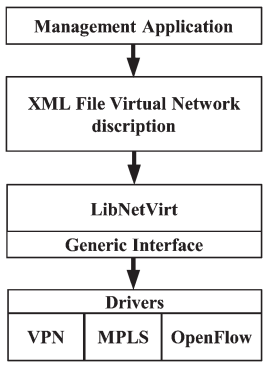


Fig. 13. LibNetVirt architecture.

and two NetFPGAs. It is the first OpenFlow-based SDN infrastructure in Europe and allows experimental and application-oriented data traffic in the same network without conflict.

In [87] a SDN virtualization system is proposed with fair resource allocation in the data/control planes as shown in Fig. 11. All SDN tenants obtain the network resource by enforcing the resource allocations in the central controller, the datapath of the forwarding elements, and the control channel between the switch and the controller. The QoS tools are applied to make fair resource allocation. It provides strict isolation between different sub-domains in a large SDN. It also allows future protocol extensions. However, there is no prototype tested in the system.

In [90] the isolation issue is solved among slices in different virtual switches. It makes all slices share the network resources in a fair way while allowing the isolation adaptation according to the expected QoS performance. It also allows multi-level isolation (see Fig. 12). A Slice Isolator is located above the switches and OpenFlow abstraction layer, and is designed as a model focusing on (a) Interface isolation; (b) Processing isolation; and (c) Memory isolation.

Evaluations of the system show that the isolation levels have significant impact on the performance and flexibility. The time for reconfiguring the hardware traffic manager increases fast when the isolation level goes up. High isolation level also leads to latency. So the best isolation level can be determined based on the update time and latency to achieve required performance.

3) Efficient Management: Network virtualization management is involved with the mapping, layer abstraction or system design to make sure the virtualized network can satisfy different demands. It is the integration of the flexibility, isolation, and convenience. A network virtualization architecture allowing management tools to be independent of the underlying technologies is presented in [91]. The paper proposes an abstraction deployed as a library, with a unified interface toward the underlying network specific drivers. The prototype is built on top of an OpenFlow-enabled network as shown in Fig. 13. It uses the single router abstraction to describe a network, and has feasibility for creating isolated virtual networks in a programmatic and on-demand fashion. In this system the management tools can be independent of the working cloud platform so that different technologies can be integrated, and the system focuses on reduce the time of creating the virtual network. The prototype named LibNetVirt is separated in two different parts: generic interface and drivers. The generic interface is a set of functions that allow interacting with the virtual network and executing the operations in the specific driver. A driver is an element that communicates to manipulate the VN in the physical equipment.

A scheme [92] as shown in Fig. 14, enables the creation of different isolated, virtual experimental sub-systems based on the same physical infrastructure. This system implements a novel optical FlowVisor, and has cross-layer for management and high isolation for multiple users.

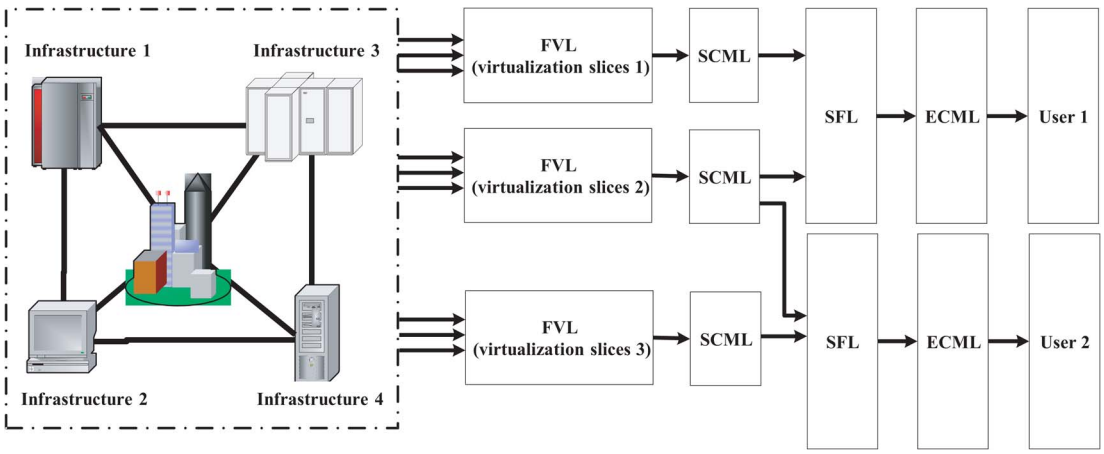


Fig. 14. Cross-layer experimental infrastructure virtualization.

TABLE II
THE COMPARISON OF THE REPORTED NETWORK VIRTUALIZATION SYSTEMS

System	Flexibility	Isolation	Management	FlowVisor
FlowN [85]	Very High	Very good	Not so hard	−(means it has equivalent functions to FlowVisor)
Integrated system [88]	High	Good	Very easy	−
MAC addressing system [89]	High	Good	Not so hard	√(means it has better performance than FlowVisor)
EHU-OEF [86]	Very High	Excellent	Easy	√
Adaptable isolation system [90]	High	Excellent	Easy	−
LibNetVirt [91]	General	Average	Very Easy	−
Cross-layer experimental infrastructure virtualization [92]	General	Average	Very Easy	√

This architecture provides several abstraction layers for the management: (a) The Flexible Infrastructure Virtualization Layer (FVL) is composed of virtualized slicing and partitioning of the infrastructure. (b) The Slice Control and Management Layer (SCML) can monitor the status of slices. (c) The Slice Federation Layer (SFL) can aggregates multiple slices into one integrated experimental system. (d) The Experiment Control and Management Layer (ECML) aims to set up experiment-specific slice parameters. It uses extended OpenFlow controller to achieve various actions.

The architecture is tested on the platform composed of eight NEC IP8800 OpenFlow-based switches and four Calient DiamondWave optical switch. The result shows that the setup time of establishing the flow path increases even for a large number of hops.

There are other aspects of the network virtualization designs. We compare the above discussed systems with respect to their focus points in Table II.

FlowVisor becomes the standard scheme of the network virtualization, so we compare these presented systems with FlowVisor (the last column). Most of the presented systems, no matter whether it is based on FlowVisor or it is built totally in a new scheme, not only have equivalent abilities to FlowVisor, but have one or more advantages over FlowVisor such as flexibility, adjustable isolation levels, etc.

D. Discussions

Network virtualization not only enables infrastructure sharing, but also provides better ways to utilize the infrastructure or to reduce the cost. Virtualization can greatly reduce the network upgrading cost for large-scale wireless or wired infrastructures. For example, a mobile network virtualization scheme is designed in [93]. It has lower cost than classical network and SDN network. A case study with a German network is given there. The considered capital expenditures can be reduced by 58.04% when using the SDN-based network instead of the classical one. A qualitative cost evaluation shows that the continuous cost of infrastructure, maintenance cost, costs for repair, cost of service provisioning are lower.

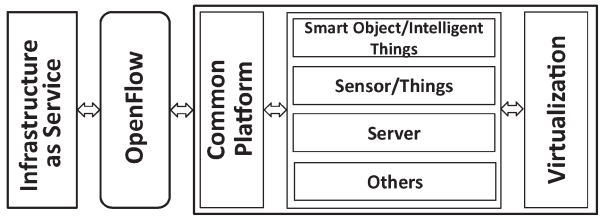


Fig. 15. Abstraction layers of the virtual network [94].

It is reported in [94] that the OpenFlow-based micro-sensor networks (its network components are shown in Fig. 15) can be seamlessly interfaced to the Internet of Things or cloud computing applications. In traditional sensor networks, some sensors away from the access point may not be reached. However, by using the virtualization we form a new concept called flow-sensors, which enables smooth data transfer between all sensors. A flow-sensor is a sensor with local flow table and wireless communications to controllers. Fig. 16 shows an example of the advantages of a flow sensor network over a conventional sensor

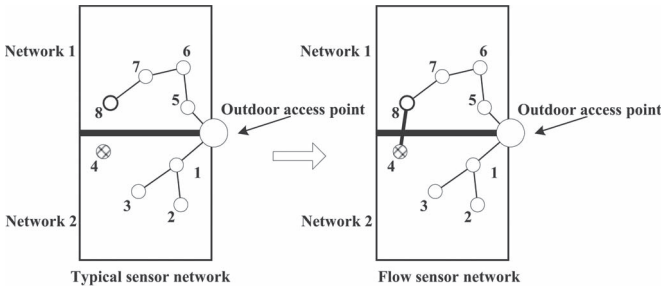


Fig. 16. Typical sensor network and flow sensor network [94].

network. In a conventional sensor network, the sensors 1 and 2 cannot communicate with each other without the access point, so node 4 is too far and is lost; within the flow sensor network, node 4 can talk to node 8, so that node 4 can be accessed. In [94] it shows that the flow sensor can have 39% higher reachability than a common sensor. This is extremely useful in large-scale sensor network (>100 sensors).

V. Quality of Service (QoS)

In past decades, the Internet Engineering Task Force (IETF) has defined two types of Quality of Service (QoS) architectures, IntServ (integrated services) and Diffserv (differentiated services). The IntServ is difficult to implement in today's large networks due to too much operation overhead in different routers. OpenFlow can provide fine-granularity QoS support (delay, jitter, throughput, etc.) [101]. This is because OpenFlow can well control packet-level or flow-level data delivery via its controllers. Such a fine-granularity means that OpenFlow allows the users to specify how to handle individual flows, which corresponds to IntServ in IETF definitions. Of course the user can also aggregate individual flows into classes (i.e., Diffserc). As a matter of fact, OpenFlow provides a series of programming tools to create/recycle slices (a slice is a virtual flow). The user can define how to allocate network resources (queues, routers, switches, etc.) to different slices with different priorities.

There are very few works targeting SDN QoS supporting issues. Among the few QoS models in SDN/OpenFlow, OpenQoS [95], [96] is one of the most typical solutions. It has a comprehensive controller architecture to support scalable video streaming in SDNs. We therefore summarize its principle first. Later on we will survey other QoS supporting schemes such as special operating system support for SDN QoS, QoSFlow, and so on.

A. OpenFlow QoS Model

Streaming multimedia applications such as Internet conferencing, IPTV, etc., all require a strict QoS (delay/jitter) control. As an example, the Scalable Video Coding (SVC) [100] encodes a video segment into two parts: a base layer and one or more enhancement layers. It is important to guarantee the QoS of the base layer since it has the detailed pixel information. However, current Internet structure cannot achieve high QoS for base layers due to hard-to-control TCP connections. Moreover, Internet tends to search the shortest path. Once that shortest path is congested, a large percentage of packets are dropped. However, OpenFlow does not stick to the shortest path. By programming the controllers, we can easily adjust the flow delivery rules. In [95] they proposed an OpenFlow-based video delivery scheme which uses dynamic QoS model to guarantee the best QoS for SVC base layer data.

