Computer Networks 81 (2015) 79–95

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Survey Paper

Software-Defined Networking: A survey

Hamid Farhady, HyunYong Lee *, Akihiro Nakao

Graduate School of Interdisciplinary Information Studies, The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan

ARTICLE INFO

Article history:
Received 30 January 2014
Received in revised form 16 February 2015
Accepted 17 February 2015
Available online 24 February 2015

Keywords:
Software-Defined Networking
Programmable networks
OpenFlow
Control and data planes

ABSTRACT

Software-Defined Networking (SDN) is considered promising to simplify network management and enable research innovations based on the decomposition of the control and data planes. In this paper, we review SDN-related technologies. In particular, we try to cover three main parts of SDN: applications, the control plane, and the data plane anticipating that our efforts will help researchers set appropriate and meaningful directions for future SDN research.

1. Introduction

Modern Internet infrastructure consists of a set of networking devices with purpose-built application-specific integrated circuits (ASICs) and chips that are used to achieve high throughput, thus realizing *hardware-centric networking*. However, the current hardware-centric Internet infrastructure suffers from several shortcomings, such as manageability, flexibility, and extensibility. Networking devices usually support a handful of commands and configurations based on a specific embedded operating system (OS) or firmware. As a result, network administrators are limited to a set of pre-defined commands, even though it would be easier, simpler, and more efficient to support more protocols and applications if it were possible to program network controls in ways that are more responsive and flexible. In addition, researchers usually have to make their own testbeds or take advantage of simulations rather than real world implementation scenarios to realize their ideas. In other words, innovation and research is costly under the current condition of hardware-centric networking.

To overcome such limitations, the Software-Defined Networking (SDN) concept has been proposed. SDN can be defined as "*an emerging network architecture where the network control is decoupled and separated from the forwarding mechanism and is directly programmable*" [1]. In SDN, there is a logically centralized controller that has a network-wide view and controls multiple packet-forwarding devices (e.g., switches) that can be configured via an interface (e.g., ForCES [2] and OpenFlow [3]). For example, an OpenFlow switch has one or more forwarding tables that are controlled by a centralized controller, thus realizing programmability in the control plane. Forwarding tables are used to control packets (e.g., forwarding or dropping). Therefore, according to the controller policy that manages the forwarding tables, an OpenFlow switch can act as a router, switch, NAT, firewall, or exhibit similar functions that depend on packet-handling rules. Due to its decoupled nature, SDN is believed to be a new networking technology that simplifies today's network operation and management and also enables network innovations and new network designs. Because of the potential benefits of SDN in the current Internet and future Internet architectures, such as information-centric networking [4], it has gained considerable attention from the community.

* Corresponding author.
E-mail addresses: farhadi@nakao-lab.org (H. Farhady), ifjesus7@gmail.com (H. Lee), nakao@nakao-lab.org (A. Nakao).

http://dx.doi.org/10.1016/j.comnet.2015.02.014
1389-1286/

An SDN instance consists of three major parts: application, control plane, and data plane (Fig. 1). The *application* label indicates a part that exploits the decoupled control and data plane to achieve specific goals, such as a security mechanism [5] or a network measurement solution [6]. Applications communicate with a controller at the control plane via the *northbound* interface of the control plane. The *control plane* is the part that manipulates forwarding devices through a controller to achieve the specific goal of the target application. The controller uses the *southbound* interface of the SDN-enabled switch to connect to the data plane. The *data plane* is the part that supports a shared protocol (e.g., OpenFlow) with the controller and handles the actual packets based on the configurations that are manipulated by the controller. Therefore, we believe that deeply understanding each part and investing balanced research attention into each of the three parts is important to maximize the potential benefits of SDN.

In this paper, we survey existing efforts on each part. In a similar spirit, several SDN survey papers [7–10] have been published recently. However, those papers mainly focus on the control plane and the application part, particularly OpenFlow-related work. On the other hand, in this paper, we try to cover all three parts anticipating that a survey of the three parts will help researchers set appropriate and meaningful objectives for future SDN research.

The rest of the paper is organized as follows. In Section 2, we briefly introduce the history of SDN and several SDN-related terms. Section 3 introduces examples of the application and Section 4 surveys the control plane related work. Section 5 surveys the data plane technologies. Finally, after we discuss several future directions for SDN research in Section 6, Section 7 concludes this paper.

2. Background

2.1. Networking paradigm

Networking paradigms can be divided into three types according to the deployment of control and data planes (Fig. 2). In the traditional hardware-centric networking, switches are usually *closed* systems that have their own control and data planes and support manufacturer-specific control interfaces. Therefore, in the traditional hardware-centric networking, deploying new protocols and services

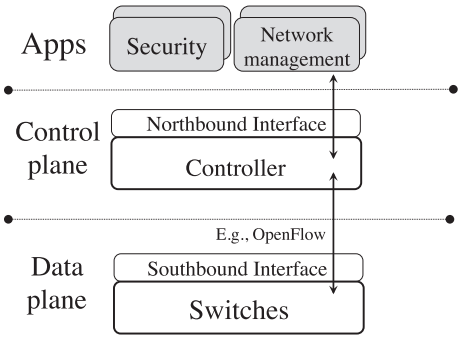

Fig. 1. Components of SDN.

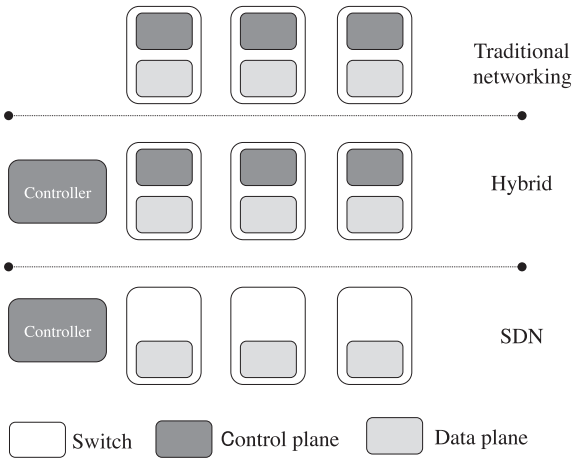

Fig. 2. High-level illustration of networking paradigms.

(and even new versions of existing protocols) is a challenge because all the switches need to be updated or replaced. In contrast, in SDN, switches become simpler (in terms of removing the control plane from the device) forwarding devices and a centralized controller derives the control mechanism of the network. This decomposition of control and data planes allows easier deployment of new protocols and services because the decomposition enables us to *program* switches via the controller. Finally, a hybrid approach supports both distributed and centralized control planes. For example, common commercial OpenFlow switches (see Section 5.1.2) are hybrid switches that support OpenFlow in addition to traditional operation procedures and protocols.

2.2. Historical foundations

Even though SDN is popular nowadays, several SDN concepts have been around for many years. In the following, we briefly review each of them.

2.2.1. Active networks

In active networks, each packet carries a program rather than raw data. When a network node receives a packet, the program inside the packet is executed and then different types of actions can be triggered against the packet (e.g., forward or drop) based on the data plane design. The idea of active networks relates to some in-network processing services and tries to treat network devices as an environment that reacts based on what the packet carries rather than passively transmitting bits from one node to another [11].

The active networking approach shows less interest in the control plane and is instead focused on providing a smart environment similar to end-point PCs compared with current dumb switches that can execute a limited set of procedures. That is, if we consider the end-points of a network, which are PCs, and servers as smart devices, then we can consider network controls (e.g., switches and routers) as dumb devices. They are dumb because they can execute limited types of tasks, albeit they do so rapidly.

