INVITED SURVEY PAPER

A Survey on OpenFlow Technologies

Kazuya SUZUKI[†a)], ***Member***, **Kentaro SONODA**[††], **Nobuyuki TOMIZAWA**[†], **Yutaka YAKUWA**[†], **Terutaka UCHIDA**[†††], **Yuta HIGUCHI**[†††], ***Nonmembers***, **Toshio TONOUCHI**[†], ***and*** **Hideyuki SHIMONISHI**[†], ***Members***

SUMMARY The paper presents a survey on OpenFlow related technologies that have been proposed as a means for researchers, network service creators, and others to easily design, test, and deploy their innovative ideas in experimental or production networks to accelerate research activities on network technologies. Rather than having programmability within each network node, separated OpenFlow controllers provide network control through pluggable software modules; thus, it is easy to develop new network control functions in executable form and test them in production networks. The emergence of OpenFlow has started various research activities. The paper surveys these activities and their results.

key words: *OpenFlow, software-defined networking, network virtualization*

1. Introduction

The Internet has evolved to accommodate a variety of services, including real-time voice communications (IP telephony), IP-TV, on-line banking, sensor networking, and content delivery, as well as computer-to-computer communication. These services impose very diverse demands on networks. For example, real-time video delivery requires high bandwidth and a low packet loss rate, whereas non-real-time video delivery requires best-effort performance and high network bandwidth. This diversity strains the current network paradigm especially when it comes to the provision of quality of service (QoS), mobility, security, and traceability.

Indeed, traditional network technologies, such as Ethernet or IP, face a challenge that makes it hard for them to evolve or accommodate new services and technologies, and much effort has been made to accelerate the development of innovative network technologies. This new era of network research has already seen the creation of a number of initiatives focused on the "Future Internet". The main motivation of these initiatives is to give researchers, network service creators, network operators, equipment vendors, and others a way of easily developing, testing, and deploying their innovative ideas in a large network infrastructure. These initiatives include the National Science Foundation's Future Internet Design (FIND) [1] and Future Internet Architecture (FIA) [2], as well as the European Commission's Seventh Framework Programme (FP7) [3]. Large-scale testbed facilities have also been funded to accelerate the related research. They include the Global Environment for Network Innovations (GENI) [4] project sponsored by the National Science Foundation and the Japan Gigabit Network (JGN-X) [5] testbed sponsored by the National Institute of Information and Communications Technology.

Network virtualization technologies, such as the slice-based facility architecture [6], have enabled researchers to share testbeds at the same time. Rather than a single unified network substrate having a common set of control mechanisms applied to all applications or users that share the network substrate, this technology aims at creating multiple network "slices" having different control mechanisms for different users or applications. Virtualization technologies enable an evolutionary cycle in which a variety of virtual networks are easily created, some of which will soon disappear and some of which will be widely used. This birth-and-death and natural selection process help to ensure the continuous evolution of network architectures.

To accelerate this process, technologies that separate control and forwarding have been proposed [7]–[9]. They provide programmability in separate controllers rather than within each network node. This has enabled the control plane to independently evolve; the resulting short evolutionary cycle has led to a wide variety of control algorithms being developed. In contrast, the data plane has a relatively long evolutionary cycle and its developments are aimed at faster packet delivery. OpenFlow is the most popular of these technologies and is used in large network testbeds like GENI and JGN-X.

OpenFlow defines atomic operations of an OpenFlow-enabled switch to handle flows and an interface for instructing such operations from a separate controller. As OpenFlow defines a flow as a set of arbitrary combinations of packet header fields, it can be applied to flow-based fine-grained control as well as to aggregated control using a destination address or tunnel label. Researchers can easily design, test, and deploy their innovative ideas in experimental or production networks with these OpenFlow features. Programming at a (possibly centralized) controller improves such research productivity. Network research that exploits OpenFlow can be found everywhere. For example, we proposed a new network architecture based on OpenFlow [10].

Manuscript received June 25, 2013.
Manuscript revised October 19, 2013.

[†]The authors are with the Knowledge Discovery Research Laboratories, NEC Corporation, Kawasaki-shi, 211-8666 Japan.
[††]The author is with the Cloud System Research Laboratories, NEC Corporation, Kawasaki-shi, 211-8666 Japan.
[†††]The authors are with the Optical IP Development Division, NEC Corporation of America, California, USA.
a) E-mail: kazuya@ax.jp.nec.xom

DOI: 10.1587/transcom.E97.B.375

There has also been a lot of research on applying OpenFlow to data center networks, enterprise networks, besides carrier networks.

This paper surveys various research projects on OpenFlow technologies. The standardized OpenFlow specifications are presented in Sect. 2. After that, a survey of research is presented on applying OpenFlow to data center networks (Sect. 3), carrier networks (Sect. 4), and network security (Sect. 5). Section 6 presents platforms and frameworks for developing OpenFlow controllers, and surveys research on verifying, testing, and debugging the developed controllers. Section 7 concludes the paper.

2. Overview of OpenFlow

2.1 OpenFlow Specification

Here, we explain the standardized OpenFlow specifications [11], which mainly define the behavior of OpenFlow switches and the OpenFlow protocol that controls these OpenFlow switches. We briefly introduce OpenFlow switches and the OpenFlow protocol.

OpenFlow switches separate the forwarding element from the control element, whereas conventional switches have both control and forwarding elements (Fig. 1). The separated control element is called an OpenFlow controller. An OpenFlow controller connects OpenFlow switches with TCP or the transport layer security (TLS) [12].

An OpenFlow switch has a flow table for storing a set of flow entries; each flow entry consists of *header fields*, *counters*, and a set of *actions* to apply to matching packets. Header fields, which consist of the 12-tuples listed in Table 1, are used for matching packets. A header field can also be specified as a wildcard, in which case the other fields are only used for matching. For example, the "incoming port number is 1, and the destination MAC address is FF:FF:FF:FF:FF:FF" and the "L4 protocol is TCP, and the destination port number is 80" can be specified as header fields. Actions specify the operations to be applied to packets that match header fields. The specifications define the actions in Table 2. A flow entry can include multiple actions, e.g., "after rewriting the source IP address, output from a specified port" and "after outputting from a specified port, and output from another port".

As outlined in Fig. 1, the OpenFlow protocol is used by the controller to control switches. The OpenFlow protocol utilizes the messages summarized in Table 3. The most distinctive messages are explained below.

Packet In This message is used for sending a packet received by the switch to the controller, when there is no match in the flow table for that packet. The controller can use this message to create flow entries based on a packet received by the switch.

Packet Out This message is used for sending a packet from the controller to the switch. For example, a controller uses this message when it wants to output a packet created by itself from the switch, or when a packet in a "Packet In" message wants to be resent to an actual destination.

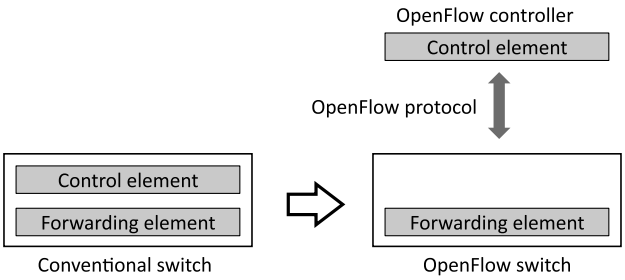

Fig. 1 Separation of forwarding and control.