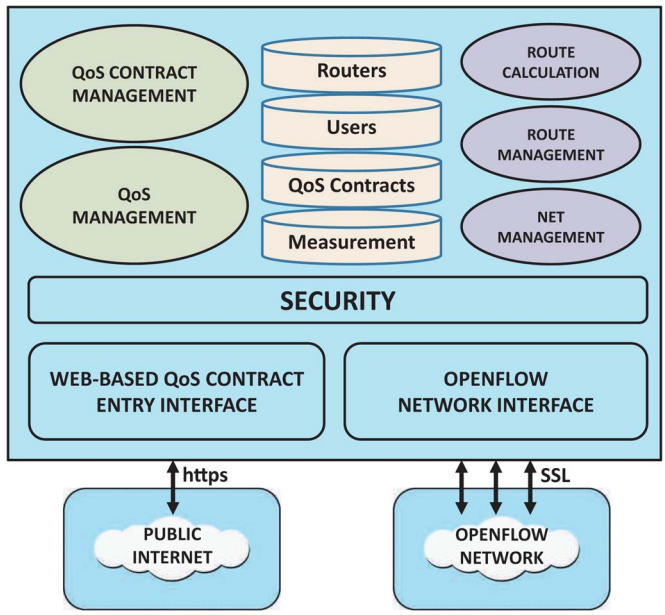


Fig. 17. Controller subsystems to support QoS [95].

QoS Optimization Model: In [95] an interesting OpenFlow QoS model is proposed. The basic principle is as follows: it formulates the dynamic QoS routing as a Constrained Shortest path (CSP) problem. For video applications, it employs delay variation as the constraint in the optimization function. It first represent the entire SDN as a simple graph. It then defines a cost function based on the delay variation constraint. The CSP problem aims to find the best path to minimize the cost function. To meet the packet loss constraint, it also defines a combined constraint with the weighted sum of packet loss measure and delay variation. The solution supports both level-1 and level-2 QoS routes. Its results show that the average quality of video streams is improved by 14% if only the base layer is rerouted. By rerouting the video data in the enhancement layer together with the base layer, the quality is further improved by another 6.5%.

B. Controller Architecture for QoS Optimization

The controller proposed in [96] has the functions of route calculation and route management. Fig. 17 illustrates the controller architecture with various sub-functions. The controller has powerful capabilities to specify QoS requirements. It can also directly control the flow table in order to differentiate between different priorities of traffic. The communications between the controller and the switches may be secured by some standards such as SSL.

Note that the forward layer has to implement the policing functions in order to ensure that the clients obey the Service Level Agreements (SLAs) specified in their QoS contracts. The following three extra features should exist in the above

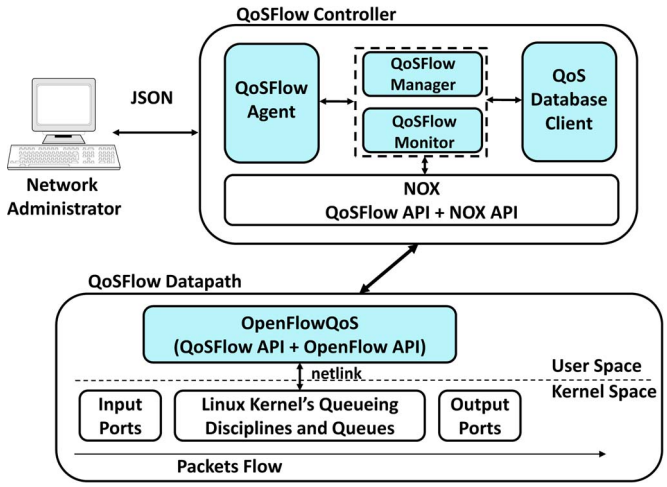


Fig. 18. QoSFlow modules [134].

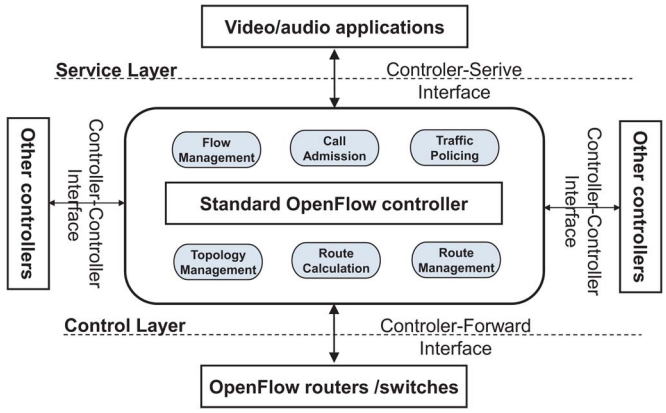


Fig. 19. QoSFlow controller architecture [134].

architecture: (1) Resource monitoring: The forwarders should comprehensively monitor their available network resources and report periodically to the controller. The controller may poll the forwarder for such profile. (2) Resource signaling: Each forwarder should use signaling messages to communicate with the controller on the current resource consumption so that certain actions can be taken by the controller, such as updating the flow table, changing QoS parameters, etc. (3) Resource reservation: From time to time the controller may command a forwarder to reserve certain resources for future QoS needs [99]. This includes the reservation of buffer size, memory space, CPU calculation time, and other resource requirements.

C. QoSFlow Architecture

In its current version, OpenFlow is not able to configure QoS parameters in a dynamic and on-demand manner (i.e., it does this manually). In order to deal with QoS problems in dynamic approach, a framework called QoSFlow (Fig. 18) that enables QoS management in OpenFlow environment is proposed in [134]. QoSFlow allows the management of traffic class and queues through rules or policy. It manages QoS resources (e.g., bandwidth, queue size) without changing the SDN architecture. All actions are invoked by an OpenFlow controller and in a dynamic and on-demand manner (not manually).

QoSFlow is an extension of the standard OpenFlow controller which provides multimedia delivery with QoS. The QoSFlow controller is based on NOX, which is responsible for managing/monitoring actions and controlling signaling messages. The new controller, besides NOX API, contains the following new components: QoSFlow agent, QoSFlow manager, QoSFlow monitor, and DB-QoSFlow client. These four modules have been designed to extend the NOX API with QoS features called QoSFlow API. QoS Agent is responsible for creating a communication module between an administrator management tool and the other two QoSFlow components: the manager and monitor QoSFlow. By using JSON interface, the agent is able to receive policies, manage or monitor commands from a third-part administrator application. The QoSFlow monitor and manager components, respectively, monitor and manage the QoS of OpenFlow domains. Fig. 19 shows its controller architecture.

The QoSFlow data-path component is responsible for creating all low-level actions on the switch ports. This component allows OpenFlow to get all the required information to run management commands created by either the administrator's tool or through header packet information. In QoS management tool, the actions are processed in the QoSFlow Agent. When receiving those actions, it checks the type of the received requests in order to select the next procedure. This new message is automatically sent to controllers through NOX. The QoS actions can be applied automatically through the packet header information. In order to support fine-granularity QoS, the incoming traffic is grouped as data flows and multimedia flows, where the multimedia flows are dynamically placed on QoS guaranteed routes and the data flows remain on their traditional shortest-path routing approach.

D. Operating System for QoS Optimization

NOX, the standard network operating system, can be used for packet-level or flow-level control. However, it does not have the necessary APIs for QoS support. For instance, it does not support QoS-oriented virtual network management, or end-to-end QoS performance monitoring. In [98] an QoS-aware Network Operating System (QNOX) is proposed to support general OpenFlow QoS functions.

The QNOX system includes the following modules: WDM/ASON, IP, MPLS-TP. Here, WDM/ASON can monitor large network traffic status. QoS-aware Open Virtual Network Programming interface (QOVNPI) allows a client to request any type of QoS performance. The service element (SE) can be used for QoS demand definitions, such as the required network bandwidth, memory overhead, preferred server locations, packet loss rates, delay bounds, and security levels. The SLA (service level agreement) and SLS (service level specification) modules can be used to assess the OpenFlow resource availability, that is, to check whether the network can meet the client's QoS demands. Obviously QNOX can define fine-granularity of QoS, such as packet-level delay or loss rate. Based on the experimental results in [98], QNOX can quickly calculate the routing path in less than 100 ms even with over 100 nodes in the SDN. The SLA/SLS can find all network resources in less than 1 s.

TABLE III
A COMPARISON OF DIFFERENT SDN SECURITY SCHEMES

Sources	[104]	[105]	[106]	[107]	[108]	[109]	[100]
Uses OpenFlow/NOX	Yes	Yes	Yes	Yes	No	Yes	Yes
Introduce new architecture based on OpenFlow/NOX	Yes	No	Yes	No	No	Yes	Yes
Experiments (E) or real networks (R)	E	E	R	E	E	E	E
Software (S) or Hardware (H)	S	S	S	S	S	S	S
Introduces new language for SDN	No	No	No	No	Yes	No	No

E. Other QoS Supporting Strategies in SDN/OpenFlow

In [135] a SDN QoS scheme called PolicyCop is proposed to implement an open, vendor agnostic QoS policy management architecture. It has a special software interface for specifying QoS-based Service Level Agreements (SLAs). PolicyCop uses the control plane of SDNs to monitor the compliances of the QoS policies and can automatically adjusts the control plane rules as well as flow table in the data plane based on the dynamic network traffic statistics.

In [136] an OpenFlow QoS enhancement scheme is proposed to allow the creation or change of the behavior of the existing routing queues. It suggests that an OpenFlow capable switch report the queue status to the control plane. It has a module called Queue Manager plug-in which allows the uniform configuration of QoS capabilities in each OpenFlow switch. Such an idea is implemented in Ofelia testbed. Its implementation is based on OpenNMS, an open-source network management system.

In [137], an Iterative Parallel Grouping Algorithm (IPGA) is proposed to manage the prioritized flow scheduling issue, It has an inherent nature of parallelism for efficient execution in OpenFlow systems. Its algorithm is based on a M-ary multi-rooted tree, a Fat-tree used in most data center networks. It assumes that the SDN switches have two layers: lower pod switches (edge switches) and upper pod switches (aggregation switches). It formulates the flow scheduling issue as a linear binary optimization problem.

VI. SDN SECURITY

A. Intrusion Detection

SDN creates some new targets for potential security attacks, such as the SDN controller and the virtual infrastructure [103]. Besides all the traditional networks' attacking places (such as routers, servers, etc.), SDN has some new target points such as: (1) SDN controller: Here, traditional attacks listed above also exist; (2) Virtual infrastructure: it could have traditional attacks on the hypervisor, virtual switch and VM (virtual machine); (3) OpenFlow Network: attacks could occur in OpenFlow protocol for OpenFlow enabled devices.