Therefore, active networks help to create smarter network controls and intermediate nodes. In active networks, packets are the entities that control the behavior of a node, starting with booting it up and proceeding onto the method of handling a single packet. A series of studies followed the idea of active networking including network management (e.g., Smart packets [12]), dynamic protocol deployment (e.g., ANTS [13]), and network architecture studies (e.g., SwitchWare [14,15]). The main shared characteristic between SDN and active networks is fostering programmability. However, the current SDN definition mostly focuses on control plane programmability whereas active networks are more focused upon data plane programmability.

2.2.2. Open signaling

In the 1990s, through a series of workshops as well as DARPA's developments in active networking, the *Open Signaling (OPENSIG)* community expanded. As the name suggests, OPENSIG takes a telecommunication approach toward programmability. OPENSIG suggests modeling of communication hardware using open programmable interfaces that lead to open access to switches and routers; it acts as a facilitator for third-party software providers to enter the telecommunications software market. An interesting shared perspective between OPENSIG and SDN is a clear distinction between concerns. OPENSIG establishes a clear distinction between transport, control, and management in the network [16]. The idea first defines open programmable interfaces for network controls and then lets service providers manipulate the network state using a middleware toolkit such as common object request broker architecture (CORBA) to be able to introduce new services easier and faster.

2.2.3. Decomposition of control and data planes

Although active networks seek programmability in networking, some projects in the community look for separation of data and control planes to simplify the network architecture and provide some sort of abstraction. Tempest [17] was an ATM virtualization project intended to enable multiple coexisting virtual networks on a shared physical infrastructure including a software controller to manage forwarding decisions. Forwarding and Control Element Separation (ForCES [2]) is an Internet Engineering Task Force (IETF) standard which provides an interface between control and data plane. Routing Control Platform (RCP [18]), Path Computation Element, (PCE [19]) SoftRouter, [20] and Intelligent Route Service Control Protocol (IRSCP [21]), all foster a centralized approach to controlling the network. Security incentives also encourage centralization of the control plane. SANE [22] and Ethane [23] both devote some consideration to 4D [24] projects, and propose flow-level access control in the network. The original implementation of the OpenFlow API is based on the Ethane switch.

2.2.4. Other work

There are dozens of works in the literature that share objectives with, SDN but work from different perspectives. For example, XMILE [25] is a dynamic XML-based policy definition platform to automatically manipulate a large number of routers. Other examples include solutions for scalable control and management of an ATM network (such as DCAN [26]). As a complementary version of SNMP, NETCONF [27] allows the definition of a new API for network controls to modify configuration data.

2.3. SDN-related terms

There are several terms that are related to SDN. For clarity's sake, we will introduce those terms.

OpenFlow. There is a misconception that SDN and OpenFlow are equivalent. This is mainly due to the fact that the term SDN was coined after the introduction of OpenFlow [28] and OpenFlow is one representative API to setup an SDN instance. However, OpenFlow is an API used for the communication between the controller and switches, as shown in Fig. 1. In other words, OpenFlow is one way of controlling network-wide forwarding behavior under an SDN concept. Moreover, different APIs and tools (e.g., JunOS SDK [29] and Cisco ONE [30]) can be used to achieve similar objectives.

Network Virtualization. SDN and Network Virtualization (NV) [31] are closely related to each other in some aspects. NV enables simultaneous coexistence of multiple network instances on a shared physical infrastructure; thus, NV can be used to run an SDN solution. SDN can also be used to realize NV by distinguishing and forwarding flows to different slices.

Network Function Virtualization. Network Function Virtualization (NFV) [32] refers to the virtualization of specific in-network functions (e.g., firewalls, WAN optimizers, load balancers, and VPN gateways) that may be connected together to provide value-added services. A virtualized network function can be implemented based on one or more virtual networking devices running different software. Similar to active networks, NFV mainly focuses on data plane programmability. In this regard, NFV may be able to extend SDN in terms of data plane programmability because current SDN is more concerned with control plane programmability. Furthermore, an NFV solution (e.g., a virtual appliance) can operate in SDN.

3. Applications

The logically centralized controller-based SDN paradigm makes network information collection and dynamic networking policy changes easier. As a result, SDN techniques can be exploited to achieve specific purposes in various areas.

3.1. Specific purposes

3.1.1. Network management

An efficient network management requires information about current network status and timely change of network control. Aster*x [33] uses OpenFlow to measure the state of the network and directly control the forwarding paths. In [34], the authors utilize the network information (collected by the centralized controller) to improve the

network utilization and reduce packet losses and delays. In particular, they focus on the case where SDN is incrementally introduced into an existing network. In addition, they formulate the SDN controller's optimization problem for traffic engineering with partial deployment. To enrich network management services, FleXam [35] enables the OpenFlow controller to access packet-level information, and the authors in [36,37] try to incorporate application awareness into an SDN that is currently agnostic to applications. There have been some works that propose new network management systems using SDN technologies. In [38], the authors combine legacy network management functions (e.g., discovery and fault detection) with the end-to-end flow provisioning and control enabled by SDN. In [39], based on SDN technologies, the authors try to improve various aspects of network management, such as enabling frequent changes to network conditions and state, providing support for network configuration in a high-level language, and better visibility and control over tasks such as network diagnosis and troubleshooting. OpenSketch [6] is a software-defined traffic measurement architecture proposed to support more customized and dynamic measurement while guaranteeing the measurement accuracy. To achieve this goal, OpenSketch separates the measurement data plane (supporting many measurement tasks) from the control plane (configuring and allocating different measurement tasks). In [40], the authors propose an inter-domain routing solution using an NOX-OpenFlow architecture that was originally created only for routing in enterprise networks. Plug-n-Serve [41] is a web traffic load-balancing solution that leverages an OpenFlow controller to dynamically change switch configuration. Real-time policy checking [42,43] on the controller is proposed to keep OpenFlow rules consistent and thus control the network traffic in a stable way.

3.1.2. Middlebox

Middleboxes (such as a load balancer) and in-network services (such as in-network caching) are very popular in today's networks [44]. SDN technologies can be exploited to produce better middlebox-based services. SIMPLE [45] is an SDN-based policy enforcement layer for efficient middlebox-specific traffic engineering. SIMPLE exploits SDN technologies to ensure that the traffic is directed through the desired sequence of middleboxes to realize efficient middlebox applications. FlowTags [46] allows middleboxes to add tags to outgoing packets so that tags are used on switches and middleboxes for systematic policy enforcement. SDN facilitates to embed various in-network services, such as advertisement targeting [47], where a software control in the middle of the network injects related advertisements based on the session content. The general framework for developing in-network services or middlebox functionalities is discussed in [48,49].

3.2. Security

The centralized control of SDN is useful for implementing security applications. OpenFlow Random Host Mutation (OFRHM) [50] is a technique that hides network assets from external/internal scanners that try to discover network targets and launch attacks. In this approach, the OpenFlow controller dynamically allocates a random virtual IP (translated to/from the real IP of the host) to each host to avoid exposing an authentic IP that could be used by the attacker. In [51], the authors implement several traffic anomaly detection algorithms using OpenFlow compliant switches and an NOX controller to deal with network security problems in home and office networks. Through experiments, they show that the SDN-supported approach results in more accurate identification of malicious activities versus the ISP-driven approach. Resonance [52] secures enterprise networks where the switches enforce dynamic access control policies based on flow-level information. Resonance allows the switches to take actions (e.g., dropping) to enforce high-level security policies. Some studies use SDN capabilities to develop applications such as edge-based authentication gateways [53] or security application development frameworks [54].