Table 1 Header fields.

Field
Ingress port
Ethernet source address
Ethernet destination address
Ethernet type
VLAN ID
VLAN priority
IP source address
IP destination address
IP protocol number
IP ToS bits
Transport source port/ICMP type
Transport destination port/ICMP code

Table 2 Actions.

Action	Description
Forward	Forward a packet to a given port
Enqueue	Forward a packet through a given queue
Drop	Drop a packet
Modify-Field	Rewrite header fields

Flow Mod This message is used for sending flow entries created by the controller to the switch. The flow entries contain two timers for hard timeout and idle timeout. A flow entry with a hard timeout is removed after a specified time elapses from its installation. A flow entry with an idle timeout is removed if the flow entry was not referred for the specified time.

Flow Removed The switch informs the controller when a flow entry is removed from its flow table for some reason. The message includes statistical information, e.g., the reference counter and lifetime of the removed flow entry.

Barrier Request/Reply Switches that received a "Flow Mod" or "Packet Out" message do not report the completion of their operations to the controller. If the controller needs to know whether they are complete, it sends a "Barrier Request" message to the switch. After receiving a "Barrier Request" message, the switch completes the pending operations, which it received before the message, and then sends a "Barrier Reply" message.

Table 3 OpenFlow messages.

Message	Description
Messages sent from controller to switch	
Packet Out	Send packets for output from given port
Flow Mod	Send flow entries for setting flow table in switch
Port Mod	Change state of given port
Set Config	Set configuration parameters in switch
Messages to request from controller and reply by switch	
Features Request/Reply	Retrieve capabilities supported by switch
Stats Request/Reply	Collect statistics in switch
Get Config Request/Reply	Query configuration parameters in switch
Barrier Request/Reply	Confirm completion of operations requested by controller
Queue Get Config Request/Reply	Query queue configurations
Messages sent from switch to controller	
Packet In	Send packets received by switch
Flow Removed	Inform that flow entry expired
Port Status	Inform change in port status
Messages sent from both controller and switch	
Hello	For connection setup
Echo Request/Reply	Confirm liveness of controller-switch connection
Error	Inform of any errors
Vendor	For vendor-defined uses

2.2 Design Variations

The OpenFlow specifications are flexible enough to support many design variations in the behavioral model.

The OpenFlow protocol supports the programming of various switch behaviors at the flow level. A flow, on which a control is based, can be flexibly defined using arbitrary parts of a packet header, whereas classical switches and routers only use specific parts of the header. A flow can be fine grained by defining a flow with a combination of multiple parts of a header, or aggregated by defining a flow with only a specific part of a header. For example, a flow can be defined as a TCP/IP session or a tunnel. Various behaviors can be defined for a flow, e.g., sending its packets to specific ports, discarding its packets, and modifying specific header fields. Therefore, the behaviors of OpenFlow switches are not limited by the classical layered architecture and can be made up of arbitrary combinations of flow definitions and behavioral definitions. For example, a flow having a specific destination IP address can be destined to a specific port after rewriting the source and destination MAC headers. In this case, the switch behaves like an IP routers for the flow. Another example is defining a flow to be a specific TCP/IP session and instructing its packets to be discarded. In this case, the switch behaves like a router with ACLs (Access Control Lists) or a firewall.

As can be seen in Fig. 2, user programs on the controller can perform various network control tasks, including routing, path management, and access control, and they add flow entries to the flow tables in the switches. When a packet arrives at a switch, the switch searches for a flow entry matching the packet and performs the actions specified by the entry. This behavior can be reactive, i.e., dynamically injecting flow entries when a new flow arrives at the switch, or proactive, i.e., statically injecting flow entries in advance into the arriving packets.

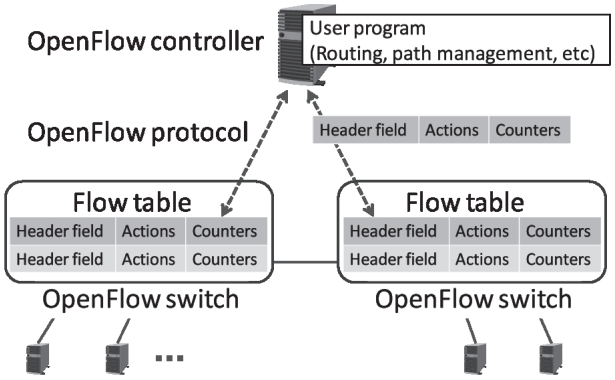

Fig. 2 OpenFlow architecture

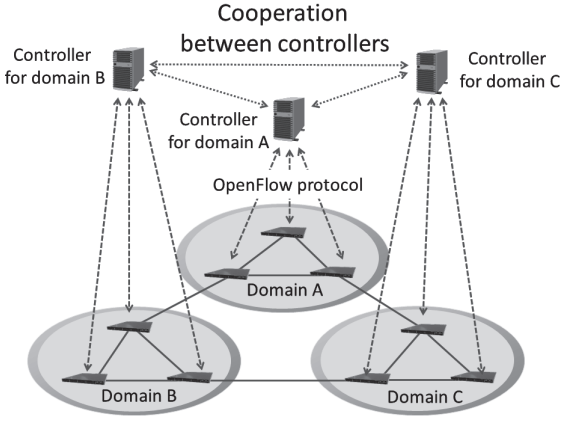

Fig. 3 Multiple controllers

An OpenFlow controller can be centralized by having a control server control all the switches, as shown in Fig. 2. Otherwise, multiple controllers can be deployed to cooperatively control the network for scalability, as shown in Fig. 3.

3. Data Center Networks

Data centers are used in various network services. However, data center networks built with conventional technologies are confronted with many problems, e.g., limits on scalability, virtualization, and configurability. Many researchers have undertaken studies in the hopes of solving these problems by using OpenFlow. This section introduces this research.

3.1 Topology

The fat tree in Fig. 4 is an attractive topology for data center networks. Conventional ethernet networks require loop-free topologies like trees. Tree topologies have characteristics that most traffic is concentrated at core switches, and these core switches require high capacity. Al-Fares et al. [13] applied the fat tree topology to data center networks. Since fat trees involve a multi-rooted topology, they can make the traffic share multiple core switches that are composed of low-capacity commodity hardware. The fat tree topology includes loop paths, and this requires a forwarding method that differs from the learning-based forwarding method of the conventional ethernet. Some researchers [14], [15] have proposed forwarding methods that use OpenFlow.

Portland [14] is a loop-free forwarding method on the fat tree topology using OpenFlow. The method for loop-free forwarding creates a pseudo-address encoding a forwarding path, which is called a pseudo-MAC (PMAC) address, and embeds it into the MAC address field in the packet. However, modifying MAC addresses forces end hosts to change their MAC addresses. To avoid this, the method makes edge switches translate between an actual MAC (AMAC) address and a PMAC address.

A forwarding method that requires the switch to have a large forwarding state capacity limits the scale of an L2 network. Portland relaxes this limitation by using PMAC addresses, which have hierarchical location information encoded into their prefixes. Encoding is designed to enable membership testing of an address to a switch subtree by prefix matching with a switch ID. Loop-free forwarding can then be implemented by forwarding packets up toward the root until they become members of the switch subtree and start forwarding down to the end host. PMAC forwarding can only be implemented using forwarding states that are proportional to the number of ports on a switch.