In the following paragraphs, we will describe some typical OpenFlow/SDN safety (such as failure recovery) issues and security schemes (see Table III). Here, safety refers to the schemes that overcome natural faults, and security means to overcome intentional attacks.

A network intrusion detection and countermeasure selection (NICE) scheme is investigated in [106]. It aims to achieve the security in a virtual networks such as SDN and cloud computing. Cloud Security Alliance (CSA) survey shows cloud computing security is the top concern among different types of networks. The conventional patch-based security schemes do not work well in cloud data centers since the users could have full access to those centers. In [106] the attack graph based analytical models are used for intrusion detection. NICE includes two important phases:

1) It uses an intrusion detection agent called NICE-A to capture the traffic in each cloud server. A Scenario Attack Graph (SAG) can be established and updated each time the NICE-A scans the network. Based on the pattern analysis of the SAG, the NICE-A knows whether it should act.
2) Deep Packet Inspection (DPI) is activated if the virtual machine (VM) enters inspection state. It can use SAG to find security threats and VM vulnerabilities.

NICE runs low-overhead security software in each cloud server. It includes 3 software modules an attack analyzer, a network controller, and a VM profiling server. The VM profiling server can monitor the network state in real-time, and construct the operation profile for all services and ports. It also takes care of the connectivity issues between VMs. The attack analyzer can deduce the event correlations among different SAG nodes. It then finds potential security holes and detect an occurring threat. The network controller can control all configurations in each hardware device and software unit based on OpenFlow protocols. As we can see, NICE fits SDN very well.

B. Modular Security

Although OpenFlow (OF) decouples the data plane and control plane and thus greatly simplifies the hardware operations, it also brings single-point security issues: once the controller is attacked, all low-level switches are misled and cannot correctly deliver the packets.

FRESCO-DB [107], a database module, can simplify the SDN security key management. It defines unified session key format and IP reputation model. Inspired by Click router design, it uses a modular and composable security protocols. It consists of two important parts: (1) Application layer: it uses APIs and interpreter to support modular applications; (2) SEK (security enforcement kernel), can be used to perform all policy-related actions. Diverse security policies, such as DROP, REDIRECT, QUARANTINE, can be enforced by Security applications developed in FRESCO scripts, to react to network threats by simply setting an action variable. The above two parts are built into NOX. A network user can use FRESCO script language to define various security modules. Regarding the implementation of FRESCO, Python is used to implement the Application Layer prototype (total around 3000 lines of codes), and runs as an OpenFlow application on NOX.

C. SDN Traffic Anomaly Detection

In [108] it proposes 4 different OpenFlow traffic anomaly detection algorithms. Each of them is evaluated in real networks including both home and business networks. In the following we summarize the ideas of those 4 traffic anomaly detection algorithms:

1) *Threshold Random Walk with Credit Based Rate Limiting (TRW-CB) algorithm*: As we know, a TCP connection can be established in a much higher success rate if the server is not attacked. By using sequential hypothesis testing (i.e., likelihood ratio test), it analyzes each connection status and attempt to detect the worm infections.
2) *Rate-Limiting*: A virus infection can cause many connection request within very short time, while a benign traffic flow will never have such a high request rate. This is the principle of rate-limiting, that is, we check the request rate and detect a malicious event.
3) *Maximum Entropy Detector*: Maximum entropy calculations can be used to find traffic statistical features. By using a baseline distribution, maximum entropy model can be used to classify the packets into different categories, and each category could be detected as benign or abnormal.
4) *NETAD*: It acts like a firewall or filter. It simply scans the packet header and blocks any suspicious packet based on the packet attributions.

D. Language-Based Security

Analyzing how to program SDN in a secure and reliable manner is discussed in [109]. The solution involves development of a new programming model that supports the concept of a network slice. The isolation of the traffic of one program from another is achieved with help of slices. They also isolate one type of traffic from other. They have developed a semantics for slices, and illustrate new kinds of formal modular reasoning principles that network programmers can now exploit. It provides definitions of end-to-end security properties that slices entail and verify the correctness of a compiler for an idealized core calculus in a slice-based network programming. They have also described their implementation which is equipped with a translation validation framework that automatically verifies compiled programs using the Z3 theorem prover.

It is challenging today to implement isolation in networks. Most systems still use manual setup to block suspicious traffic. Such a setup is often labor-intensive and vendor-specific. In [109], it suggests that using a high-level programming language to set up the data delivery policies and isolate different domains. It leaves the error-prone low-level device configurations to the SDN compilers. Such a scheme overcomes the shortcoming of NOX, which cannot easily isolate different subnetworks when security holes are detected.

The language-based security [109] relieves the programmers from complicated security programming due to the use of slice isolation concept. A slice is defined as a virtual connection consisting of routers, switches, communication ports or links. The slices have been defined with both attributes and actions in [109]. A slice can be isolated from another if running them side by side in the same network does not result in slice leaking packets into the other slice. They defined several intuitive security properties like isolation and developed an operational condition called separation that implies the isolation property. Finally, they formalized a compilation algorithm and proved that it establishes separation and isolation.

E. Loop Detection Problem

The routing loops make packets never reach the final destination. In [110] it presents a dynamic algorithm which is built on header space analysis, and allows the detection of loops in SDNs. There the network model has been illustrated as a directed graph. Hence, concepts of header space analysis has been translated into the language of graph theory. Rule graphs and the dynamic loop detection problem are studied in [110]. They have shown how to model a network as a directed graph. By analyzing the reachability and connectivity of the topology graph, a node-to-node, no-loop path can always be found. A dynamic strongly connected component algorithm is proposed in [110] to allow us to keep track of edge insertions and deletions. It can also be used to detect loops in a routing path.

A comparison of all the above SDN security schemes is presented in the tabular form below:

F. SDN Safety Issue: Failure Recovery

In order to build a trustworthy SDN, we need to make a SDN resistant to both external failures (security issues) and internal failures (safety issues) [146]. Here, external failures refer to external, intentional attacks by adversaries. The above discussed security solutions aim to detect and overcome external attacks. The internal failures refer to natural faults due to some system-related shortcomings or unintentional human factors. We regard those internal failures as safety issues. For example, a SDN could fail if the communication link between the controller and the switches has outages due to bandwidth unavailability. Thus all controllers' commands cannot be delivered to the switches' flow tables. If the switch-to-switch path has link failure, many packets can get lost. Therefore, some type of link quality monitoring and path recovery schemes are needed to overcome the link failure.

There could be many of other safety issues in a SDN. For example, the controller may not be able to synchronously update all switches' flow tables due to schedule management failure. The switch may not be able to timely report traffic delivery status to the controller (thus the controller may not update the flow table for quite a while). When using multiple controllers in a SDN, the controllers may not be able to keep the consistent control due to communication delay. In the following discussion, we will illustrate some existing schemes that aim to address the SDN safety issues.

In [104] a fast failure recovery scheme is proposed for OpenFlow networks. It investigated the switch-over frequency and packet loss rate in its evaluation. It uses NOX software to recover services.

In OpenFlow network we can immediately or proactively add a flow entry to the table after a failure occurred. The total recovery time is determined by the lifetime of the flow entries. In [104] two values of timeouts are defined, one is called idle timeout, which means the time interval that a flow entry should be removed if not used for certain time (that is, no packet for that type of flow entry is passing through a switch); the other one is hard timeout, which is the maximum time interval that a flow entry can stay. No matter which timeout occurs, it will trigger the failure recovery.

Note that the system cannot be recovered if the controller has no idea on what type of failure occurred. The controller may just randomly add a flow entry in the table if the failure type is not recognized. In [104] NOX has been used to implement L2-learning scheme for failure detection. It is written in C++ (called L2-lerarning switch) or in Python (it is called L2-learning Pyswitch).

If a failure occurs, the incorrect flow entries should be erased from all switches, and new entries should be immediately added to each switch. The controller should have robust schemes to detect the failure, and find new routing path to deliver the flows. The controller will check the old routing path associated with the failed links. If the old path is still usable, it will not establish a new path. Otherwise, new path needs to be added to the flow entries and old entries should be removed immediately.

In [104] Ubuntu 9.04 is used to install Open vSwitch 1.1.0 and NOX 0.9.0. Over 10 K ping packets were sent out at the pace of one packet every 10 ms. The packet loss rate is calculated by counting the number of received ping packets. Hard timeout is set to 20 seconds, and idle timeout is 10 s. The routing loops are avoided by using spanning tree algorithms. The path reestablishment scheme in [104] is faster than conventional MAC re-convergence or ARP. It only uses 12 ms to recover from a link failure.

In [105] a scheme is called Operations, Administration, and maintenance (OAM) tool is used to re-establish a new path. To minimize the path switching time, it uses a proactive approach, that is, a backup path is pre-stored in the flow table in case a path fails. This scheme makes path recovery time less than 50 ms. In addition, some probing packets are periodically sent in the network. If it is not received by a node, the system knows that a path failure occurs. If it takes a long time to receive the probing packet, a failure is also detected. Thus [105] provides an efficient way to recover from path failure.

VII. OpenFlow for Wireless and Optical Networks

A. Overview

Why OpenFlow for wireless networks? Wireless infrastructure is more hybrid and complicated than wired ones. Many wireless standards, such as Wi-Fi, Wi-Max, cellular networks, etc., are all co-operating in the same backbone for providing anywhere Internet access. Managing such a heterogeneous wireless infrastructure is a big challenge. To make things worse, different wireless products have their own lower layer (physical/MAC layers) specifications, and are very difficult to re-configure for dynamic mobile applications. For example, Wi-Max forwards data in a point-to-point style in microwave frequency; while Wi-Fi uses one-to-many star topology in free frequency (2.45 GHz). OpenFlow can offload the wireless MAC layer operations to virtual machines, and uses software-defined network programming to achieve high flexibility and reconfigurability. OpenFlow decouples lower layer wireless transmission from higher layer control; thus it makes wireless data forwarding reach higher rate (Gbits/sec). This can fully explore 802.11 potential data rate.

The network virtualization in OpenFlow can significantly improve the scalability of the wireless virtual LAN tagging and firewall filtering operations. When the networks are moving to cloud computing, it becomes harder and harder to manage the dynamic and distributed cloud servers. OpenFlow can easily update the cloud policies over a dynamic deployment environment. However, it needs some innovative designs if applying SDN/OpenFlow to wireless world since the original SDN motivation was to use wired, high-speed switches to perform dump data forwarding, and to use reliable wires (not wireless) to achieve stable communications among SDN control plane units. If we shift everything to wireless media, how do we allocate different wireless channels for switch-to-switch or controller-to-controller communications? What if those radio channels are not available from time to time due to signal fading and shadowing? Some studies are solving those issues [19], [35], [111]–[114]. Later on we will use two examples (wireless sensor networks and wireless mesh networks) to explain how we can integrate OpenFlow with wireless technologies.