3.2.1. Others

Virtual Machine (VM) migration is a promising way to ensure load balancing and power saving by VM reallocation. However, VM migration is a difficult task because it requires updates of the network state; this may lead to inconsistency and violations of service level agreement. In [55], the authors exploit the abilities of SDN (i.e., running algorithms in a logically centralized controller and manipulating the forwarding layer of switches) to handle VM migration issues. In particular, given the network topology, SLA requirements, and set of VMs that are to be migrated, along with their new locations, the algorithms (running in the SDN controller to orchestrate these changes within the network) output an ordered sequence of VMs to migrate, as well as a set of forwarding state changes. To support VM migration that is transparent to applications, LIME [56] clones the data plane state to a new set of switches and then incrementally migrates VMs by leveraging SDN technologies. The traditional IP multicast technique based on IGMP is not scalable because it consumes a large amount of resources such as IP multicast tables and CPU time. In [57], the authors propose to manage an IP multicast in overlay networks using OpenFlow. In particular, they eliminate periodic Join/Leave messages and achieve more than 4,000 tenants. It is also possible to combine SDN and NFV to provide a virtual networking lab for computer science education without any simulation [58].

3.3. Networks

3.3.1. Data center network

ElasticTree [59] is a network-wide energy optimizer that monitors data center traffic conditions and chooses the set of network elements that must stay active to meet the performance requirements. The authors discuss several strategies to find minimum-power network subsets. Scissor [60] tries to save energy by removing redundant traffic. Scissor replaces the redundant header information with a flow ID to be used for the forwarding. Scissor leverages SDN technologies to dynamically allow switches to route packets based on the flow IDs. NCP [61] supports scalable

service replication in data centers through SDN. NCP identifies flows based on network addresses and ports, and specifies a replication target for each identified flow. NCP then determines the ideal switch for replication and installs corresponding forwarding rules so that the identified flow goes to a designated server. In [62], the authors exploit wildcard rules of switches to direct large aggregates of client traffic to server replicas in a scalable way.

Some research tries to satisfy the need for customized routing and management of distributed data centers that cannot be easily achieved by a traditional WAN architecture. B4 [63] exploits OpenFlow to connect Google's data centers to satisfy massive bandwidth requirements, maximize average bandwidth, and enable rate limiting and measurement at the edge. The centralized traffic engineering of B4 drives links to nearly 100% utilization. By exploiting the centralized control, SWAN [64] aims at boosting the utilization of inter-data center networks. SWAN controls when and how much traffic each service sends, and re-configures the data plane to match current traffic demand. M2cloud [65] aims to support global network performance isolation for concurrent tenants by providing scalable network control for multi-site data centers. M2cloud employs two-level controllers to provide each tenant with flexible virtualization support in both intra- and inter-data center networks.

3.3.2. Optical network

The use of SDN technologies in optical transport networks may offer many benefits (e.g., improved network control and management flexibility) [66]. The potential benefits and challenges of extending SDN concepts to various transport network architectures (including optical wavelength, and fiber and circuit switches) are discussed in [67]. In [68], the combination of an SDN controller and optical switching is exploited to optimize application performance and network utilization of big data applications. It is shown that the combination has great potential to improve application performance with relatively small configuration overhead. In [69], it is argued that the use of SDN in the transport layers can enable packet-optical integration and improves transport network applications. The authors discuss extensions to OpenFlow v1.1 to achieve control of switches in the multi-technology transport layer. Furthermore, OpenFlow and GMPLS control plane integration is exploited for software-defined packet over optical networks [70]. Moreover, OpenFlow-based control plane architectures for optical SDN [71] and Flexi-Grid optical networks [72] are proposed. In [73], the authors introduce a QoS-aware unified control protocol for optical burst switching. OpenFlow is also used to dynamically create a bidirectional wavelength circuit for a TCP flow [74] or wavelength path control for light-path provisioning in transparent optical networks [75]. A simple programmable architecture is proposed in [76] to abstract a core transport node as a programmable virtual switch that meshes well with the SDN paradigm while leveraging OpenFlow protocol for control.

3.3.3. Wireless network

By exploiting a light virtual AP abstraction, Odin [77] introduces programmability in enterprise WLANs to support various functionalities (e.g., authentication, mobility, and AP association decision). Odin allows a network operator to implement enterprise WLAN services as network applications while not requiring client-side modifications. OpenRadio [78] proposes a programmable wireless data plane by providing modular and declarative programming interfaces across the entire wireless stack. OpenRadio can be used to realize modern wireless protocols, such as WiFi and LTE, while providing flexibility to modify the PHY and MAC layers. OpenRoads [79,80] aims to create an open platform where various mobility solutions, network controllers, and routing protocols are examined. OpenRoads provides control of the data path and device configuration using OpenFlow and SNMP, respectively. The centralized control of SDN can also be used to provide flow-based routing and forwarding capabilities in wireless mesh networks [81], to abstract all base stations in a local geographic area [82], or to achieve complete virtualization and programmability in radio access networks [83]. In [84], the authors argue that SDN can simplify the design and management of cellular data networks and enable new services. In [85], the authors exploit OpenFlow to optimize handovers in heterogeneous wireless environments, particularly for media-independent handover procedures.

3.3.4. Home network

Small networks, such as home networks, are becoming more complex to manage because the market is presently saturated with low-cost networking devices, which are popular among end users. SDN can be utilized to manage those networks as well. The authors in [51] argue that the advent of SDN provides a unique opportunity to effectively detect and contain network security problems in home networks and shows that SDN-enabled approaches can outperform alternatives. The authors in [86] present a design for a home router that focuses on monitoring and controlling network traffic flows to provide a user with control over his or her network behavior. A home network data recorder (prototyped after NOX) is proposed to manage and troubleshoot home networks [87]. By exploiting the programmable network switches that enable flexible remote management, the authors in [88] propose a way to outsource the management of home networks to a third party with operations expertise and a broader view of network activity.

3.3.5. Information-Centric Networking

Information-Centric Networking (ICN) [4] has been proposed to solve fundamental limitations of the current Internet in supporting content-oriented services. Because ICN takes a revolutionary approach, its development requires a new network architecture. In this regard, SDN is a valuable tool to build a testbed for ICN [89,90]. In [91], the authors discuss how ICN functionalities can be implemented over an OpenFlow network and how OpenFlow should be modified to better suit ICN functionalities. The question of how SDN and ICN could concretely be combined, deployed, and tested is addressed in [92]. The authors outline a possible realization of a novel design for ICN solutions and point to possible testbed deployments for future testing. SDN features can also be utilized

to realize ICN capabilities in an efficient manner. In [93], the authors present an OpenFlow-based architecture that performs efficient caching for ICN where caching strategy might dramatically influence performance and efficiency of ICN. C-flow [94] seeks to achieve efficient content delivery by leveraging the current OpenFlow functionalities. C-flow can deliver content to mobile hosts by using the byte-range option of the HTTP header. In addition, multicast/anycast is naturally supported by mapping between files and their corresponding IP addresses.

4. Control plane

The control plane acts as an intermediary layer between applications and the data plane. The control plane in SDN is called a *controller*; it communicates with the application via the northbound interface and with the switches via the southbound interface. Therefore, in the control plane, it is important to design interfaces and the controller itself in an efficient way. Thus, in this section, we survey existing efforts to realize and improve various aspects of the control plane. In particular, we focus on OpenFlow-related research because OpenFlow is prominently successful, whereas other approaches are not as successful in practice. In fact, other approaches for the control plane (e.g., IETF ForCES [2]) have not been adopted by the major router vendors, which has hampered incremental deployment [8], even though they can be used to achieve the same goals using a somewhat different SDN model [95].