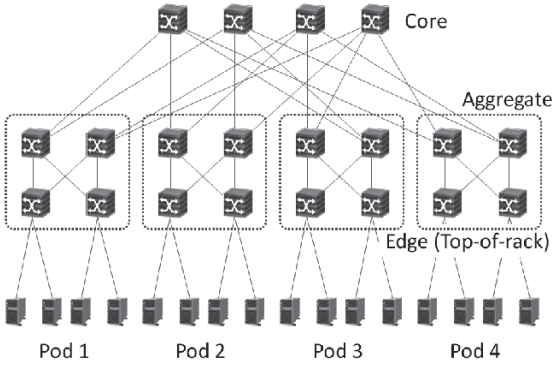

Fig. 4 Fat tree topology

Portland places a manager, which intercepts all ARP requests and responds to them. To enable switches to forward packets with PMAC addresses instead of AMAC addresses, the manager replies with a PMAC address. The sender host sends a packet whose destination address is the replied PMAC address, and the switches forward the packet to an edge switch connected to the actual destination host. To ensure that the destination host remains unaware that the MAC address of the packet differs from the actual address of the host, the edge switch rewrites the destination MAC address of the packet from a PMAC address to an AMAC address and sends the packet to the destination host. The manager is implemented as part of the OpenFlow controller.

Tavakoli et al. [15] estimated the number of flow entries installed in the switches in data center networks that use Portland. In their study targeting data center networks with 10K servers in 5000 racks, core switches needed 5000 entries (the most of all switches of the network). Because Portland uses location-based forwarding, the number of flow entries in the core switches is at most the number of top-of-rack (ToR) switches. The authors explained that, since this number is not much different from the number of access control lists supported by commodity conventional switches, Portland is applicable to large-scale data centers.

Heller et al. [16] proposed ElasticTree, which is a management method to reduce energy consumption in data center networks. It manages power consumption by dynamically turning switches on or off in response to changes in network usage. It determines whether switches are turned on or off by referring to multiple parameters such as topology, traffic, power models of switches, and fault tolerance properties. OpenFlow controllers install flow entries into the turned-on switches according to the results.

3.2 Scalability

Pries et al. [17] evaluated the performance of controllers that manage data center networks by using flow inter-arrival times measured by Benson et al. [18] at actual data centers. Their evaluations demonstrated that a centralized controller could control not only campus-wide networks but also data center networks. Since data center networks need to handle a huge number of flows over the processing capacity of the centralized controller, this causes data losses and long sojourn times in the controller.

DIFANE [19] is an architecture for distributed flow management that hierarchically distributes rules to switches and handles all data traffic in the fast path. The rules represent flow entries for forwarding or blocking specified flows. In the architecture, the OpenFlow controller makes flows hit on the OpenFlow switches as much as possible. It can handle wildcard rules and react quickly to network dynamics

such as changes to the network access policy or topology.

Yeganeh et al. [20] proposed Kandoo, a hierarchical method of controlling OpenFlow networks. The method deploys hierarchical OpenFlow controllers, local controllers, and a root controller. The local controller manages a group of switches and links between these switches, and it processes only local events, such as the arrival of traffic from inside the group or an inside link-down of the group. The root controller manages all local controllers and only handles events necessary for processes, such as the arrival of traffic at switches managed by different local controllers or a link-down between switches of different groups.

Macapuna et al. [21] tried to improve the scalability of controllers with a bloom filter, which is a probabilistic data structure for efficiently storing members of a set. They used the bloom filter to indicate a set of switches through which a packet goes. Their method separates the control of ToR switches from that of core switches, aggregate switches, etc., in order to achieve scalability. When a new flow arrives, the rack manager, which is an OpenFlow controller managing a ToR switch, calculates the end-to-end path and generates a bloom filter that includes switches along with the path. The rack manager then installs a flow entry to embed the bloom filter in the MAC address fields of the arrived packets. Flow entries created from the relationship between neighboring switches and output ports are proactively installed in the core and aggregate switches. A core or aggregate switch that has received a packet searches the bloom filter embedded in the MAC address field for a neighboring switch and forwards the packet to the neighboring switch. Their method makes each rack manager only control its corresponding ToR switch. As a result, their method, which is scalable, can be applied to large-scale data centers. Similar approaches have been taken by other researchers [14], [22].

3.3 Virtualization

Modern data centers need to accommodate large numbers of tenants that have isolated and independent networks. Virtual networks that are composed over physical networks are used to flexibly configure these networks of tenants. Thus far, VLAN tagging [23] has been widely used for creating virtual networks. However, the VLAN headers defined by [23] have a 12-bit tag field for identifying virtual networks, and so only 4094 networks can be created. However, virtual networks that overcome these limitations can be created by OpenFlow [24]. Because an OpenFlow controller can look in a "Packet In" message including a received packet, it can decide which virtual network the packet belongs to by incoming port or MAC address of the packet. Therefore, the OpenFlow controller can create virtual networks without using VLAN tagging.

The above research [24] could only be applied to networks consisting of OpenFlow switches. Barabash et al. [25] proposed a virtualized method by using an overlaying approach that utilizes tunnels between edge switches that connect virtual machines on a server with a physical network in order to achieve virtualization on existing physical networks. Packets forwarded to tunnels are encapsulated into tunnel headers and delivered over the underlying physical network. Their approach can be applied to an existing network without having to deploy any new facilities.

OpenFlow is useful (but not necessary) in the overlay approach. Pettit et al. [26] stated the importance of edge switching. The overlay approach is required to manage the relationship between overlay networks composed of tunnels and virtual machines that act on multiple servers at a data center. Their research revealed that a centralized OpenFlow controller can manage the relationship by controlling the edge switches on each server.

It is important to monitor virtual networks. Because more than one virtual network can be set up over one physical network, the performance of one virtual network will influence the others. In order to manage their performance, they need to be monitored. Monitoring is especially important in a data center network in which virtual networks are assigned to different customers. Argyropoulos et al. [27] proposed PaFloMon, a method of monitoring traffic through virtual networks. The monitoring database server is deployed in the OpenFlow network, and it collects monitoring data by using various protocols, i.e., sFlow, SNMP, and OpenFlow.

4. Carrier Networks

Some research projects have started to apply OpenFlow to carrier networks. Most carrier networks use IP and multiprotocol label switching (MPLS) [28] to forward traffic to their customers. Any new technologies for carrier networks are required to be able to work along with these forwarding mechanisms. Much of the OpenFlow research on carrier networks has focused on practical uses of MPLS/IP networks. This section surveys the OpenFlow research on MPLS and IP networks.

4.1 MPLS

Das et al. [29] tried to simplify the control of MPLS networks by using OpenFlow. Since conventional MPLS networks are large autonomous distributed systems, they require many control protocols. For example, each node in a MPLS network utilizes OSPF [30] to collect topology and bandwidth information about the network. When a label switch path is created, MPLS nodes use the signaling protocol, e.g., LDP [31] or RSVP-TE [32]. Das et al. claimed that a centralized control plane attained by OpenFlow makes these protocols unnecessary.