Why OpenFlow for optical networks? There is a great deal of benefits when adopting SDN/OpenFlow for optical network control: (1) Current optical networks have difficulties to react independently to requests from client systems distributed at the network edge. SDN provides programmable, abstracted interface for flexible application re-configurations in optical control units. (2) Existing optical networks cannot easily upgrade the software in each optical switch due to the embedded software nature. OpenFlow could easily upgrade services due to its separation of control and data planes. (3) SDN/OpenFlow allows multi-level abstraction via its networked re-programming and virtualization technologies. This makes optical network stack suit easily adapt to different network topologies. (4) The cost of optical hardware is typically high, especially the photonics and associated electronic components. SDN/OpenFlow could reduce those costs due to its 'dump' hardware operations—just simply following the flow table.

B. OpenFlow for Wireless Sensor Networks

Wireless sensor networks (WSN) have become important platforms for environmental monitoring. There are many sensor hardware designs such as CrossBow, Imotes, etc. However, all those sensor products cannot be easily programmed due to vendor-specific SDK (software development kit) and the tight integration of hardware and software in one sensor node.

Moreover, those sensors are difficult to re-task [115], [116] if a new environmental monitoring mission is required. For example, how can we re-program 100 sensors in a lake WSN to detect a new type of pollution? Obviously today we need

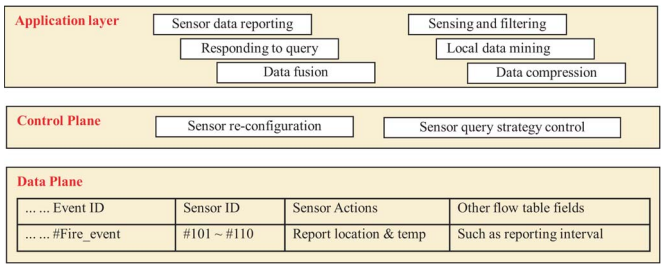


Fig. 20. SDN-based sensor networks.

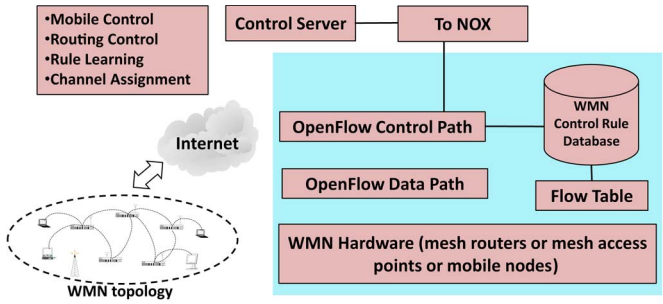


Fig. 21. OpenFlow for WMN management.

to take each sensor out of the water and change the programs embedded into the sensor hardware. This is not realistic in large-scale WSN with so many nodes.

Although some over-the-air programming techniques are used for some vendors' sensor boards, their data sensing and forwarding schemes are still vendor-specific. For example, they may use different operating systems, or different programming languages. The programmer needs to check different manuals to get familiar with the API functions. It will be better if the user just simply configure a network controller based on universal networked operating system. The sensor hardware and embedded stack protocols could be decoupled such that the users do not need to worry about the data forwarding details in each sensor, and just simply configure the controller's flow table. The data forwarding rules do not necessarily follow the specific MAC layer protocols (such as ZigBee, 802.11, or other protocols).

SDN/OpenFlow can well solve the above issues. It makes each sensor just simply forward the sensor data based on the specified flow table and rules. All those rules can be easily changed through controllers' programming. Since all nodes follow universal operating system (such as NOX), the re-tasking can be easily achieved by following standard scripts programming. A Software-Defined WSN architecture, called sensor openflow [116], can be used to address key technical challenges mentioned above. We illustrate its main ideas in Fig. 20.

It has three layers: the application layer has all sensor data query related applications such as local data processing; the control plane and data plane are totally separate: the former can remotely re-configure the sensor parameters, and the latter can check the flow table and perform the corresponding actions. Its main idea is to make the large-scale sensor network easy-to-manage via programmable control plane and user-customizable flow table. Sensors are no longer application-dependent and the sensor data query policies can be easily reset. Sensor OpenFlow allows policy changes in an easy style since a programmer can simply change the controller's software instead of dealing with the wireless sensors.

C. OpenFlow for Wireless Mesh Networks

OpenFlow could be very useful for wireless mesh network (WMN) management. Today WMN is often used in community networks or military applications for re-tasking from time to time. For example, an Internet provider may re-program a community mesh network to set up different IPTV services. A military center may want to re-configure a wireless network to adapt to different surveillance scenarios. Existing mesh network nodes are full fledged with all physical to application layer functionalities. The network manager needs to setup each mesh node individually since each node may have vendor-specific programming features or proprietary device management profiles. Overall, today it is very difficult to perform rapid re-tasking or policy changes in the heterogeneous mesh clients (such as laptops, PDAs, phones, etc.) in a mesh network. OpenFlow decouples network control and hardware communications completely, and leave only basic data forwarding functions in each node, while the entire network can be easily re-programmed through a standard network OS (such as NOX) running in a control panel. As long as different vendors' products support OpenFlow's flow table managements, a mesh network can be easily re-tasked through a standard network control script programming.

To make OpenFlow applicable to WMN, we need to overcome a few challenges [20], [117]:

- Challenge 1: Fading channel: Unlike Stanford OpenFlow testbed where fixed wired network is the backbone, WMN has wireless channels everywhere (issues: radio fading, hidden terminal problem, wireless broadcast nature, etc.).
- Challenge 2: Dynamic Topology: Due to WMN link variations and nodes membership dynamics, the network topology changes at a much higher pace than in wired network. The OpenFlow needs to build a control plane to perform autonomous topology discovery and swiftly react on changes of the WMN topology.
- Challenge 3: In-band or out-band control: OpenFlow often adopts out-of-band signaling, that is, the channel to NOX is separate from the actually data forwarding network. However, in WMN we may not have different RF channels for separate control. On the other hand, using in-band control would decrease data network throughout.

Fig. 21 shows the basic principle of using OpenFlow for WMN control. The WMN has both mesh routers and mesh clients. A radio channel control strategy is achieved by the control panel for router-to-router, router-to-client, and client-to-client communications. The control server in control plane can perform mobility management, routing strategy, and channel assignment.

In [20] an OpenFlow-enabled mesh routing scheme is proposed. It has OpenFlow-enabled routers, clients and gateways. Each node has multiple radio cards for multi-radio communications. The data path uses local sockets to talk with the

control plane units. The control path communicates with NOX via secure channel. Connection to Internet is achieved through mesh gateways.

In [20] the in-band wireless communications are used between the controller and the switches. The high-quality channels are used for controller-to-server communications since the controller's commands cannot be lost even there is signal fading.

D. OpenFlow for Optical Networks

Today optical networks have become the fastest Internet data transmission approaches due to the high-speed light propagation in optical fibers. Typically an optical network consist of nodes such as Wavelength Cross-Connects (WXC), Reconfigurable Optical Add-Drop Multiplexers (ROADM), and Photonic Cross-Connects (PXC) [118]. Current optical nodes can be controlled by Element Management System (EMS) and the Network Management System (NMS), which uses either manual or semi-static style for lightpath provisioning [119]. Although this approach is reliable, it is difficult to design a control plane technique to achieve a control of dynamic wavelength paths in metro/backbone optical networks. Such a control plane should be able to reduce operational expense, shorten the data transmission latency, and should be highly scalable to the network traffic. An important optical control scheme is called Generalized Multi-Protocol Label Switching (GMPLS) [120]. It is a distributed packet forwarding control scheme. However, It has not been popularly used in optical network products [121]. One important reason is its complex control scheme that is not suitable to dynamic control of both IP and optical layers via a unified control plane (UCP).

SDN architecture, in particular, the OpenFlow protocol, could become a solution to the above issue. Although the initial purpose of using OpenFlow is to create a re-programmable network, it can also serve as a promising candidate for a UCP solution in hybrid networks [122]. It has been studied in optical network enhancements [123]–[125]. But it is still in the early stage for real networking.

In [119] an Openflow based PXC architecture is proposed. It uses a concept called virtual Ethernet interfaces (veths) to connect to each OpenFlow switch. Those veths look "virtual" from the viewpoint of the PXC physical interfaces. Thus the control plane can easily manage all PXC interfaces. The virtual OpenFlow switch is also called OpenFlow agent. The integrated OpenFlow agent and the PXC is called OpenFlow-enabled PXC (OF-PXC). It can be managed by a NOX controller. When the packets are received by the NOX, it can either insert a new record to the flow table (if this is the first packet) or decides which veth to forward the data.

VIII. Example of Complete SDN System

To illustrate a complete SDN system, here we use a good reference solution called MobileFlow [139] which uses a SDN architecture to implement a mobile network. A software-defined mobile network (SDMN) provides maximum flexibility, openness, and programmability to future carrier. It designs

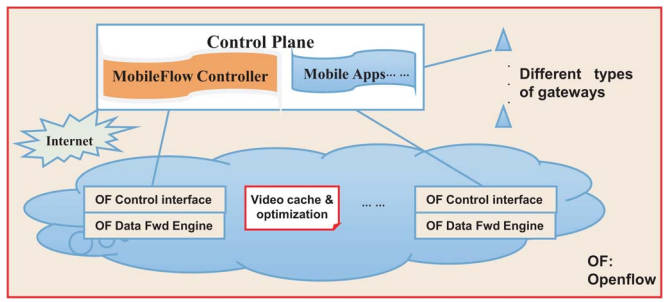


Fig. 22. Software defined mobile network.

a special SDN data plane called MobileFlow forwarding engine (MFFE). All MFFEs are interconnected by an underlying IP/Ethernet transport network. Its SDN control plane has MobileFlow controller (MFC). The MFC has mobility management entity (MME).

As shown in Fig. 22, it has MobileFlow and OpenFlow levels for the management convenience. In both of them the control plane is decoupled from the data plane. The data forwarding function in MFFE is fully defined in software, while the control software can steer the user flows to different service enables (such as video caching and optimization). Those services can be distributed throughout the mobile network. Note that the user's traffic can go through both MobileFlow level and OpenFlow level, or, it just goes through the OpenFlow level and reaches the Internet. Fig. 24 also shows the OpenFlow-based decoupling of the IP/Ethernet transport network in the lower level. This is because in some cases the user traffic may not go through the mobile network infrastructure (and just stays in the wired network, no wireless).