4.1. OpenFlow

The OpenFlow protocol defines an API for the communication between the controller and the switches (Fig. 3). Therefore, both the controller and the switches should Understand the OpenFlow protocol. The controller manipulates the flow tables of the switch by adding, updating, and deleting flow entries. This happens reactively (when the controller receives a packet from the switch) or proactively according to the OpenFlow controller implementation. The controller communicates with the switches via a secure channel. The OpenFlow switch supports the flow-based forwarding by keeping one or more flow tables. Flow tables are used to determine how the given packets are handled. Each flow table entry contains a set of packet fields to match (e.g., Ethernet addresses and IP addresses), and a set of actions to be applied upon a match (e.g., send-out-port, modify-field, or drop). Flow entries also maintain counters to collect statistics for particular types of flow (e.g., the number of transmitted bytes). When the switch receives a packet that causes a miss in the forwarding table matching process, the action to be taken depends on the table-miss flow entry. That packet can be forwarded to the controller, passed to the next flow table, or dropped. When the packet is forwarded to the controller, the controller then decides how to handle this packet. The controller can drop the packet, or it can add a flow entry that gives the switch direction on how to forward similar packets in the future. From version 1.1, OpenFlow supports multiple flow tables and pipeline processing. Therefore, continuing the matching process on the next flow table can be one option for the non-matching packets. Various OpenFlow versions have been released, and features of each version are as follows. Version 1.2 supports a more flexible matching structure, which used to be a static fixed-length structure, and more protocols such as IPv6. Version 1.3 provides more support for protocols such as MPLS BoS bit and IPv6 extension headers. It also includes a better expression of switch capabilities (i.e., refactor capabilities negotiation) and improves metering facilities (e.g., per flow metering). Version 1.4 improves the OpenFlow extensible match introduced in v1.2 that gives more flexible classification capabilities to the user to match packet header fields and classify flows. Version 1.4 also supports optical ports. For more detailed description about the OpenFlow protocol, please refer to [96].

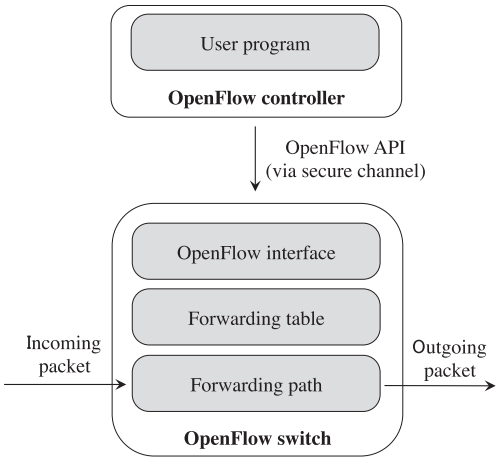

Fig. 3. OpenFlow.

4.2. Controller implementations

To date, various OpenFlow (compatible) controllers have been publicly released. Unless otherwise stated, the controllers we review here support OpenFlow v1.0 API.

- NOX [97] (C++/Python, Nicira): NOX is the first OpenFlow controller. NOX Classic were written in C++ and Python and current NOX is written in C++.
- POX [98] (Python, Nicira): POX is a general SDN controller that supports OpenFlow controller.
- SNAC [99] (C++, Nicira): SNAC is an OpenFlow controller based on NOX. It supports a graphical user interface and a policy definition language.
- Maestro [100] (Java, Rice University): Maestro tries to improve the system throughput by exploiting multicore processors and parallelism.
- Ryu [101] (Python, NTT): Ryu is a component-based SDN framework that supports OpenFlow v1.0, v1.2, and v1.3.
- MUL [102] (C, Kucloud): MUL supports a muli-threaded infrastructure and a multi-level northbound interface. It supports OpenFlow v1.0 and v1.3.
- Beacon [103] (Java, Stanford University): Beacon is a cross-platform and modular OpenFlow controller. It supports event-based and threaded operation.

- Floodlight [104] (Java, BigSwitch): Floodlight supports a broad range of virtual and physical OpenFlow switches and can handle mixed OpenFlow and non-OpenFlow networks.
- IRIS [105] (Java, ETRI): IRIS is a recursive OpenFlow controller that aims to support scalability, high availability, and multi-domain support.
- OESS [106] (Perl, NDDI): OESS is a set of softwares to configure and control dynamic VLAN networks using OpenFlow-enabled switches.
- Jaxon [107] (Java, independent developer): Jaxon is a NOX-dependent OpenFlow controller.
- NodeFlow [108] (JavaScript, independent developer): NodeFlow is an OpenFlow controller written for Node.JS.
- ovs-controller [109] (C, independent developer): ovs-controller is an OpenFlow reference controller packaged with Open vSwitch [110].
- Flowvisor [111] (C, ON.LAB): As a transparent proxy between OpenFlow switches and multiple OpenFlow controllers, Flowvisor allows multiple tenants to share the same physical infrastructure by dividing traffic flow space into slices.
- RouteFlow [112] (C++, CPQD): RouteFlow provides virtualized IP routing services over OpenFlow-enabled hardware. RouteFlow is composed by an OpenFlow controller, an independent RouteFlow server, and a virtual network environment.
- Helios [113] (C, NEC): Helios is an extensible OpenFlow controller that provides a programmable shell for performing integrated experiments (not publicly available).

4.3. Controller designs

In OpenFlow, a physically or logically centralized controller manages multiple switches. Therefore, the controller design significantly affects the overall performance of the network. Due to this, there has been much research on enhancing the controller design to improve the performance of various aspects.

4.3.1. State consistency

OpenFlow controllers and switches need to keep the same forwarding policy for stable forwarding. FlowAdapter [114] allows a consistent forwarding policy to be fitted into different types of hardware by converting the flow entry rules of the controllers to switch the hardware flow table pipeline. A similar approach is discussed in [115], which proposes a set of axioms for policy transformation to enable rewriting of rules across multiple switches while preserving the forwarding policy. Palette distribution framework [116] decomposes large SDN tables into small ones and then distributes them across the network while preserving the overall SDN policy semantics. Efficient rule-placement algorithms are also proposed in [117] to distribute forwarding policies across SDN-inspired networks while managing rule-space constraints. ONOS [118] maintains consistency across the distributed state by providing a network topology database to the controller. The idea of a network information base is also proposed in [119] so as to satisfy the state consistency and durability requirements. HOTSWAP [120] supports the upgrade of an SDN controller to a new version in a disruption-free manner by maintaining a history of network events managed by the old controller. HOTSWAP bootstraps the new controller by replaying the history. In the case where multiple controllers are used, HyperFlow [121] localizes the decision-making to individual controllers by synchronizing network-wide views of OpenFlow controllers to minimize the control plane response time for data plane requests. The controller uses transactional semantics to achieve consistent packet processing [122]. Some other works try to understand various aspects of the state consistency. In [123], it is shown that control-state inconsistency significantly degrades the performance of logically centralized control applications. The trade-off between update time and rule-space overhead [124] and the trade-off between maintaining consistency during configuration updates and the update performance [125] are also studied.