Early work on OpenFlow MPLS [33] involved extending OpenFlow specifications to MPLS nodes. OpenFlow version 1.0 specifications that target data center networks do not support MPLS. Kempf et al. proposed that three extensions are needed for this purpose.

1. Extending header fields to identify MPLS labels,

2. Adding new actions to push, pop, and swap MPLS shim headers, and
3. Adding virtual ports and tables for the virtual port.

These extensions were later included in the OpenFlow version 1.1 specifications [34].

Ferkouss et al. [35] proposed to use OpenFlow to change the roles of MPLS nodes. MPLS nodes play different roles depending on their positions in the network.

- *Label edge router* (LER), which is an MPLS node placed on the network border, classifies a received packet into a *Forwarding Equivalence Class* (FEC) and adds an MPLS shim header with a label assigned to each FEC to the packet.
- *Label switch router* (LSR), which is in the network core, swaps labels in a received packet and forwards it.
- Egress LERs have to remove the MPLS shim header from a received packet and to forward the packet through standard routing operations. In order to reduce the load on the egress LER, the LSR positioned one hop before the LER removes MPLS shim header from a received packet and sends it to the LER. The function of the LSR is called *Penultimate Hop Popping* (PHP), and it lets the LER do only the standard routing operations.

Ferkouss et al. stated that OpenFlow control could be used to make an MPLS node able to assign the above roles. This would make it easy to add (remove) nodes to (from) networks and to connect two MPLS networks. For example, an MPLS node assigned to LSR could install flow entries for forwarding packets in the network. More specifically, the flow entry would consist of match fields including the MPLS label and actions for swapping the MPLS label, decrementing an MPLS-TTL value, and outputting the packet. Another example is when an MPLS node is assigned to the ingress LER. The flow entry installed to the MPLS nodes includes match fields to identify the arrived packets as a flow (fine-grained or aggregated, as mentioned in 2.2) and actions for pushing an MPLS label and outputting the packets.

Path computation element (PCE) technology [36], which can control networks in a centralized way like OpenFlow, calculates an end-to-end path over multiple domains without their sharing internal topology information. PCEs deployed in each domain calculate an internal path that is part of the end-to-end path by using internal topology information collected by routing protocols such as OSPF. One PCE gathers the internal paths from the other PCEs and uses that information to determine an end-to-end path. The end-to-end path is then set up by using a signaling protocol such as RSVP-TE.

Giorgetti et al. [37] compared PCE and OpenFlow technologies by conducting a simulation of single-domain MPLS networks. The results of the path creation time showed that OpenFlow can create a path faster than PCE can because the signaling protocol takes more time for setting up a path calculated by PCE. On the other hand, an OpenFlow controller only sends flow entries directly to switches on the calculated path.

Although Giorgetti et al. presented only advantage of OpenFlow technologies, they have also disadvantage. For instance, PCE technology is designed to be utilized with existing routing and signaling protocols for MPLS networks. RSVP-TE has a function to determine whether a created path is available or not. In contrast, if MPLS networks are to be controlled by OpenFlow, the controller itself must have functions to collect topological information and maintain the created paths. As the size of the network increases, the load of these functions increases. Consequently, network operators hoping to use an OpenFlow controller must know the size of the network that the OpenFlow controller can control.

4.2 IP Networks

RouteFlow [38], previously called QuagFlow [39], is an open source project to provide virtualized IP routing services over OpenFlow networks. An OpenFlow controller needs to collect route information through routing protocols, e.g., OSPF or BGP [40]. For this purpose, RouteFlow prepares a virtual machine (VM) to run the routing engine and relates a physical interface with an interface of the VM to enable communication between the routing engine and neighbors. It thereby conserves on the labor involved in tightly integrating a routing engine into the OpenFlow controller. The OpenFlow controller of RouteFlow retrieves the route information from the VM and creates flow entries for IP routing services based on the collected information.

Kotani et al. [41] proposed multicast tree management for reliable streaming. It takes a lot of time to reconstruct multicast trees in traditional networks when a switch failure or a link failure occurs because multicast trees cannot be reconstructed until the unicast paths are stabilized. In such situations, significant packet losses, and, say, degraded video quality, cannot be avoided. One approach to solving this problem is to use redundant trees, but it is very hard to compute redundant trees in a distributed way. The proposed controller allows redundant multicast trees under IP multicast protocols. Kotani et al. devised an OpenFlow controller that supports IP multicast protocols; e.g., it snoops IGMP messages to manage multicast recipient groups, and it provides a method to set up multiple multicast trees for fast tree switching.

5. Security

Network security is becoming one of the hottest topics in OpenFlow research. The research can mostly be classified into two categories: network access control and attack detection and defense. This section introduces important research in these categories.

5.1 Network Access Control

Some of the research on access control uses OpenFlow to oversee network access policies. It aims at automating the isolation and configuration of complex physical and virtual networks. Centralized control by an OpenFlow controller prevents unauthorized access and information from being leaked because of misconfigurations by network administrators.

Casado et al. [42] proposed Ethane, an access control for enterprise networks by a centralized controller. The controller manages high-level global policies and translates them into local policies configured to individual switches. All these policies are expressed in groups and have rules. The groups are lists of users and services, e.g., http services of servers. The rules are permissions per service for each group. These policies are described in the original description language. Although Ethane itself does not uses OpenFlow, the following research using OpenFlow is based on the concept of Ethane.

Kim et al. [43] proposed Lithium, a method of event-driven access control for campus and home networks. Traffic patterns in campus networks are subject to time considerations, e.g., when the semester starts. Lithium defines a policy in four domains, i.e., time, users, flows and history, which are past traffic patterns, in order to control access according to the situation, e.g., time and traffic patterns. The method allows network operators to specify the policies and control flows according to the situation.

Watanabe et al. [44] proposed a method of roaming between campuses with flexible access control by using OpenFlow. Suppose that visitors attempt to access the network or the system of a campus. Their system obtains an access policy defined for each user from the policy database of their affiliation and installs flow entries for access control based on that policy. This is a flexible access control; e.g., permission could be granted for visiting researchers to access certain systems on campus during a conference for attendees by interworking between OpenFlow and the authentication system. Suenaga et al. [45] proposed a similar method to enable OpenFlow networks and Shibboleth [46], which is authentication system for single sign-on, to cooperate and make it possible for users to connect to applications between organizations.

5.2 Detection of Attacks and Defense Against Them

OpenFlow enables controllers to inspect all packets that arrive at networks and to make switches drop suspicious packets. This feature of OpenFlow can be put to good use in detect and defense against attacks. This subsection describes the research on using OpenFlow in this way.

Yao et al. [47] proposed VAVE, a method to protect against IP spoofing. VAVE embeds a source address validation module in an OpenFlow controller. The module hooks the "Packet In" messages and checks a white list of valid source addresses. To reduce the processing load, the method installs filtering rules for detecting and dropping invalid packets in the switches in advance.

Shin et al. [48] proposed CloudWatcher, another method of network security that differs from VAVE. The OpenFlow controller in their approach does not have any security modules, but instead has existing security appliances, such as intrusion detection systems. The OpenFlow controller captures packets that arrive at the network and forwards them into the security appliances that inspect all of them. CloudWatcher has a simple policy language to help network operators describe policies.