MobileFlow Controller: Such a controller can perform network-level management including topology auto-discovery, device monitoring, topological resource view, topology virtualization, etc. More importantly, it can handle all mobility-related activities, such as mobility anchoring, service migration, channel handoff, etc. A network operator can freely use MFFEs from different vendors. The operator can also use MFFEs to adopt novel mobile network architectures. The system supports m:1 mapping between MobileFlow applications and NFFEs since each mobile application belongs to a different control plane, and therefore enabling multitenancy.

Mobile Management: Supporting mobility based on 3GPP or other systems can be easily realized by introducing the mobile applications above the northbound interface of the MFC. The MFC cab send out flow control rules to each MFFEs involved in handling the particular flow. The channel handoff (i.e., communication frequency switching) can be easily implemented in each MFC when the QoS requirements are not met in the multimedia applications.

Testbed Implementation: It uses COTS x86 based general-purpose servers and OpenFlow-supported routers/switches from different vendors. It enables the software-based definition of different mobile networks (3G, 4G, etc.) with different characteristics (radio coverage, bandwidth, radio frequency, etc.). It supports virtualization for both control- and data-plane resources. The same MFFE can be reused by different types of virtual networks. Each virtual mobile network can evolve

and be upgraded solely by replacing the corresponding virtual machine software units in the control plane servers. The MFFEs remain intact. The entire testbed is interconnected via COTS LAN switches.

Note that the above system adds MobileFlow level above regular OpenFlow level. Thus it can directly use OpenFlow language syntax for network definition. For example, it can use the following language abstraction to monitor a port 80 traffic:

$$Def switch\_join(switch)) :$$
$$P = inport : 2, tp\_src : 80$$
$$Install(switch, p, DEFAULT, [\,])$$
$$Query\_stats(switch, p)$$

Such a high-level language can run in NOX and interpreted by all OpenFlow-compatible routers. Overall, the MobileFlow system is a complete SDN/OpenFlow implementation in mobile applications. It allows flexible re-configuration of mobile channel allocation, QoS parameters, mobile access, and service roaming between wireless networks.

IX. Research Trends

There are still many questions on how to make the SDN more efficient, how to optimize it across all the network sets, and how to achieve tradeoffs between different implementations. There is a need to have quantitative metrics/approach for evaluating the performance and efficiency of the SDN such as its scalability, availability and latency. In the following we point out some important future research topics.

A. Intelligent Flow Table/Rules Management

It is true that the motivation of designing SDN/OpenFlow is to simplify the network switches' operations: instead of using vendor-dependent, embedded software in each switch, SDN only assumes 'dump' data forwarding functionalities in each switch. The switch just simply checks the flow table to determine how to forward each received data packet.

The tricky part is: although the switches do not analyze the traffic, they can always report forwarding results to the higher layer—OpenFlow controller. The results could be simple success or failure for a data forwarding operation, or some error messages (such as switch failure), or other forwarding status data. In the current OpenFlow specifications, they do not specify how to handle those feedback data. They just point out that the flow table should be set up based on a set of rules defined by the controller. But the issue is: since the switches could have high traffic burden, it will slow down their data forwarding operations if (1) network traffic is heavy, and/or (2) the flow table is large and has complex rules. It is important to perform self-learning in the controllers based on the pattern analysis of the traffic flowing through each switch.

We believe that the future trend of SDN/OpenFlow will include high intelligence in flow table/rules control. We illustrate our idea in Fig. 23. The network controller can learn what's going on based on the feedback from the switches. For example, if the switches report a long delay for a source-destination IP

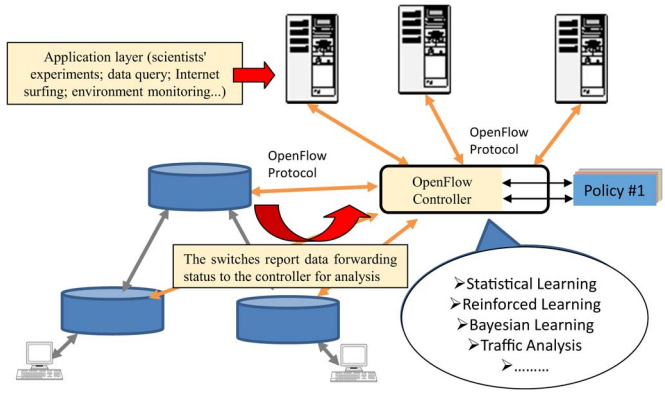


Fig. 23. Intelligent flow table/rules management.

pair, it indicates a possible routing loop or switch congestion somewhere. When the controller analyzes the statistical patterns from the switches' feedback data, it can use any of the recent learning tools (such as Bayesian learning, reinforced learning, etc.) to deduce the optimal 'actions' in the future in order to obtain a higher accumulative reward. The reward could be defined based on the QoS performance metrics. The 'actions' could be any packet handling operations or any policy changes.

Statistical analysis could be based on any traffic pattern data mining schemes. Through dimension reduction, we could extract the intrinsic features from complex, multi-attribute traffic data. Since some dimensions are not useful in pattern recognition, they could be removed by using Principle Component Analysis (PCA), Non-negative Matrix Factorization (NMF), or other dimension reduction schemes.

B. Scalable Controller Management

With the application of SDN/OpenFlow in larger networks, the network controllers could become a performance bottleneck due to large amount of incoming signaling messages and forwarding requests. Those controllers do not necessarily be deployed in the same sub-net since a company network may cross multiple places (even in different countries, such as IBM, Intel, etc.). The controllers located in a distributed network may compete for common computing resources (such as communication channels).

To manage the coordination issue of large-scale controller-to-controller communication system, a carefully designed scheduling strategy with collision avoidance should be used. On one hand, SDN administrators want to see a virtually consistent controller system. On the other hand, they need to design a virtual-to-physical mapping model to manage the physically distributed controllers. Perhaps a tree-based hierarchical management scheme could be used to coordinate those controllers. The higher level controllers should be able to handle more heavy requests. The root controller then communicates with NOX on the global requests. Fault tolerance techniques could be used for controllers system. Even one controller is down, others should be able to compensate for the missing operations. By using fault tolerance model, we could figure out the optimal controller deployment strategy, such as which controller should be deployed in which sub-net.

Resource sharing strategies should also be made among controllers. By using a queuing model with shared resource pool, we could calculate the controller serving time and waiting time.

Note that the flow table could handle the packets in different granularity levels. It could define the packet forwarding in individual packet level; or, it could define the flow-level forwarding actions. The controllers must be able to manage different granularity levels in order to accurately adjust the flow table status (such as adding or deleting the forwarding rules).

C. Highly Flexible Language Abstractions

The SDN programming language should be able to adapt to frequent flow rules changes. It should also be suitable to policy changes due to network topology change, regular SDN maintenance, emergent failures, etc. An SDN programmer would like to use a good language to perform policy changes atomically to each switch. However, atomic change is difficult to implement since it needs the disruption of the entire network during policy change [29]. The future SDN language should at least achieve two levels of abstractions if the atomic level cannot be achieved.

1) In packet level, the language should be able to specify the policy changes for all packets that meet similar attributes. It needs to make sure that all packets belonging to the same context are delivered with the same structural invariants such as loop-freedom.
2) In flow-level, the language should allow the definitions of flow-oriented rules/policies, such as queuing models, delivery order, load balancing, etc. The compliers and runtime system should be able to respond to the aggregated flow-level rule changes.

Another language abstraction trend is to support modular programming, that is, it should allow the handling of isolation issues between multiple programs that control different portions of the traffic. Each piece of program has different tasks, some target host monitoring, some for failure recovery, some for virtualization, and so on. How do we make all pieces of program interface to each other in a transparent way? This needs a high level of abstraction for SDN programming.

D. Low-Cost Fine-Granularity QoS Implementation

As we know, QoS strategies include class-based differentiated service (DS) and fine-grained integrated service (IS). SDN/OpenFlow could define packet-level or flow-level priorities and performance metric. Therefore, it can be used to support IS. Although some IS-oriented OpenFlow QoS models are defined [96], [97], not many practical implementations are conducted. The main challenges include.

1) The integration of multimedia coding with OpenFlow QoS management: There are many video encoding standards. Especially some standards such as H.264+/SVC can support priority-based coding, that is, different video data layers could be assigned different priorities, and the enhancement layer has the most important video data. OpenFlow could support such video streaming by controlling different flows in different policies. However, the detailed flow rules need to be integrated with different video encoding standards, which needs further research;
2) Load balancing: Content Distribution Networks (CDNs) need load balancing capability in order to distribute the heavy workload across the network elements. While most of conventional load balancing strategies for multimedia streaming (live or on-demand) over CDNs rely on server-based load balancing, OpenFlow allows the load balancing actions in each possible flow path (Fig. 24). The future research needs to define the detailed procedure in OpenFlow controller in order to achieve such an integrated server balancer selection and path selection.
3) Use Cross-layer design style to optimize QoS: Many QoS optimization schemes are based on cross-layer designs [126]–[128]. However, OpenFlow removes the boundaries of traditional Internet, and uses open, programmable model. Then the issue is: how do we implement cross-layer design style in OpenFlow in order to share all available network parameters in different places for QoS optimization?

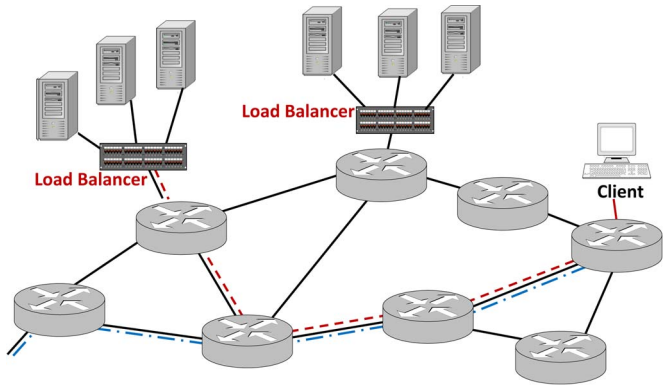


Fig. 24. Load Balancing: Integrate server balancer selection with path selection [96].

E. Resilient Security in SDN

SDN/OpenFlow uses network virtualization technology to simplify the resource management of the large network. It enables the definition of virtual slices/slivers for different physical utilities (such as hard disk, memory, etc.). A slice could include several slivers. Each slice/sliver pair could be assigned a unique ID. Due to resource limitations, some malicious OpenFlow terminals may use attacks to try to overuse the resource slice/slivers. Therefore, we need to create a scale security scheme that can overcome the resource access attacks in slice/sliver establishment.