4.3.2. Scalability

The centralized control of SDN naturally faces scalability issues. Many works try to improve the scalability, particularly by reducing the overhead of the centralized controller in various aspects. DIFANE [126] keeps all traffic in the data plane by selectively directing packets through intermediate switches that store the necessary rules, so as to avoid a bottleneck at the controller and achieve better performance and scalability. For this, the controller partitions the forwarding rules over the switches. Similarly, DevoFlow [127] handles short flows at the OpenFlow switch while only large flows are directed to the controller. Kandoo [128] applies two layers of controllers to limit the overhead of frequent events on the control plane so as to realize scalable SDN. Controllers at the bottom layer try to handle most of the frequent events using the state of a single switch. A logically centralized controller of the top layer is only used to handle the events that cannot be handled by the controllers at the bottom layer. A hybrid control model that combines both central control and distributed control is proposed in [129] to provide more relaxed and scalable control. By defining new features such as the proactive flows to be activated under certain conditions, they try to reduce the overhead of the central controller. The scalability issue in specific areas is also studied. SDN can be a natural platform for network virtualization, but scalability becomes an issue in supporting a large number of tenants. To handle this scalability issue, FlowN [130] exploits a database technology to efficiently provide each tenant the illusion of its own topology and controller. To support scalable SDN slicing, AutoSlice [131] introduces a distributed hypervisor architecture that can handle a large number of flow table control messages from multiple tenants. OpenFlow protocol can be used to realize a load balancer for data centers by dividing traffic over the server replicas. However, the current OpenFlow approach may lead to a huge number of rules for the switches and a heavy load on the controller. In [62], the authors argue that the controller should exploit the wildcard rules to direct large aggregates of client traffic to

server replicas in a more scalable way. They present several methods to compute wildcard rules to achieve a target distribution of the traffic and to adapt to changes in load-balancing policies. Several scalability trade-offs in SDN design space are discussed in [132].

4.3.3. Flexibility

Some works try to support the same functions in a more efficient and flexible way. ElastiCon [133] manages the controller pool that is dynamically expanded or reduced according to traffic conditions. In addition, the load is dynamically shifted across controllers to gracefully handle the traffic. The reconfigurable match tables model is proposed in [134] to permit the forwarding plane to be changed without modifying hardware, and to enable the programmer to modify all header fields much more comprehensively than in OpenFlow. To realize far more flexible processing of counter-related information, the software-defined counters that utilize general-purpose CPUs rather than ASIC-based inflexible counters is introduced in [135]. In a similar spirit, a CPU in the switches is used to handle not only the control plane but also data plane traffic to overcome the limitations of the ASIC-based approach [136]. In [137], the authors present research directions that could significantly reduce TCAM and control plane requirements through classifier sharing and reuse of existing infrastructure elements. In [138], the authors propose the extensible session protocol to proactively install flow entries based on the requirements of the application. The goal of this work is to provide a general and extensible protocol for managing the interaction between applications and network-based services, as well as between the devices. Modularity is one of the key factors in managing complexity of any software system, including SDN. In [139], the authors introduce new abstractions for building applications from multiple, independent modules that jointly manage network traffic to ensure that rules installed to perform one task do not override other rules.

4.3.4. Security

OpenFlow architecture may suffer from trust issues on OpenFlow applications because it allows third-party development. The abuse of such trust could lead to various types of attacks that impact the entire OpenFlow network. One general way to deal with this potential trust issue is to constrain the applications. PermOF [140] allows a minimum privilege to the applications. The authors summarize a set of permissions to be enforced at the API entry of the controller. Similarly, ForNox [141] provides role-based authorization and security constraint enforcement for the NOX OpenFlow controller. FRESCO [54] provides a programming framework to execute and link security-related applications. Some works try to unveil potential vulnerabilities of SDN. In [140], the authors discuss several vulnerabilities of the OpenFlow protocol as it is currently deployed by hardware and software vendors. In [142], the authors introduce a new attack that fingerprints SDN networks and launches further efficient-resource- consumption attacks (e.g., DDOS-like attacks that issue many requests to the centralized controller supporting the reactive-mode). In [143], the authors describe several threat vectors that may exploit the SDN vulnerabilities and sketch the design of a secure and dependable SDN control platform.

4.3.5. Availability

The OpenFlow controller and switches should be robust in various circumstances. In [144], the authors discuss a runtime system that automates failure recovery by spawning a new controller instance and replaying inputs observed by the old controller. The controller can install static rules on the switches to verify topology connectivity and locate link failures based on those rules [145]. For fast recovery, frequent issuing of monitoring messages is needed, but this may place a significant load on the controller. To realize scalable fault management, in [146], the authors propose to implement a monitoring function in the switches so that those switches can send monitoring messages without placing a load on the controller. RuleBricks [147] is a system to embed high availability support in existing OpenFlow policies by introducing three key primitives: drop, insert, and reduce. In this work, the authors describe how these primitives can express various flow assignments and backup policies.

4.3.6. Placement

In the controller-driven SDN architectures, the position of the controller may impact the performance of an SDN. In [148], the authors try to answer the following question: "given a topology, how many controllers are needed, and where should they go?" To answer this question, they examine fundamental limits to controlling the plane propagation latency. Through experiments, they show that the answers depend on the topology and that one controller location is often sufficient to meet existing reaction-time requirements. In [149], the authors try to understand the impact of the latency between an OpenFlow switch and its controller. They show that latency drives the overall behavior of the network and bandwidth affects the number of flows that the controller can process. In [150], the authors solve the controller placement issue in such a way that the reliability of control networks is maximized. They present several placement algorithms along with a metric to characterize the reliability of SDN control networks.

4.4. Development tools

There have been many approaches to assisting various aspects of SDN development.

4.5. Simulators and frameworks

- Mininet [151]: Mininet emulates multiple virtual OpenFlow switches, end hosts, and controllers on a single machine. Mininet supports OpenFlow v1.3 and NOX.
- ns-3 [152]: ns-3 supports OpenFlow v0.89.
- OMNeT++ [153]: OMNeT++ supports OpenFlow v1.2 through a plugin.
- EstiNet 8.0 OpenFlow network simulator [154,155]: It can simulate thousands of OpenFlow v1.3.2 and v1.0.0 switches and run the real-world NOX, POX, Floodlight, and Ryu controllers.

- Trema [156]: Trema is a framework for developing OpenFlow controllers in Ruby and C.
- Mirage [157]: Mirage is an exokernel for constructing network applications across a variety of platforms and supports OpenFlow.

4.6. Debuggers

- Cbench [158]: Cbench tests OpenFlow controllers by emulating a bunch of OpenFlow switches and generating packet-in events for new flows.
- NICE [159,160]: Based on the combination of the model checking and symbolic execution, NICE tests OpenFlow controller programs for the NOX controller platform to find potential programming errors that make communications less reliable.
- SDN Troubleshooting System (STS) [161]: To find the erroneous codes that lead the controller software to break in an automatic way, this tool simulates the devices of a given network and allows a user to programmatically generate tricky test cases.
- Oflops [162,163]: Oflops quantifies an OpenFlow switch performance in various aspects by supporting a modular framework for adding and running implementation-agnostic tests.
- OFTest [164]: OFTest is a Python-based framework that tests basic functionality of OpenFlow switches.
- OFRewind [165,166]: OFRewind enables a user to record the control and data traffic and to replay it at an adaptive rate for network-level troubleshooting and debugging.

There also have been several works on improving OpenFlow development. Inspired by gdb (i.e., the debugger for software programs), ndb [167] is proposed to debug network control software in SDN. ndb pinpoints the sequence of events that lead to a network error by using familiar debugger actions such as breakpoint and backtrace. To troubleshoot bugs in SDN control software, the authors in [168] propose cross-layer correspondence checking (to find what problems exist and where in the control software the problem first developed) and simulation-based causal inference (to identify the minimal set of events that triggered the problem). VeriFlow [42] tries to check network-wide invariants in real time without affecting the network performance. As a layer between the controller and the switches, VeriFlow dynamically checks for network-wide invariant violations while each forwarding rule is inserted. ATPG [169] is an approach for testing and debugging networks including SDNs. Based on router configurations, ATPG generates a set of test packets to exercise network links and rules to detect both functional (e.g., incorrect forwarding rules) and performance problems (e.g., a congested queue). HSA [43] is a protocol-agnostic framework that allows checking of network specifications and configurations to identify an important class of network failures, including those of SDNs (e.g., reachability failures), regardless of the protocols running.