FRESCO [49] is a framework for the controller to easily implement security applications. In contrast with the other frameworks described in 6.1, FRESCO provides developers with an original scripting language to easily implement security modules for detection, protection, etc. FRESCO has a resource controller that manages flows entering the switches. If a flow table in which a security module has attempted to install a flow entry is full, the resource controller executes garbage collection on unimportant flow entries that were not installed by security modules.

Instead of the OpenFlow controller, OpenFlow switches can be used to inspect packets. Kumar et al. [50] proposed that OpenFlow switches be equipped with a function to detect intrusions. The switches have a flow table, which differs from the usual flow tables of standard OpenFlow switches, that contains a blacklist of source IP addresses and signatures of various attacks. The switches check ingress packets against the blacklist and signatures before looking at the usual flow tables, and they drop any anomalous packets that they discover.

6. Developing Controllers

OpenFlow is the first standard that has been accepted by both academia and industry for programmable networks. We can automate network control and management processes by using OpenFlow technology. However, the OpenFlow protocol controls the forwarding of packets, and its level seems to be too low for advanced network programming. Although we need a more advanced, productive programming foundation for network programming, OpenFlow is at present immature in this regard. Bugs will be inevitable if networks become dynamically programmable and configurable with software. Therefore, it is also important to provide various "debugging" techniques/methods for network operators and network programmers to make software defined networking more reliable. However, the networks in use today do not have any systematic approaches to making themselves more reliable.

There is, however, research on "programmable networks" (e.g., [51], [52]). Moreover, there is research on verification of network protocols [53], network configurations [54], implementation of network stacks [55] and discovery of network security vulnerabilities [56]. Moreover, with the spread of the OpenFlow protocol, research on pro-

Table 4 Platforms of the OpenFlow controller.

Platform	Language	Major contributor
NOX[61]	C++/Python	Nicira
POX[61]	Python	James McCauley
Floodlight[62]	Java	Big Switch Networks
Trema[63]	Ruby/C	NEC

gramming, verifying, and debugging OpenFlow-based networks has started accelerating. For example, McKeown [57] advocated the importance of formally verifying network behaviors and systematically identifying bugs and their root causes. He also introduced his own group's research on this topic (header space analysis [58], automatic test packet generation [59], and network debugger [60]).

This section mentions the currently available OpenFlow controller platforms of various open source projects and reviews the following three topics of current research:

1. Higher-level programming languages and domain specific languages (DSL) for OpenFlow,
2. Formal verification of OpenFlow network behaviors, and
3. Effective testing and trouble shooting techniques for OpenFlow networks.

6.1 Platform and Framework

There are various open source projects on making OpenFlow controller platforms for accelerating the development of controllers (Table 4).

NOX [64] is an OpenFlow controller platform supporting C++ and Python. NOX is an early development, and it is widely used in many research projects. However, development of NOX itself has finished. A subsequent project, POX, which supports Python, is now under development.

Floodlight is a controller platform for Java language that has a modular architecture consisting of controller and application modules. The controller modules provide functions which a majority of applications commonly use, such as those for discovering the network topology and controlling switches with the OpenFlow protocol. The application modules, which run with the controller modules, control the OpenFlow network as the user likes by giving them the means to easily set up firewalls, learning switches, and so on. The modular architecture enables diverse applications to be executed simultaneously.

Unlike other OpenFlow controller platforms, Trema [65] focuses on productivity; it provides not only a platform for OpenFlow controllers but also a modularized programming framework on top of it. Users can develop their own OpenFlow controllers by combining multiple control modules which they can develop themselves or obtain from others. In addition, Trema has an integrated network emulator, so that users can easily test the controllers they are developing.

Section 5 presented various security research on using OpenFlow. It is also important to ensure the security of the OpenFlow controller itself. In particular, FortNOX [66] has a kernel module with its own security functions for role-based authorization and conflict detection in flows created by NOX. The role-based authorization function strictly manages the production of flows in compliance with roles such as operators, secure applications, and non-security-related OpenFlow applications. The conflict detection function detects conflicts between rules defined by the administrator and requests to add, delete or modify a flow from NOX. A request is executed only if a conflict is not detected.

6.2 Higher Level Programming Language and DSL

The current mainstream OpenFlow controller platforms and frameworks (See 6.1) are based on event-driven styles that model the OpenFlow protocol as an event stream. Although frameworks in this style that resemble early window system's frameworks are easy to implement and enable fine-grained packet control over protocols, they are error-prone and hard to modularize except in small applications. Higher level programming languages or DSLs have recently been proposed for making OpenFlow-based programming easier and modular. These are based on the programming language concepts of logic programming [67], [68], functional reactive programming [69], [70], and declarative policies [71]. NetCore [72] and the hierarchical flow table (HFT) [73] are also declarative languages for packet forwarding descriptions, but they focus on using formal semantics to verify the behaviors of OpenFlow based systems.

6.3 Verification

Formal techniques are important because they can mathematically verify the "correctness" of OpenFlow controller programs and detect bugs in them. Substantial research has been carried out in this direction. For example, McGeer [74] described a framework to deal with the verification problem with OpenFlow networks as a satisfiability problem, and he discussed its computational cost. Reitblatt et al. [75] proposed update mechanisms for flow tables in an OpenFlow network with consistency before and after updates, and they formally proved the correctness of the mechanisms with a theorem prover called Coq. McGeer [76] proposed a similar update protocol for OpenFlow networks and manually proved it to be correct.

A number of tools have been developed to verify the properties of controller programs. NICE [77] verifies a Python program executed on a NOX controller by using model checking [78] (a technique for automatically verifying properties of a target system or software by representing the target as a finite automaton and exhaustively exploring it) together with symbolic execution [79] (a technique for analyzing a program to determine what inputs cause each part of it to execute by running the program with symbolic inputs and computing constraints to run each part). Verificare [80] translates a controller model written in VML, the language of Verificare, into a model written in one or more

input languages of off-the-shelf model checkers and verifies the model with these checkers.

There are tools that collect messages passed in OpenFlow networks and that dynamically verify them. FlowChecker [81] extracts the slice policies of controllers and flow tables of switches in networks with FlowVisor, which models a state machine that encodes the entire behavior of switches as a binary decision diagram (BDD) and verifies properties written in temporal logic, e.g., computational tree logic (CTL). VeriFlow [82] intercepts new rules sent to switches from controllers, builds forwarding graphs based on these rules, and explores them to check if any problems occur in the network after the rules are updated. Anteater [83] collects information on the network topology and devices' forwarding information bases (FIBs), translates them into a satisfiability problem (SAT) with an invariant to be verified, and checks if there are any potential bugs with an off-the-shelf SAT solver.

Furthermore, the compilers of some of the languages described in 6.2 have been formally verified as to whether they generate "correct" operations. For instance, Coq has been used to verify the translation rules of PANE [73], [84]'s compiler that generates operations from network policies written in the form of HFTs [73]. Guha et al. [85] used Coq to prove that the compiler of NetCore [72] is correct, and they also verified the correctness of a NetCore run-time system.