Some SDN security challenges include: (1) Scalability issues: If many slivers are needed in a slice, it has high overhead to generate/distribute different session keys for different slivers. (2) Sliver deactivation: When a sliver deactivates a sliver, we need to make sure none of the stored data can be decoded independently by that user (this is called forward secrecy).

Here, we suggest a possible security solution based on ID-based cryptography [130]. While conventional public key schemes use random string to generate public key, ID-based

crypto generates public keys from user IDs. Thus, it makes key management in SDN much easier since we do not need to distribute public keys to SDN users. Moreover, the encryption/decryption can be done offline (thus a key generation center is not needed). To implement the above ID-based security, some issues need to be addressed such as mutual authentication between experimenters and slices, key escrow issue, etc.

F. Robust Wireless Integration

Although OpenFlow has been well developed in wired, local network, there are very few studies on its performance in wireless networks. All the existing wireless network OpenFlow designs [19], [35], [111]–[114] have not overcome the following two challenging issues.

1) Integrated Management of Channel Access and Data Forwarding: Many wireless networks support multi-channel communications among routers and clients, especially in cognitive radios. Therefore, each wireless node needs to detect available channels, and select the high-quality ones. Moreover, the nodes can change physical characteristics for optimal link radio communications. Thus in a wireless networks the OpenFlow data panel must perform efficient channel sensing/access. However, the existing OpenFlow standards only define data forwarding functions in the hardware. We will need to significantly improve the existing OpenFlow model (including its control/data panels' task division, FlowVisor control, network visualization, etc.) by designing a brand-new flow-table management scheme. Such a scheme may use reinforcement learning to simultaneously manage two flow tables one for real-time data forwarding and another for multi-channel sensing/selection.

2) Conflict-Free Scheduling of Control Traffic and Data Traffic: Unlike wired OpenFlow model that can use cables to easily achieve control/data packet communications among nodes, wireless network uses unreliable wireless links for both control panel communications and data panel packet forwarding. The control panel demands a high-quality channel for loss-free delivery, while the data panel should use other available channels for routing. Therefore, a carefully designed channel allocation and packet scheduling scheme is required for conflict-free control/data traffic delivery in routers and nodes.

X. Conclusion

In this paper, we have comprehensively surveyed the design issues for SDN/OpenFlow. Especially we have covered all important issues in concrete network implementation including language abstraction, controller design, virtualization scheme, QoS support, security issues, and wireless/optical network integration.

Below we briefly summarize some important aspects for highlight: SDNs have many applications including developing new protocols prior to implementing them in real networks, increasing connectivity in rural environments, making both cloud based and regular data centers better, and supporting mobile device offloading. As the Internet continues to grow and becomes available to increasing more people, networks will need to be able to adapt to ever changing circumstances. SDNs allow the data and control planes to be separated, and hence to be easier for improvements.

Network virtualization based on OpenFlow is a successful implementation of Software Defined Networking, and its development offers users a great deal of convenient services. We have reviewed different virtualization architectures that focus on the improvement of the network flexibility, isolation and management. It can be seen that embedding additional module or abstraction layer on the top of OpenFlow or FlowVisor provides solutions to these challenges. Utilizing the database can also help to simplify the creation of the abstraction layer.

QoS is an import issue in many applications especially in streaming media, VoIP, Videoconferencing, and so on. Many experiments have been conducted to make OpenFlow support QoS control. However, these designs are still under testing phase, and need to be further examined. Many designs are related to optimization problems, such as dynamic rerouting for SVC, dynamic QoS re-negotiation for multimedia flows, etc. And the solution needs heavy calculations in reality as the dimension increases. It also needs a comprehensive test before being applied to real world applications. Many experiments are actually under a small scale of tests to verify the proposed design, and no large-scale experiments have been performed yet.

Security and privacy are always important in any network. OpenFlow brings many new security challenges due to the virtual network management. It is important to design new low-overhead security/privacy schemes to protect the virtual-to-physical mapping protocols in SDN/OpenFlow.

We have also pointed out some important unsolved research issues in this exciting field. Those issues may serve as the thesis/dissertation topics for graduate students. SDN/OpenFlow is a relatively new field, and many practical design issues are waiting for in-depth investigations. We believe that this comprehensive survey could help R&D people to understand the state-of-the-art in SDN/OpenFlow.

Acknowledgment

We sincerely thank the following people for their help with this survey: A. Gerrity, M. Farooq, T. Zhang, R. Ma, C. Dickerson, X. Fu, N. Hegde, (they are all with ECE department at the University of Alabama), and A. V. Vasilakos (University of Western Macedonia in Greece). They have provided valuable comments and inputs on some of the above discussed topics. We also appreciate the editor and reviewers' time and effort for reviewing this paper.

References

[1] S. Ortiz, "Software-defined networking: On the verge of a breakthrough?" *Computer*, vol. 46, no. 7, pp. 10–12, Jul. 2013.

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

[3] K. Bakshi, "Considerations for software defined networking (SDN): Approaches and use cases," in *Proc. IEEE Aerosp. Conf.*, Mar. 2013, pp. 1–9.

[4] S. Agarwal, M. Kodialam, and T. V. Lakshman, "Traffic engineering in software defined networks," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 2211–2219.

[5] S. Fang, Y. Yu, C. H. Foh, and K. M. M. Aung, "A loss-free multipathing solution for data center network using software-defined networking approach," *IEEE Trans. Magn.*, vol. 49, no. 6, pp. 2723–2730, Jun. 2013.

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

[7] S. Das *et al.*, "Packet and circuit network convergence with OpenFlow," in *Proc. Opt. Fiber Commun. Conf. Expo.*, Mar. 2010, pp. 1–3.

[8] R. Sherwood, M. Chan, and A. Covington, "Carving research slices out of your production networks with OpenFlow," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 129–130, Jan. 2010.

[9] OpenFlow. [Online]. Available: http://www.openflow.org/

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

[11] C. S. Li and W. Liao, "Software defined networks," *IEEE Comm. Mag.*, vol. 51, no. 2, p. 113, Feb. 2013.

[12] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian, "Fabric: A retrospective on evolving SDN," in *Proc. Workshop Hot Topics Softw. Defined Netw.*, Aug. 2012, pp. 85–90.

[13] Y. Kanaumi, S. Saito, and E. Kawai, "Toward large-scale programmable networks: Lessons learned through the operation and management of a wide-area OpenFlow-based network," in *Proc. Int. Conf. Netw. Serv. Manage.*, Oct. 2010, pp. 330–333.

[14] H. Fei, *Network Innovation Through OpenFlow and SDN: Principles and Design*. New York, NY, USA: Taylor & Francis, 2014.

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

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

[17] T. D. Nadeau and P. Pan, "Software driven networks problem statement," *IETF Internet-Draft (Work-in-Progress)*, Oct. 2011, draft-nadeau-sdn-problem-statement-01.

[18] M. Yu, J. Rexford, M. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," *Proc. ACM SIGCOMM Comput. Commun. Review*, vol. 40, no. 4, pp. 351–362, Oct. 2010.

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

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

[21] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng, "Reliability-aware controller placement for software-defined networks," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage.*, May 2013, pp. 672–675.

[22] P. T. Congdon, P. Mohapatra, M. Farrens, and V. Akella, "Simultaneously reducing latency and power consumption in OpenFlow switches," *IEEE/ACM Trans. Netw.*, vol. 22, no. 3, pp. 1007–1020, Jun. 2014.

[23] A. Khan and N. Dave, "Enabling hardware exploration in software-defined networking: A flexible, portable OpenFlow switch," in *Proc. Annu. Int. Symp. Field-Programmable Custom Comput. Machines*, Apr. 2013, pp. 145–148.

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

[25] Y. Kanaumi *et al.*, "Deployment and operation of wide-area hybrid OpenFlow networks," in *Proc. IEEE NOMS*, Apr. 16–20, 2012, pp. 1135–1142.

[26] OpenFlow Switch Specification, Version 1.0.0. [Online]. Available: http://www.openflow.org/documents/openflow-spec-v1.0.0.pdf

[27] OpenFlow in Europe—Linking Infrastructure and Applications. [Online]. Available: http://www.fp7-ofelia.eu/

[28] Y. Kanaumi, "Large-scale OpenFlow testbed in Japan," in *The 31st APAN Meet.*, Feb. 2011, pp. 1–22.

[29] N. Foster, M. J. Freedman, A. Guha, and R. Horrison, "Languages for software-defined networks," *IEEE Commun. Mag. Feature Topic Softw. Defined Netw.*, vol. 51, no. 2, pp. 128–134, Feb. 2013.

[30] F. de O. Silva, J. H. de S. Pereira, P. F. Rosa, and S. T. Kofuji, "Enabling future Internet architecture research and experimentation by using software defined networking," in *Proc. Eur. Workshop Softw. Defined Netw.*, Oct. 2012, pp. 73–78.

[31] S. Hasan, Y. Ben-David, C. Scott, E. Brewer, and S. Shenker, "Enhancing rural connectivity with software defined networks," in *Proc. ACM Symp. Comp. Develop.*, 2013, no. 49, pp. 1–2.

[32] R. Narayanan *et al.*, "Macroflows and microflows: Enabling rapid network innovation through a split SDN data plane," in *Proc. Eur. Workshop Softw. Defined Netw.*, Oct. 2012, pp. 79–84.

[33] B. Boughzala, R. Ben Ali, M. Lemay, Y. Lemieux, and O. Cherkaoui, "OpenFlow supporting inter-domain virtual machine migration," in *Proc. Int. Conf. Wireless Opt. Commun. Netw. (WOCN)*, May 2011, pp. 1–7.

[34] C. Baker, A. Anjum, R. Hill, N. Bessis, and S. L. Kiani, "Improving cloud datacenter scalability, agility and performance using OpenFlow," in *Proc. Int. Conf. Int. Netw. Collaborative Syst.*, Sep. 2012, pp. 20–27.

[35] A. Gember, C. Dragga, and A. Akella, "ECOS: Leveraging software-defined networks to support mobile application offloading," in *Proc. Eighth ACM/IEEE Symp. Architectures Netw. Commun. Syst.*, 2012, pp. 199–210.

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

[37] P. A. A. Gutiérrez and D. R. Lopez, "An OpenFlow network design cycle," in *Netw. Innovation through OpenFlow SDN: Principles Design*. New York, NY, USA: Taylor & Francis LLC, CRC Press, 2014.

[38] R. Stallman, R. Pesch, and S. Shebs, *Debugging With GDB: The Source Level Debugger*. Boston, USA: GNU Press, 2002.