4.7. Configuration languages

Some works try to assist SDN development by providing high-level abstractions. To maintain the simplicity of the centralized programming model of SDN that can be compromised by challenges in managing low-level details, Maple [170] simplifies SDN programming by allowing a user to decide the behavior of an entire network based on a standard programming language. In addition, it provides an abstraction that runs on every packet entering a network to overcome the challenge of translating a high-level policy into sets of rules on distributed switches. To enable fault-tolerant network programs, FatTire [171] supports a new programming construct based on regular expressions that allow a user to specify the set of paths that packets may take through the network and the degree of fault tolerance required. Some studies suggest policy declaration languages such as FML [172], which uses NOX to calculate and deploy flow table entries on switches, or Procera [173], which also handles the controller architecture. The Frenetic project [174] proposes ideas for designing a compiler, run-time environment, and modular programming constructs for SDN [172], as well as constructs for certain tasks such as updates [175].

4.8. Testbeds

Several testbeds support OpenFlow protocol.

- Planetlab Europe [176]: It supports OpenFlow through sliver-ovs that is a modified version of Open vSwtich. Experimenters can create an OpenFlow overlay network by specifying the links between PlanetLab Europe nodes.
- OpenFlow in Europe: Linking Infrastructure and Applications (OFELIA) [177]: OFELIA allows researchers to not only experiment on a test network but to control and extend the network itself precisely and dynamically.
- National LambdaRail (NLR) [178]: Experimental (i.e., NetFPGA) and commercial (i.e., Hewlett Packard 6600) OpenFlow platforms are deployed within several NLR Points of Presence (PoPs) to enable researchers to create and run experiments using realistic scenarios.
- Internet2 [179]: It supports the NDDI-substrate mostly consisting of interconnected OpenFlow-capable switches. NEC G8264 switch and NOX controller are used.
- Research Infrastructure for large-Scale network Experiments (RISE) [180]: This testbed is a large scale OpenFlow testbed on JGN-X network in Japan.
- SURFnet [181]: It is an OpenFlow testbed linked to the Internet2 network.
- COTN [182]: COTN deploys OpenFlow-enabled switches into the backbone of CENIC's CalREN network and connects with emerging OpenFlow testbeds within the national research networks such as NLR and Internet2.
- FIBRE [183]: FIBRE project is a Future Internet research platform that hosts joint experiments between Brazilian and European researchers.

Moreover, the original Emulab does not support OpenFlow, but there is a research that tries to add functionality to support OpenFlow [184].

4.9. Industry standardizations and forums

There have been several approaches for standardizing, coordinating, and implementing SDN and related technologies such as NFV and cloud computing. ONF [185], backed by some university research centers such as ONRC [186], introduces an OpenFlow standard that is a vital element of the current SDN architecture. In particular, they focus on standardizing the northbound and southbound interfaces. The open source software community, including OpenDaylight [187], OpenStack [188], and CloudStack [189] is developing basic building blocks for SDN implementation for the advancement of SDN. For example, OpenDaylight is intended to be extensible and configurable to potentially support emerging SDN open standards (e.g., OpenFlow, I2RS, VxLAN, PCEP). The IETF Overlay (NVO3, L2VPN, TRILL, LISP, PWE3), API (NETCONF, ALTO, CDNI, XMPP, SDNP, I2AEX), Controller (PCE, FORCES), Protocol (IDR, IS-IS, OSPF, MPLS, CCAMP, BFD), and Interface to the Routing System (I2RS) [190] working groups are also involved with SDN-related activities. For example, the IETF FORCES working group [2] defines protocols and interfaces for the separation of IP control and forwarding, centralized network control, and the abstraction of network infrastructure. Furthermore, the IRTF initiated an SDN working group to contribute to the community. Even though IEEE is still not involved deeply with SDN, we can see some 802.1 Overlay Networking projects that look at SDN-related concepts. The Software-Defined Networking Research Group (SDNRG) [191] studies SDN from various perspectives (e.g., scalability, abstractions, and programming languages) to identify approaches that can be used in near-term and future research challenges. Some ITU-T [192] study groups are working on requirements for SDN. ITU-T SG13 is investigating SDN-related questions and network virtualization. ITU-T SG11 is developing signaling requirements and protocols for SDN, and this work aligns with the functional requirements and architectures developed by ITU-T SG13. Joint Coordination Activity on SDN (JCA-SDN) is established for coordinating and helping plan work to ensure that the ITU-T SDN standardization progresses in a well-coordinated manner among relevant SGs (e.g., SG13 on use-cases, requirements, and architecture; SG11 on protocols and interoperability). In addition, the Metro Ethernet Forum (MEF) [193] is now introducing SDN technologies to the Carrier Ethernet paradigm. Finally, ETSI owns an initiative on Network Function Virtualization.

Table 1 summarizes existing efforts for the control plane implementations.

5. Data plane

One basic and primary function of the data plane is packet forwarding. In addition, the data plane can support some types of in-network services to realize various services such as in-network caching and transcoding. In this section, we survey existing efforts and implementations for the data plane. Please note that there are not many SDN-related works in this aspect due to the early vision of SDN, as we mentioned before. Consequently, the following survey includes works that have been performed in different areas. We introduce these works because we believe that these technologies can be used to improve or realize the functions of the data plane.

5.1. OpenFlow switches

5.1.1. Software switches

Several OpenFlow software switch implementations are publicly available. Unless otherwise stated, OpenFlow v1.0 is supported.

- OpenFlow reference implementation [194] (C): This is a minimal OpenFlow stack that is used to track OpenFlow specification. The code base is now considered as an archive (supports v1.1.0 specification) and transferred to the ONF website [185].
- Open vSwitch [110] (C/Python): Open vSwitch is a an OpenFlow stack that is used both as a vswitch in virtualized environments and has been ported to multiple hardware platforms. Open vSwitch currently supports multiple virtualization technologies including Xen/XenServer, KVM, and VirtualBox.
- CPQD ofsoftswitch13 [195] (C/C++): ofsoftswitch is an OpenFlow compatible user-space switch implementation. It supports OpenFlow v1.3.

Table 1
Control plane implementations.

	Name	Language	Institute	Version
OpenFlow controllers	NOX	C++/Python	Nicira	1.0
	POX	Python	Nicira	1.0
	SNAC	C++	Nicira	1.0
	Maestro	Java	Rice Univ	1.0
	Ryu	Python	NTT	1.3
	MUL	C	Kucloud	1.3
	Beacon	Java	Stanford	1.0
	Floodlight	Java	BigSwitch	1.0
	IRIS	Java	ETRI	1.0
	OESS	Perl	NDDI	1.0
	Flowvisor	C	ON.LAB	1.0
	RouteFlow	C++	CPQD	1.0
	Jaxon	Java	–	1.0
	NodeFlow	JavaScript	–	1.0
	ovs-controller	C	–	1.0
	Helios	C	NEC	1.0
Tools	Simulation & Framework	Mininet, ns-3, OMNeT++, Trema EstiNet 8.0, Mirage, Wakame-vdc		
	Debuggers	OFlops, Cbench, NICE, STS OFTest, OFRewind		
Testbeds	PlanetLab Europe, OFELIA, NLR, Internet2, RISE, SURFnet, COTN, FIBRE			
Standards	ONF, IETF, SDNRG, ITU-T, IRTF, IEEE, ETSI, MEF			

- Indigo [196] (C): Indigo supports OpenFlow on physical and hypervisor-based switches.
- OpenFaucet [197] (Python): As a pure Python implementation of OpenFlow protocol, OpenFaucet can implement both switches and controllers.
- Pantou [198] (C): Pontou is an OpenFlow port to the OpenWRT wireless environment.