6.4 Testing, Trouble Shooting and Debugging

Networks are inherently physically, geographically and organizationally distributed systems of nodes and links that face difficulties such as asynchronicity and partial failure. It is therefore hard to test, debug, and troubleshoot the networks. In contrast, network operators and administrators do not have systematic techniques for accomplishing these tasks. Although there does not seem to be *any silver bullet* to solve this difficult problem, researchers have started to develop techniques and tools for testing, troubleshooting, and debugging OpenFlow networks.

As previously noted, NICE [77] tries to verify or find bugs in controller programs. In contrast, SOFT [86] tries to test the interoperability of OpenFlow switches through symbolic execution, and OFTEN [87] tries to test (physical) OpenFlow switches in a test environment with a scenario-based dummy OpenFlow controller.

It is also important to generate test packets to check whether the physical network is working correctly. Automatic test packet generation (ATPG) [59] tries to minimize automatically generated, periodically sent test packets for physical network testing with header space analysis [58].

Simulation is a traditional networking technique, especially for traffic engineering. Jin et al. [88] tried to extend their discrete time network simulator testbed with virtual-machine based emulation and parallel simulation to support OpenFlow.

It is also a hard task to troubleshoot running networks and large data centers. OFRewind [89] introduced a record/replay debugging mechanism into the OpenFlow network. The SDN trouble shooting simulator (STS) [90] is a similar trouble shooting tool that uses delta debugging to minimize 'input' data that reproduced invalid statuses. The network debugger (NDB) [60] enables operational OpenFlow networks to have breakpoint capabilities.

7. Conclusion

We introduced the various studies on OpenFlow technologies. After presenting standardized OpenFlow specifications, we surveyed research on applying OpenFlow to data center networks, carrier networks, and network security. Since it is important to develop controllers that effectively utilize OpenFlow, we presented platforms and frameworks to develop OpenFlow controllers and surveyed research on verifying, testing and debugging the developed controllers.

Although OpenFlow is a novel technology for controlling networks and has many distinctive features, conventional technologies can do most of what OpenFlow can do. Therefore, it is worthless to simply replace conventional networks with OpenFlow networks. It is important for researchers to have viewpoints as to how OpenFlow can be utilized to solve problems. We summarized these viewpoints and hope that this paper will be useful in future research.

Acknowledgements

This work was partly supported by the Ministry of Internal Affairs and Communications, Japan.

References

[1] "FIND (Future Internet Design)." http://www.nets-find.net/
[2] "NSF Future Internet Architecture Project." http://www.nets-fia.net
[3] "Seventh Framework Programme (FP7)." http://cordis.europa.eu/fp7/home_en.html
[4] "The Global Environment for Network Innovations (GENI)." http://www.geni.net/
[5] "New Generation Network Testbed JGN-X." http://www.jgn.nict.go.jp/english/
[6] L. Peterson, S. Sevinc, J. Lepreau, R. Ricci, J. Wroclawski, T. Faber, and S. Schwab, "Slice-based facility architecture," Tech. Rep., http://svn.planet-lab.org/attachment/wiki/GeniWrapper/sfa.pdf, 2009.
[7] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," ACM SIGCOMM Computer Communication Review, vol.38, no.2, pp.69–74, 2008.
[8] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. Van Der Merwe, "The case for separating routing from routers," Proc. ACM SIGCOMM Workshop on Future directions in network architecture, pp.5–12, 2004.
[9] M. Casado, T. Garfinkel, A. Akella, M.J. Freedman, D. Boneh, N. McKeown, and S. Shenker, "SANE: A protection architecture for enterprise networks," USENIX Security Symposium, 2006.
[10] H. Shimonishi, S. Ishii, S. Lei, and Y. Kanaumi, "Architecture, implementation, and experiments of programmable network using OpenFlow," IEICE Trans. Commun., vol.E94-B, no.10, pp.2715–2722, Oct. 2011.