[39] The Eclipse Foundation. [Online]. Available: http://www.eclipse.org

[40] G. T. Heineman and W. T. Councill, Eds., *Component-Based Software Engineering: Putting the Pieces Together*. Reading, MA, USA: Addison-Wesley, 2001.

[41] L. M. Correia, Ed., *Architecture and Design for the Future Internet*. New York, NY, USA: Springer-Verlag, 2011.

[42] A. de C. Alves, *OSGi Application Frameworks*. Shelter Island, NY, USA: Manning Publications, 2009.

[43] J. Highsmith and A. Cockburn, "Agile software development: The Business of Innovation," *Computer*, vol. 34, no. 9, pp. 120–122, Sep. 2001.

[44] K. Petersen, C. Wohlin, and D. Baca, "The waterfall model in large-scale development," in *Proc. Int. Conf. Prod.-Focused Softw. Process Improvement*, 2009, pp. 386–400.

[45] OpenVSwitch. [Online]. Available: http://openvswitch.org/development/openflow-1-x-plan

[46] Pica8 Open Network Fabric. [Online]. Available: http://www.pica8.org/solutions/openflow.php

[47] Indigo—Open Source OpenFlow Switches. [Online]. Available: http://www.openflowhub.org/display/Indigo/

[48] NOX. [Online]. Available: http://www.noxrepo.org/nox/about-nox/

[49] POX. [Online]. Available: http://www.noxrepo.org/pox/about-pox/

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

[51] Floodlight: A Java-Based OpenFlow Controller. [Online]. Available: http://floodlight.openflowhub.org/

[52] D. Erickson. [Online]. Available: https://openflow.stanford.edu/display/Beacon/Home

[53] Floodlight is an Open SDN Controller. [Online]. Available: http://floodlight.openflowhub.org/

[54] OpenStack: Open Source Software for Building Private and Public Clouds, 2012. [Online]. Available: http://www.openstack.org/

[55] A. Orebaugh, G. Ramirez, J. Burke, and L. Pesce, *Wireshark & Ethereal Network Protocol Analyzer Toolkit (Jay Beale's Open Source Security)*. Sebastopol, CA, USA: Syngress Publishing, 2006.

[56] ns3. [Online]. Available: http://www.nsnam.org

[57] J. Pelkey, OpenFlow Software Implementation Distribution. [Online]. Available: http://code.nsnam.org/jpelkey3/openflow

[58] T. L. Hinrichs, N. S. Gude, M. Casado, J. C. Mitchell, and S. Shenker, "Practical declarative network management," in *Proc. ACM Workshop Res. Enterprise Netw.*, 2009, pp. 1–10.

[59] N. Foster *et al.*, "Frenetic: A network programming language," in *Proc. ACM SIGPLAN Int. Conf. Functional Program.*, 2011, pp. 279–291.

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

[61] Y. Chiba and Y. Nakazawa, Tremashark: A Bridge for Printing Various Events on Wireshark. [Online]. Available: http://github.com/trema/trema.gitmaster/tremashark

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

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

[64] M. Canini, D. Venzano, P. Pereni, D. Kostic, and J. Rexford, "A NICE way to test OpenFlow applications," in *Proc. USENIX Conf. Netw. Syst. Design Implementation*, 2012, pp. 10–10.

[65] M. Canini, D. Venzano, P. Pereni, D. Kostic, and J. Rexford, A NICE Way to Test OpenFlow Controller Applications. [Online]. Available: http://code.google.com/p/nice-of/

[66] GENI: Exploring Networks of the Future. [Online]. Available: http://www.geni.net

[67] Ofelia: OpenFlow Test Facility in Europe. [Online]. Available: http://www.fp7-ofelia.eu

[68] JGN-x Utilization Procedure. [Online]. Available: http://www.jgn.nict.go.jp/english/info/technologies/openflow.html

[69] M. Christopher and A. Story, "Language abstractions for software-defined networks," *IEEE Comm. Mag.*, vol. 51, no. 2, pp. 128–134, Feb. 2013.

[70] K. Daisuke, K. Suzuki, and H. Shimonishi, "A design and implementation of OpenFlow controller handling IP multicast with fast tree switching," in *Proc. IEEE/IPSJ Int. Symp. Appl. Internet*, Jul. 2012, pp. 60–67.

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

[72] H. Kudou, M. Shimamura, T. Ikenaga, and M. Tsuru, "Effects of routing granularity on communication performance in OpenFlow networks," in *Proc. IEEE Pacific Rim Conf. Commun., Comput. Signal Process. (PacRim)*, Aug. 2011, pp. 590–595.

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

[74] G. Schaffrath *et al.*, "Network virtualization architecture: Proposal and initial prototype," in *Proc. ACM SIGCOMM VISA*, 2009, pp. 63–72.

[75] N. Sarrar *et al.*, "Leveraging Zipf's law for traffic offloading," in *Proc. ACM SIGCOMM Comput. Commun. Review*, 2012, pp. 16–22.

[76] Y. Yiakoumis, K.-K. Yap, S. Katti, G. Parulkar, and N. Mckeown, "Slicing home networks," in *Proc. ACM SIGCOMM Workshop Home Netw.*, 2011, pp. 1–6.

[77] S. Sundaresan *et al.*, "Broadband Internet performance: A view from the gateway," in *Proc. ACM SIGCOMM*, Aug. 2011, pp. 134–145.

[78] M. Al-fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM Conf. Data Commun.*, 2008, pp. 63–74.

[79] M. Casdo *et al.*, "Ethane: Taking control of the enterprise," in *Proc. ACM SIGCOMM Comput. Commun. Review*, 2007, pp. 1–12.

[80] Z. Cai, A. L. Cox, and T. S. E. NG, "Maestro: A System for Scalable OpenFlow Control," Rice University-Department of Computer Science, Houston, TX, USA, Tech. Rep. TR10-11, Dec. 2010.

[81] A. R. Curtis *et al.*, "DevoFlow: Scaling flow management for high-performance networks," in *Proc. ACM SIGCOMM Conf.*, 2011, pp. 254–265.

[82] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: Measurements and analysis," in *Proc. ACM Internet Meas. Conf.*, 2009, pp. 202–208.

[83] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. ACM Int. Meas. Conf.*, 2010, pp. 267–280.

[84] A. Tootoonchian, S. Gorbunov, Y. Ganjali, and M. Casado, "On controller performance in software-defined networks," in *Proc. USENIX Workshop Hot Topics Manag. Internet, Cloud, Enterprise Netw. Serv.*, 2012, p. 10.

[85] D. Drutskoy, "Software-Defined Network Virtualization," M.S. thesis, Princeton University, Princeton, NJ, USA, 2012.

[86] J. Matias, B. Tornero, A. Mendiola, E. Jacob, and N. Toledo, "Implementing layer 2 network virtualization using OpenFlow: Challenges and solutions," in *Proc. Eur. Workshop Softw. Defined Netw.*, Oct. 2012, pp. 30–35.

[87] P. Skoldstrom and K. Yedavalli, "Network virtualization and resource allocation in OpenFlow-based wide area networks," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2012, pp. 6622–6626.

[88] B. Sonkoly *et al.*, "OpenFlow virtualization framework with advanced capabilities," in *Proc. Eur. Workshop Softw. Defined Netw.*, Oct. 2012, pp. 18–23.

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

[90] M. El-azzab, I. L. Bedhiaf, Y. Lemieux, and O. Cherkaoui, "Slices isolator for a virtualized OpenFlow node," in *Proc. Int. Symp. Netw. Cloud Comput. Appl.*, Nov. 21–23, 2011, pp. 121–126.

[91] D. Turull, M. Hidell, and P. Sjodin, "libNetVirt: The network virtualization library," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2012, pp. 5543–5547.

[92] R. Nejbati, S. Azodolmolky, and D. Simeonidou, "Role of network virtualization in future Internet innovation," in *Proc. Eur. Conf. Netw. Opt. Commun.*, Jun. 2012, pp. 1–4.

[93] B. Naudts *et al.*, "Techno-economic analysis of software defined networking as architecture for the virtualization of a mobile network," in *Proc. Eur. Workshop Softw. Defined Netw.*, Oct. 2012, pp. 67–72.

[94] A. Mahmud, R. Rahmani, and T. Kanter, "Deployment of flow-sensors in Internet of things' virtualization via OpenFlow," in *Proc. FTRA Int. Conf. Mobile, Ubiquitous, Intell. Comput.*, Jun. 2012, pp. 195–200.

[95] H. E. Egilmez, B. Gorkemli, A. M. Tekalp, and S. Civanlar, "Scalable video streaming over OpenFlow networks: An optimization framework for QoS routing," in *Proc. IEEE Int. Conf. Image Process.*, Sep. 2011, pp. 2241–2244.

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

[97] H. E. Egilmez, S. Civanlar, and A. M. Tekalp, "An optimization framework for QoS-enabled adaptive video streaming over OpenFlow networks," *IEEE Trans. Multimedia*, vol. 15, no. 3, pp. 710–715, Apr. 2013.

[98] K. Jeong, J. Kim, and Y.-T. Kim, "QoS-aware network operating system for software defined networking with generalized OpenFlows," in *IEEE/IFIP Workshop Manag. Future Internet*, Apr. 2012, pp. 1167–1174.

[99] A. Kassler, L. Skorin-Kapov, O. Dobrijevic, M. Matijasevic, and P. Dely, "Towards QoE-driven multimedia service negotiation and path optimization with software defined networking," in *Proc. Int. Conf. Softw., Telecommun. Comput. Netw.*, Sep. 2012, pp. 1–5.

[100] H. Sun, A. Vetro, and J. Xin, "An overview of scalable video streaming," *Wireless Commun. Mobile Comput.*, vol. 7, no. 2, pp. 159–172, Feb. 2007.

[101] S. Jivorasetkul and M. Shimamura, "End-to-end header compression over software-defined networks: A low latency network architecture," in *Proc. Int. Conf. Intell. Netw. Collaborative Syst.*, Sep. 2012, pp. 493–494.

[102] E.-S. M. El-Alfy, "A review of network security," *IEEE Distrib. Syst. Online*, vol. 8, no. 7, Jul. 2007.

[103] ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, Aug. 2013.

[104] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Enabling fast failure recovery in OpenFlow networks," in *Proc. Int. Workshop Des. Reliable Commun. Netw.*, Oct. 2011, pp. 164–171.

[105] J. Kempf, E. Bellagamba, A. Kern, and D. Jocha, "Scalable fault management for OpenFlow," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2012, pp. 6606–6610.