5.1.2. Commercial switches

Currently available commercial OpenFlow switches are as follows.

- Arista: 7050 series (v1.0)
- Brocade Communications: MLX Series, NetIron CES series, NetIron XMR series, ICX 7750 switch (v1.0)
- Dell Force10: Dell Force10 Z9000, Dell Force10 S-Series S4810 (v1.0)
- Extreme networks: Summit X440, X460, X480, and X670 (v1.0)
- HP: 3500 series, 3800 series, 5400 series, 6200 series, 6600 series, and 8200 series (v1.0)
- IBM: RackSwitch G8264 (v1.0)
- Juniper: MX-series (v1.0)
- Larch networks: the extendible Linux-based OpenFlow switch (v1.0)
- NEC: PF5240, PF5248, PF5820, and PF1000 virtual switch (v1.0)
- NoviFlow: NoviSwitch 1248 and NoviSwitch 1132 (v1.3)
- Pica8: P-3290, P-3295, P-3780, and P-3922 (v1.3)

Table 2 summarizes OpenFlow-compatible software and commercial switches.

Table 2
OpenFlow switches.

	Name	Language	Ver.
Open source	Open vSwitch	C/Python	1.0
	ofsoftswitch13	C/C++	1.3
	Indigo	C	1.0
	OpenFaucet	Python	1.0
	Pantou	C	1.0
Commercial	Vendor	Model	Ver.
	Arista	7050 series	1.0
	Brocade		1.0
	Dell Force10	Dell Force10 Z9000 and S-Series S4810	1.0
	Extreme	Summit X440, X460, X480, and X670	1.0
	HP	3500, 3800, 5400, 6200, 6600, and 8200 series	1.0
	IBM	RackSwitch G8264	1.0
	Juniper	MX-series	1.0
	NEC	NoviFlow 1248, 1132	1.3
	Pica8	P-3290, P-3295, P-3780, and P-3922	1.3
	Larch networks	OpenFlow switch	1.0

5.2. Data plane technologies

5.2.1. Packet forwarding

To improve OpenFlow switching, network processor-based acceleration cards are used in [199]. The authors show that their proposed approach reduces the packet delay by 20% while maintaining comparable packet forwarding throughput compared to other existing OpenFlow reference designs. The authors in [200] describe an approach to improve lookup performance of OpenFlow switching in Linux using a standard commodity network interface card. They show that their approach improves the packet switching throughput by 25% compared to regular software-based OpenFlow switching. In [201], the authors compare OpenFlow switching, layer-2 Ethernet switching, and layer-3 IP routing performance. In particular, forwarding throughput and packet latency in underloaded and overloaded conditions with different traffic patterns are analyzed. System scalability is also analyzed using different forwarding table sizes.

Parallelism is one promising way to improve forwarding performance. PacketShader [202] exploits the parallelism of a GPU to handle computation and memory-intensive tasks and achieves a traffic handling capacity of 40 Gbps. SP4 [203] is a software-based programmable packet processing platform. It leverages multicore processors and parallelism. Using a pipelined architecture plus parallelism, SP4 can achieve a reasonable throughput. FLARE [204] exploits many core general purpose processors and an optimized version of Click. FLARE can process 40 Gbps of traffic and inherits all the programmability characteristics of Click. Some works [205,206] study how parallelism affects the performance of software-based packet forwarding. In [206], the authors investigate different architectural setups of a software router using Click to ascertain the main performance limitation. They conclude that memory access delay is the primary bottleneck of the system, and software routers are sensitive to this parameter. In [205], an optimization framework is proposed for parallel processing of packets. The framework produces an optimal mapping of flows to process cores based on server resources and the resource consumption of flows. RouteBricks [207] realizes a high-speed software router based on multiple servers. Servers are connected using a full mesh of wires and use some randomization to pass packets from the servers that receive the packet to the server that sends the packet out of the router. A load-balanced routing algorithm spreads traffic between sender and receiver nodes in multiple paths without any centralized scheduling mechanism. HyperSplit [208] achieves 100 Gbps of packet classification throughput based on a tree data structure of rules. Each level of the tree forms a step of the classification pipeline on the FPGA. HyperSplit processes a pair of packets in each cycle and supports 50,000 rules in the tree.

5.2.2. In-network services

Data plane programmability is the key to supporting various in-network services. ClickOS [209] supports various applications such as NATs, rate-limiters, and application accelerators by exploiting the Click modular router. FLARE [204] seeks to enable programmability of the data plane so as to realize arbitrary in-network services in a modular manner based on the Click router architecture. NetOpen switch [210] is a programmable networking

switch node that supports in-network processing. The NetOpen switch node prototype is based on NOX. OpenRadio [78] supports a programmable wireless data plane by dividing wireless protocols into the processing and decision planes. This decoupling allows a user to program the platform while hiding the underlying complexity of execution. Odin [77] is a programmable AP for WLAN service applications. The AP is virtualized using a light hypervisor (i.e., OpenWrt) and then two services are run on top. OpenFlow is used to pass incoming packets to appropriate service VMs and then back to the network. The virtualized WiFi AP can also be used to realize Ad targeting that embeds advertisement into the result set of multiple search engines [47].

In-network services require additional processing that consumes the switch's resources, such as CPU, memory, and storage. Therefore, we need an elaborative resource allocation method that does not degrade the performance of the switch. In [211], the authors propose an offline prediction method that predicts the performance degradation of a resource on a specific workload in the case of resource contention among applications. Rather than using an expensive, purpose-built devices for in-network services, software-based tools can be exploited. SSLShader [212] is a software SSL accelerator that exploits the parallelism of a GPU and achieves processing speeds of about 13 Gbps.

6. Future directions

6.1. Data plane programmability

From our survey shown in previous sections, we can see that research on data plane programmability is absent from most SDN-related work, even though supporting data plane programmability is technically possible. We believe that this is mainly due to the early vision for SDN that focused on control plane programmability. However, we argue that research on both the data and control planes is required to enrich SDN services and applications. With well-balanced support for both planes, we may be able to define SDN as a network architecture that decomposes the control and data planes to provide and penetrate programmability deeply and comprehensively into the networking stack. We refer to the total solution that covers programmability of both the data and control planes as *Deeply Programmable Network (DPN)* (Fig. 4). One of the current efforts to realize DPN is FLARE [204]. FLARE is a deeply programmable network node using many core general-purpose processors as the hardware platform. FLARE aims to enable data and control plane programmability while guaranteeing high performance and scalability of the device.