[11] "OpenFlow Switch Specication Version 1.0.0 (Wire Protocol 0x01)." OpenFlow Switch Consortium, 2009.
[12] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246, Internet Engineering Task Force, 2008.
[13] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," ACM SIGCOMM Computer Communication Review, vol.38, no.4, pp.63–74, 2008.
[14] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "PortLand: A scalable fault-tolerant layer 2 data center network fabric," ACM SIGCOMM Computer Communication Review, vol.39, no.4, pp.39–50, 2009.
[15] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker, "Applying NOX to the datacenter," Proc. 8th ACM Workshop on Hot Topics in Networks (HotNets-VIII), 2009.
[16] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "ElasticTree: Saving energy in data center networks," Proc. 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI'10), p.17, 2010.
[17] R. Pries, M. Jarschel, and S. Goll, "On the usability of OpenFlow in data center environments," Proc. IEEE International Conference on Communications (ICC), pp.5533–5537, 2012.
[18] T. Benson, A. Akella, and D.A. Maltz, "Network traffic characteristics of data centers in the wild," Proc. 10th ACM SIGCOMM Conference on Internet Measurement (IMC), pp.267–280, 2010.
[19] M. Yu, J. Rexford, M.J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," ACM SIGCOMM Computer Communication Review, vol.41, no.4, pp.351–362, 2011.
[20] S.H. Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," Proc. ACM Workshop on Hot Topics in Software Defined Networks (HotSDN), pp.19–24, 2012.
[21] C.A. Macapuna, C.E. Rothenberg, and M.F. Magalhaes, "In-packet Bloom filter based data center networking with distributed OpenFlow controllers," Proc. GLOBECOM Workshops, pp.584–588, 2010.
[22] Y. Chiba, Y. Shinohara, and H. Shimonishi, "Source flow: Handling millions of flows on flow-based nodes," ACM SIGCOMM Computer Communication Review, vol.40, no.4, pp.465–466, 2010.
[23] "Virtual bridged local area networks," IEEE Std. 802.1Q, 2006.
[24] L. Sun, K. Suzuki, Y. Chiba, Y. Hatano, and H. Shimonishi, "A network management solution based on OpenFlow towards new challenges of multitenant data center," Proc. 9th Asia-Pacific Symposium on Information and Telecommunication Technologies (APSITT), IEICE/IEEE, 2012.
[25] K. Barabash, R. Cohen, D. Hadas, V. Jain, R. Recio, and B. Rochwerger, "A case for overlays in DCN virtualization," Proc. 3rd Workshop on Data Center-Converged and Virtual Ethernet Switching, pp.30–37, 2011.
[26] J. Pettit, J. Gross, B. Pfaff, M. Casado, and S. Crosby, "Virtual switching in an era of advanced edges," Proc. Workshop on Data Center–Converged and Virtual Ethernet Switching (DC-CAVES), 2010.
[27] C. Argyropoulos, D. Kalogeras, G. Androulidakis, and V. Maglaris, "PaFloMon — A slice aware passive flow monitoring framework for OpenFlow enabled experimental facilities," Proc. European Workshop on Software Defined Networking (EWSDN), pp.97–102, 2012.
[28] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol label switching architecture," RFC3031, Internet Engineering Task Force, 2001.
[29] S. Das, A.R. Sharafat, G. Parulkar, and N. McKeown, "MPLS with a simple OPEN control plane," Proc. Optical Fiber Communication Conference and Exposition, and the National Fiber Optic Engineers Conference (OFC/NFOEC), 2011.
[30] J. Moy, "OSPF Version 2," RFC 2328, Internet Engineering Task Force, 1998.
[31] L. Andersson, I. Minei, and B. Thomas, "LDP Specification," RFC 5036, Internet Engineering Task Force, 2007.
[32] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow, "RSVP-TE: Extensions to RSVP for LSP tunnels," RFC3209, Internet Engineering Task Force, 2001.
[33] J. Kempf, S. Whyte, J. Ellithorpe, P. Kazemian, M. Haitjema, N. Beheshti, S. Stuart, and H. Green, "OpenFlow MPLS and the open source label switched router," Proc. 23rd International Teletraffic Congress, pp.8–14, 2011.
[34] "OpenFlow switch specication version 1.1.0 implemented (Wire Protocol 0x02)," OpenFlow Switch Consortium, 2011.
[35] O. El Ferkouss, S. Correia, R. Ben Ali, Y. Lemieux, M. Julien, M. Tatipamula, and O. Cherkaoui, "On the flexibility of MPLS applications over an OpenFlow-enabled network," Proc. Global Communications Conference (GLOBECOM), 2011.
[36] A. Farrel, J.P. Vasseur, and J. Ash, "A path computation element (PCE)-based architecture," RFC 4655, Internet Engineering Task Force, 2006.
[37] A. Giorgetti, F. Cugini, F. Paolucci, and P. Castoldi, "Openflow and PCE architectures in wavelength switched optical networks," Proc. 16th International Conference on Optical Network Design and Modeling (ONDM), 2012.
[38] "The RouteFlow Project Web Site." https://sites.google.com/site/routeflow/
[39] M. Nascimento, C. Rothenberg, M. Salvador, and M. Magalhães, "Quagflow: Partnering quagga with openflow," ACM SIGCOMM Computer Communication Review, vol.40, pp.441–442, 2010.
[40] Y. Rekhter, T. Li, and S. Hares, "A border gateway protocol 4 (BGP-4)," RFC 4271, Internet Engineering Task Force, 2006.
[41] D. Kotani, K. Suzuki, and H. Shimonishi, "A design and implementation of OpenFlow controller handling IP multicast with fast tree switching," Proc. International Symposium on Applications and the Internet (SAINT), pp.60–67, 2012.
[42] M. Casado, M.J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," ACM SIGCOMM Computer Communication Review, vol.37, no.4, pp.1–12, 2007.
[43] H. Kim, A. Voellmy, S. Burnett, N. Feamster, and R. Clark, "Lithium: Event-driven network control," Technical Report GT-CS-12-03, Georgia Institute of Technology, 2012.
[44] T. Watanabe, S. Kinoshita, J. Yamato, H. Goto, and H. Sone, "Flexible access control framework considering IdP-Side's authorization policy in roaming environment," Proc. Computer Software and Applications Conference Workshops (COMPSACW), pp.76–81, 2012.
[45] M. Suenaga, M. Otani, H. Tanaka, and K. Watanabe, "Opengate on OpenFlow: System outline," Proc. 4th International Conference on Intelligent Networking and Collaborative Systems (INCoS), pp.491–492, 2012.
[46] "Shibboleth." http://shibboleth.net/
[47] G. Yao, J. Bi, and P. Xiao, "Source address validation solution with OpenFlow/NOX architecture," Proc. International Conference on Network Protocols (ICNP), pp.7–12, 2011.
[48] S. Shin and G. Gu, "CloudWatcher: Network security monitoring using OpenFlow in dynamic cloud networks," Proc. International Conference on Network Protocols (ICNP), 2012.
[49] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson, "FRESCO: Modular composable security services for software-defined networks," Proc. ISOC Network and Distributed System Security Symposium, 2013.
[50] S. Kumar, T. Kumar, G. Singh, and M.S. Nehra, "Open flow switch with intrusion detection system," International J. Schientific Research Engineering & Techonology (IJSRET), vol.1, pp.1–4, 2012.
[51] A.T. Campbell, H.G. De Meer, M.E. Kounavis, K. Miki, J.B. Vicente, and D. Villela, "A survey of programmable networks," ACM SIGCOMM Computer Communication Review, vol.29, no.2, pp.7–23, April 1999.
[52] P. Lin, J. Bi, H. Hu, T. Feng, and X. Jiang, "A quick survey on