[106] C.-J. Chung, P. Khatkar, T. Xing, J. Lee, and D. Huang, "NICE: Network intrusion detection and countermeasure selection in virtual network systems," *IEEE Trans. Dependable Secure Comput.*, vol. 10, no. 4, Jun. 2013.

[107] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson, "FRESCO: Modular composable security services for software-defined networks," in *Proc. Netw. Distrib. Syst. Security Symp.*, Apr. 2013.

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

[109] C. Schlesinger, Language-Based Security for Software-Defined Networks. [Online]. Available: http://www.cs.princeton.edu/cschlesi/isolation.pdf

[110] D. Kordalewski and R. Robere, A Dynamic Algorithm for Loop Detection in Software Defined Networks 2012. [Online]. Available: http://www.cs.toronto.edu/robere/paper/networkgraph-1214.pdf

[111] S. Costanzo, L. Galluccio, G. Morabito, and S. Palazzo, "Software defined wireless networks: Unbridling SDNs," in *Proc. Eur. Workshop Softw. Defined Netw.*, Oct. 2012, pp. 1–6.

[112] M. Mendonca, K. Obraczka, and T. Turletti, "The case for software-defined networking in heterogeneous networked environments," in *Proc. ACM Conf. CoNEXT Student Workshop*, 2012, pp. 59–60.

[113] K. Yap *et al.*, "Blueprint for introducing innovation into wireless mobile networks," in *Proc. ACM SIGCOMM workshop Virtualized Infrastructure Syst. Architectures*, 2010, pp. 25–32.

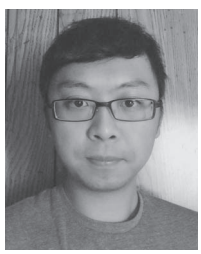
[114] L. E. Li, Z. M. Mao, and J. Rexford, "Toward software-defined cellular networks," in *Proc. Eur. Workshop Softw. Defined Netw.*, Oct. 2012, pp. 7–12.

[115] A. Mahmud and R. Rahmani, "Exploitation of OpenFlow in wireless sensor networks," in *Proc. Int. Conf. Comput. Sci. Netw. Technol.*, Dec. 2011, pp. 594–600.

[116] T. Luo, H.-P. Tan, and T. Q. S. Quek, "Sensor OpenFlow: Enabling software-defined wireless sensor networks," *IEEE Commun. Lett.*, vol. 16, no. 11, pp. 1896–1899, Nov. 2012.
[117] J. Chung *et al.*, "Experiences and challenges in deploying OpenFlow over real wireless mesh networks," *IEEE Latin Am. Trans. (Revista IEEE Am. Latina)*, vol. 11, no. 3, pp. 955–961, May 2013.
[118] L. Y. Ong, "OpenFlow/SDN and optical networks," in *Network Innovation Through OpenFlow and SDN: Principles and Design*. New York, NY, USA: Taylor & Francis, 2014.
[119] L. Liu, H. Guo, and T. Tsuritani, "OpenFlow/SDN for metro/backbone optical networks," in *Network Innovation Through OpenFlow SDN: Principles Design*. New York, NY, USA: Taylor & Francis, 2014.
[120] E. Mannie, "Generalized multi-protocol label switching (GMPLS) architecture," *IETF RFC*, 2004.
[121] S. Das, G. Parulka, and N. Mckeown, "Why OpenFlow/SDN can succeed where GMPLS failed," presented at the Eur. Conf. Exhib. Opt. Commun., Sep., 2012, Tu.1.D.1.
[122] L. Liu, T. Tsuritani, I. Morita, H. Guo, and J. Wu, "Experimental validation and performance evaluation of OpenFlow-based wavelength path control in transparent optical networks," *Opt. Express*, vol. 19, no. 27, pp. 26578–26593, Dec. 2011.
[123] L. Liu *et al.*, "Field trial of an OpenFlow-based unified control plane for multi-layer multi-granularity optical switching networks," *IEEE/OSA J. Lightw. Technol.*, vol. 31, no. 4, pp. 506–514, Feb. 2013.
[124] S. Azodolmolky *et al.*, "Integrated OpenFlow-GMPLS control plane: An overlay model for software defined packet over optical networks," *Opt. Express*, vol. 19, no. 26, pp. B421–B428, Dec. 2011.
[125] M. Channegowda *et al.*, "Experimental demonstration of an OpenFlow based software-defined optical network employing packet, fixed and flexible DWDM grid technologies on an international multi-domain testbed," *Opt. Express*, vol. 21, no. 5, pp. 5487–5498, Mar. 2013.
[126] C. H. Liu, A. Gkelias, Y. Hou, and K. K. Leung, "Cross-layer design for QoS in wireless mesh networks," *Wireless Pers. Commun.*, vol. 51, no. 3, pp. 593–613, Nov. 2009.
[127] L. Qiu and M. Song, "QoS oriented cross-layer design for supporting multimedia services in cooperative networks," in *Proc. Int. Conf. Service Sci.*, May 2010, pp. 314–318.
[128] J. Chen, T. Lv, and H. Zheng, "Cross-layer design for QoS wireless communications," in *Proc. Int. Symp. Circuits Syst.*, May 2004, vol. 2, pp. 217–220.
[129] D. Li, X. Hong, and D. Witt, "ProtoGENI, a prototype GENI under security vulnerabilities: An experiment-based security study," *IEEE Syst. J.*, vol. 7, no. 3, pp. 478–488, Sep. 2013.
[130] M. Smith, C. Schridde, B. Agel, and B. Freisleben, "Identity-based cryptography for securing mobile phone calls," in *Proc. Int. Conf. Adv. Inf. Netw. Appl. Workshops*, 2009, pp. 365–370.
[131] L. Sarakis, T. Zahariadis, H.-C. Leligou, and M. Dohler, "A framework for service provisioning in virtual sensor networks," in *EURASIP J. Wireless Commun. Netw., Special Issue Recent Advances Mobile Lightw. Wireless Syst.*, Apr. 2012, 135.
[132] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow," in *Proc. 2010 Internet Netw. Manage. Conf. Res. Enterprise Netw. (INM/WREN'10) USENIX Assoc.*, Berkeley, CA, USA, 2010, pp. 3–3.
[133] Z. Liu, Y. Li, L. Su, D. Jin, and L. Zeng, "M2cloud: Software defined multi-site data center network control framework for multi-tenant," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 517–518, Oct. 2013.
[134] A. Ishimori, F. Farias, I. Furtado, E. Cerqueira, and A. Abelém, Automatic QoS Management on OpenFlow Software-Defined Networks, 2012. [Online]. Available: http://siti.ulusofona.pt/aigaion/index.php/attachments/single/362
[135] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "PolicyCop: An autonomic QoS policy enforcement framework for software defined networks," in *Proc. IEEE SDN Future Netw. Serv. (SDN4FNS)*, Nov. 2013, pp. 1–7.
[136] B. Sonkoly *et al.*, "On QoS support to Ofelia and OpenFlow," in *2012 Eur. Workshop Softw. Defined Netw. (EWSDN), Publ.*, 2012, pp. 109–133.
[137] B.-Y. Ke, P.-L. Tien, and Y.-L. Hsiao, "Identity-based cryptography for securing mobile phone calls," in *Proc. Int. Conf. Adv. Inf. Netw. Appl. Workshops*, 2013, pp. 217–218.
[138] S. Gutz, A. Story, C. Schlesinger, and N. Foster, "Splendid isolation: A slice abstraction for software-defined networks," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw. (HotSDN'12). ACM*, New York, NY, USA, 2012, pp. 79–84.
[139] K. Pentikousis, Y. Wang, and W. Hu, "MobileFlow: Toward software-defined mobile networks," *Commun. Mag., IEEE*, vol. 51, no. 7, pp. 44–53, Jul. 2013.
[140] T. Luo, H.-P. Tan, P. C. Quan, Y. W. Law, and J. Jin, "Enhancing responsiveness and scalability for OpenFlow networks via control-message quenching," in *Proc. Int. Conf. CT Convergence (ICTC)*, Oct. 15–17, 2012, pp. 348–353.
[141] Y. Kanizo, D. Hay, and I. Keslassy, "Palette: Distributing tables in software-defined networks," in *Proc. IEEE INFOCOM*, Apr. 14–19, 2013, pp. 545–549.
[142] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng, "BalanceFlow: Controller load balancing for OpenFlow networks," in *Proc. IEEE 2nd Int. Conf. Cloud Comput. Intell. Syst. (CCIS)*, Oct./Nov. 2012, vol. 2, pp. 780–785.
[143] J. Biswas *et al.*, "The IEEE P1520 standards initiative for programmable network interfaces," *IEEE Commun. Mag.*, vol. 36, no. 10, pp. 64–70, Oct. 1998.
[144] A. Doria *et al.*, Forwarding and Control Element Separation (ForCES) Protocol Specification. [Online]. Available: http://tools.ietf.org/html/rfc5810
[145] T. V. 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.
[146] On the Difference Between Security and Safety: A Reference Answer is Provided in WiKi. [Online]. Available: http://wiki.answers.com/Q/1_Explain_the_difference_between_safety_and_security?#slide=1

Fei Hu received the Ph.D. degree in signal processing from Tongji University, Shanghai, China, in 1999 and the Ph.D. degree in electrical and computer engineering from Clarkson University, New York, NY, USA, in 2002. He is currently an Associate Professor with the Department of Electrical and Computer Engineering, The University of Alabama, Tuscaloosa, AL, USA. He has published over 150 journal/conference papers and book (chapters). His research has been supported by the U.S. NSF, Cisco, Sprint, and other sources. His research interests include cyber-physical system security and medical security issues; intelligent signal processing, such as using machine learning algorithms to process sensing signals; and issues on wireless sensor network design.

Qi Hao received the B.E. and M.E. degrees from Shanghai Jiao Tong University, Shanghai, China, in 1994 and 1997, respectively, and the Ph.D. degree from Duke University, Durham, NC, USA, in 2006, all in electrical engineering. His postdoctoral training in the Center for Visualization and Virtual Environment, The University of Kentucky, Lexington, KY, USA was focused on 3-D computer vision for human tracking and identification. From 2007 to 2014, he was an Assistant Professor with the Department of Electrical and Computer Engineering, The University of Alabama, Tuscaloosa, AL, USA. He is currently an Associate Professor with South University of Science and Technology of China, Shenzhen, China. His research has been supported by the U.S. NSF and other sources. His current research interests include smart sensors, intelligent wireless sensor networks, and distributed information processing.



Ke Bao is currently working toward the Ph.D. degree in the Department of Electrical and Computer Engineering, The University of Alabama, Tuscaloosa, AL, USA. His research interests include wireless networks, multimedia QoS, and wireless test beds.