selected approaches for preparing programmable networks," Proc. Asian Internet Engineering Conference (AINTEC), pp.160–163, 2011.
[53] G.J. Holzmann, Design and validation of computer protocols, Prentice-Hall, 1991.
[54] K. Bhargavan, D. Obradovic, and C.A. Gunter, "Formal verification of standards for distance vector routing protocols," J. ACM, vol.49, no.4, pp.538–576, 2002.
[55] M. Musuvathi and D.R. Engler, "Model checking large network protocol implementations," Proc. USENIX Symposium on Networked Systems Design and Implementation(NSDI), pp.12–12, 2004.
[56] R.W. Ritchey and P. Ammann, "Using model checking to analyze network vulnerabilities," Proc. IEEE Symposium on Security and Privacy, pp.156–165, 2000.
[57] N. McKeown, "Making SDNs Work." Invited talk at Open Networking Summit, 2012. Slide and talk video are available at http://tiny-tera.stanford.edu/˜nickm/talks/index.html
[58] P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: Static checking for networks," Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI), p.9, 2012.
[59] H. Zeng, P. Kazemian, G. Varghese, and N. McKeown, "Automatic test packet generation," Proc. International Conference on Emerging Networking Experiments and Technologies (CoNEXT), pp.241–252, 2012.
[60] N. Handigol, B. Heller, V. Jeyakumar, D. Maziéres, and N. McKeown, "Where is the debugger for my software-defined network?," Proc. ACM Workshop on Hot Topics in Software Defined Networks (HotSDN), pp.55–60, 2012.
[61] "NOX/POX Web Site." http://www.noxrepo.org/
[62] "Floodlight OpenFlow controller — Project floodlight." http://www.projectfloodlight.org/floodlight/
[63] "Trema: Full-Stack OpenFlow framework in ruby and C," http://trema.github.io/trema/
[64] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: Towards an operating system for networks," ACM SIGCOMM Computer Communication Review, vol.38, no.3, pp.105–110, 2008.
[65] H. Shimonishi, Y. Takamiya, Y. Chiba, K. Sugyo, Y. Hatano, K. Sonoda, K. Suzuki, D. Kotani, and I. Akiyoshi, "Programmable network using OpenFlow for network researches and experiments," Proc. 6th International Conference on Mobile Computing and Ubiquitous Networking (ICMU 2012), pp.164–171, 2012.
[66] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for OpenFlow networks," Proc. ACM Workshop on Hot Topics in Software Defined Networks (HotSDN), pp.121–126, 2012.
[67] T.L. Hinrichs, N.S. Gude, M. Casado, J.C. Mitchell, and S. Shenker, "Practical declarative network management," Proc. ACM Workshop on Research on Enterprise Networking (WREN), pp.1–10, 2009.
[68] N. Praveen Katta, J. Rexford, and D. Walker, "Logic programming for software defined networks," Proc. International Workshop on Cross-model Language Design and Implementation, 2012.
[69] N. Foster, R. Harrison, M.J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," Proc. ACM SIGPLAN International Conference on Functional Programming (ICFP), pp.279–291, 2011.
[70] A. Voellmy and P. Hudak, "Nettle: Taking the sting out of programming network routers," in Practical Aspects of Declarative Languages, pp.235–249, Springer, 2011.
[71] A. Voellmy, H. Kim, and N. Feamster, "Procera: A language for high-level reactive network control," Proc. ACM Workshop on Hot Topics in Software Defined Networks (HotSDN), pp.43–48, 2012.
[72] C. Monsanto, N. Foster, R. Harrison, and D. Walker, "A compiler and run-time system for network programming languages," Proc. ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL), pp.217–230, 2012.
[73] A.D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi, "Hierarchical policies for software defined networks," Proc. ACM Workshop on Hot Topics in Software Defined Networks (HotSDN), pp.37–42, 2012.
[74] R. McGeer, "Verification of switching network properties using satisfiability," Proc. IEEE International Conference on Communications (ICC), pp.6638–6644, 2012.
[75] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," Proc. ACM SIGCOMM Conference, pp.323–334, 2012.
[76] R. McGeer, "A safe, efficient update protocol for OpenFlow networks," Proc. ACM Workshop on Hot Topics in Software Defined Networks (HotSDN), pp.61–66, 2012.
[77] M. Canini, D. Venzano, P. Peresni, D. Kostic, and J. Rexford, "A NICE way to test OpenFlow applications," Tech. Rep. EPFL-REPORT-169211, École Polytechnique Fédérale de Lausanne, 2011.
[78] C. Baier and J.P. Katoen, Principles of Model Checking (Representation and Mind Series), The MIT Press, 2008.
[79] C. Cadar and K. Sen, "Symbolic execution for software testing: three decades later," Commun. ACM, vol.56, no.2, pp.82–90, Feb. 2013.
[80] R.W. Skowyra, A. Lapets, A. Bestavros, and A. Kfoury, "Verifiably-safe software-defined networks for CPS," Proc. ACM International Conference on High Confidence Networked Systems, pp.101–110, 2013.
[81] E. Al-Shaer and S. Al-Haj, "FlowChecker: Configuration analysis and verification of federated openflow infrastructures," Proc. ACM Workshop on Assurable and Usable Security configuration, pp.37–44, 2010.
[82] A. Khurshid, W. Zhou, M. Caesar, and P.B. Godfrey, "VeriFlow: Verifying network-wide invariants in real time," ACM SIGCOMM Computer Communication Review, vol.42, no.4, pp.467–472, Sept. 2012.
[83] H. Mai, A. Khurshid, R. Agarwal, M. Caesar, P.B. Godfrey, and S.T. King, "Debugging the data plane with anteater," ACM SIGCOMM Computer Communication Review, vol.41, no.4, pp.290–301, Aug. 2011.
[84] A.D. Ferguson, A. Guha, J. Place, R. Fonseca, and S. Krishnamurthi, "Participatory networking," Proc. USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE), 2012.
[85] A. Guha, M. Reitblatt, and F. Nate, "Machine-verified network controllers," Proc. Annual ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pp.217–230, 2012.
[86] M. Kuzniar, P. Peresini, M. Canini, D. Venzano, and D. Kostic, "A SOFT way for openflow switch interoperability testing," Proc. International Conference on Emerging Networking Experiments and Technologies (CoNEXT), pp.265–276, 2012.
[87] M. Kuzniar, M. Canini, and D. Kostic, "OFTEN testing OpenFlow networks," Proc. European Workshop on Software Defined Networking (EWSDN), pp.54–60, 2012.
[88] D. Jin and D.M. Nicol, "Parallel simulation of software defined networks," Proc. ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, pp.91–102, 2013.
[89] A. Wundsam, D. Levin, S. Seetharaman, and A. Feldmann, "OFRewind: Enabling record and replay troubleshooting for networks," Proc. USENIX Annual Technical Conference, pp.29–29, 2011.
[90] C. Scott, A. Wundsam, S. Whitlock, A. Or, E. Huang, K. Zarifis, and S. Shenker, "How did we get into this mess? Isolating fault-inducing inputs to SDN control software," Tech. Rep. UCB/EECS-2013-8, Electrical Engineering and Computer Sciences, University of California, Berkeley, Feb. 2013.

Kazuya Suzuki received the M.E. degree in Electrical Engineering from Tokyo Metropolitan University in 1997 and Ph.D. degree in Systems Management from the University of Tsukuba in 2011. He joined NEC Corporation in 1997, where he has been developing network equipment. He is currently with the Knowledge Discovery Research Laboratories, NEC Corporation. His research interests include improving availability in unicast and multicast routing and software-defined networking.

Kentaro Sonoda received the B.S. degree in computer science and engineering from University of Aizu, Japan, in 2000. From 2000 to 2005, he was with a systems integration company and worked in corporate & business development on security technologies & solutions. After that he joined NEC Corporation in 2005. Currently he works in Cloud System Research Labs., NEC, and has been engaged in research on technologies for a security with Software-Defined Networking.

Nobuyuki Tomizawa received his M.S. degree in Information Science from Tokyo Institute of Technology in 1994. He joined NEC Corporation in 1994. He now works in NEC's Knowledge Discovery Research Laboratories and is engaged in research on software-based technologies for reliable Software-Defined Networking.

Yutaka Yakuwa received his M. Engineering degree from Waseda University in 2009. He joined NEC Corporation in 2009 and has been undertaking research on formal methods. He now works in NEC's Knowledge Discovery Research Laboratories and is engaged in research on software-based technologies for reliable Software-Defined Networking.

Terutaka Uchida received the Bachelor's degree in Commerce from Meiji University, Tokyo, Japan. He is currently a researcher in the Optical IP Development Division, NEC Corporation of America. His research interests include networking and business communication software.

Yuta Higuchi received the Master's degree in Information Science from Nagoya University. He joined NEC Corporation in 2007, where he has been undertaking development of a platform framework for operation management software stacks. He is currently with the Optical IP Development Division, NEC Corporation of America. His research interests include architecture design of scalable frameworks for Software-De ned Networking using OpenFlow and other protocols.

Toshio Tonouchi is working in NEC Corporation. He received a master degree in Department of Information Science, Faculty of Science, University of Tokyo, and a Ph.D. from Graduate School of Information Science and Technology, Osaka University. He worked for OSI management platforms and development process and tools of the platforms. He was a visitor of Department of Computing, Imperial College, UK in 2000–2001, and he studied the policy-based management, there. He won Yamashita memorial award, IPSJ and ICM research award, IEICE. His research interest is how to develop reliable SDN networks efficiently.

Hideyuki Shimonishi received the M.E. and Ph.D. degrees from the Graduate School of Engineering Science, Osaka University, Osaka, Japan, in 1996 and 2002. He joined NEC Corporation in 1996 and has been engaged in research on traffic management in high-speed networks, switch and router architectures, and traffic control protocols. As a visiting scholar in the Computer Science Department at the University of California at Los Angeles, he studied next-generation transport protocols. He now works in NEC's Knowledge Discovery Research Laboratories and is engaged in research on technologies for future Internet architectures including OpenFlow. He is a member of the IEICE.