Data plane programmability has the potential to enrich SDN applications. For example, because the data plane (i.e., switches) can look inside the packet, data plane programmability enables us to realize in-network caching, transcoding, compression, and encryption efficiently. Without data plane programmability, a flow may need to be forwarded to the controller for those applications; sending a packet back and forth to a controller may take time and cause inefficiency in the network. The data plane programmability may also enable us to handle new protocols. To realize data plane programmability in an efficient way, we may need to concern ourselves with the following. i) *APIs:* Opening up some interfaces that expose resources of a switch such as storage (e.g., for in-network caching), processors (e.g., for transcoding and encryption), and packet queues (e.g., dynamic adjustment of queue sizes for specific applications) could be useful. Furthermore, standard and open interfaces for resources on network controls such as storage, processing, or packet queues are a handy tool for SDN data plane application programmers to develop their solution faster and more portable across hardware platforms. ii)*Tools:* Similar to the tools for the control plane programmability, we need tools to simulate, debug, and test data plane applications and APIs. Moreover, unlike the debuggers for the control plane that use formal methods to debug forwarding table entries, we need different types of debuggers that can test data plane APIs because they mostly deal with packets and payloads themselves.

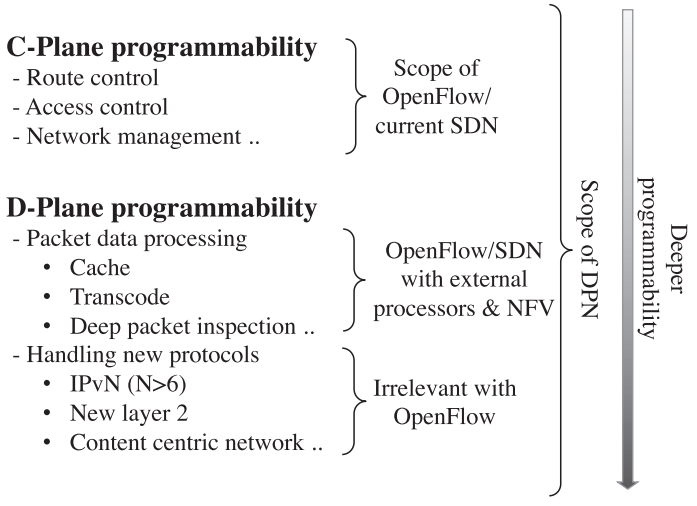

Fig. 4. Deep programmability within network.

6.2. Platform independency

Not depending on specific hardware or protocols enables us to be more productive and innovative. Traditional hardware-centric networking and current OpenFlow-compatible switches both make networking dependent on a special family of hardware or software, which hampers innovation and extensibility. Therefore, using commodity hardware (e.g., CPU, GPU, and NPU) can be an important move for creating and independent SDN data plane. Similarly, in terms of software-related issues, we can develop SDN In such a way that it suffers from less dependencies. For example, protocol-oblivious forwarding [213] is proposed to remove any dependency of protocol-specific configurations on the forwarding elements and enhance the data-path with new stateful instructions or actions [214] to support genuine SDN behavior [215,216]. ONF is still in early development stages of concepts such as protocol independent/oblivious/agnostic forwarding (i.e., PiF, PoF, and PaF) and they are all products of data plane programmability. We believe that we need to depart from the current ⟨pattern-match, action⟩ architecture of SDN.

6.3. User-driven control

Most SDN APIs are developed to be used by network operators and/or administrators. Although these APIs are useful and should be considered, there can be some APIs implemented by users (i.e., authorized users who can program their own software controls to execute their tasks based on hardware and software interfaces of the network operator) (e.g., [217,218]). APIs for end-users can be utilized to realize on-demand services. For example, an intrusion detection application on a user machine could request the network to filter traffic from a specific source. Alternatively, a MapReduce-style application could request bandwidth guarantees to improve performance of its shuffle phase. These examples mean that an API needs to exist between the network control plane and its client applications. These principals need both read and write access to learn the network's status and make independent configuration changes for their own benefit, respectively [218]. For the user-defined control, several challenges need to be resolved. Trust may play an important role in such APIs because we must give a portion of network control to a semi-external entity. We also may need to resolve conflicts across user requests while maintaining baseline levels of fairness and security.

6.4. Deployment

Early SDN deployments mainly focused on university campus networks and data centers. However, recent works have explored applications of SDN to a wider range of networks, including optical, home, wireless, and cellular networks, as well as ICN as described in Section 3. Because each network has its own settings, applying SDN to those networks may introduce new opportunities and challenges to be solved [8]. Another potential issue is an incremental deployment. Most existing works that exploit SDN assume a full SDN deployment. But, in the real world, only part of the network can be upgraded at a time when budgets are constrained. In this case, one challenge is to support compatibility between existing network components and SDN-enabled components (e.g., [219]). Another issue is determining which existing switches or routers need to be upgraded to maximize the benefits of SDN (e.g., [220]). We also may need to consider inter-networking across multiple SDN domains. Most current SDN-related work focuses on the context of a single administrative domain. The current focus is well-suited to SDN's logically centralized control. However, the logically centralized control may not be suited to the case of multiple SDN networks where the network controls are independently driven by their own controllers. For example, a federation of physically distributed SDN networks (including testbeds) needs agreement from the corresponding administrators. In addition, the controller may need to be extended to cover inter-networking.

7. Conclusion

SDN is becoming popular due to the interesting features it offers that unlock innovation in how we design and organize networks. However, there are still important challenges to be solved before realizing successful SDN. In this survey, we introduce existing SDN-related technologies and also argue that data plane programmability needs to be considered in the SDN definition. In particular, we provide data plane technologies and discuss several future directions to realize data plane programmability. In fact, current southbound APIs are not flexible and are mostly translated as OpenFlow. If we can define a better southbound API, then we can add interesting operation and management features and facilities to the data plane. We believe the SDN community should consider data plane programmability as a facilitator for new SDN applications.

References

[1] O.N. Foundation, Software-Defined Networking (SDN) Definition. <http://goo.gl/O2eTti>.
[2] A. Doria, R. Gopal, H. Khosravi, L. Dong, J. Salim, W. Wang, Forwarding and Control Element Separation (Forces) Protocol Specification. <https://ietf.org/wg/forces/charter/>.
[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, OpenFlow: enabling innovation in campus networks, ACM SIGCOMM CCR 38 (2) (2008) 69–74.
[4] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, B. Ohlman, A survey of information-centric networking, IEEE Commun. Mag. 50 (7) (2012) 26–36.
[5] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, M. Tyson, Fresco: modular composable security services for software-defined networks, in: Proceedings of Network and Distributed Security Symposium, 2013.
[6] M. Yu, L. Jose, R. Miao, Software defined traffic measurement with opensketch, in: USENIX NSDI, vol. 13, 2013.
[7] B. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, T. Turletti, A survey of software-defined networking: past, present, and future of programmable networks, IEEE Commun. Surv. Tutorials PP (99) (2014) 1–18.
[8] N. Feamster, J. Rexford, E. Zegura, The Road to SDN: An Intellectual History of Programmable Networks. <http://goo.gl/X8TCjy>.
[9] A. Lara, A. Kolasani, B. Ramamurthy, Network Innovation using OpenFlow: a survey, IEEE Commun. Surv. Tutorials 16 (1) (2014) 493–512.

[10] F. Hu, Q. Hao, K. Bao, A survey on Software Defined Networking (SDN) and OpenFlow: from concept to implementation, IEEE Commun. Surv. Tutorials PP (99) (2014) 1.

[11] D.L. Tennenhouse, D.J. Wetherall, Towards an active network architecture, ACM SIGCOMM CCR 26 (2) (1996) 5–17.

[12] B. Schwartz, A.W. Jackson, W.T. Strayer, W. Zhou, R.D. Rockwell, C. Partridge, Smart packets for active networks, in: IEEE OPENARCH, 1999, pp. 90–97.

[13] D.J. Wetherall, J.V. Guttag, D.L. Tennenhouse, Ants: a toolkit for building and dynamically deploying network protocols, in: IEEE OPENARCH, 1998, pp. 117–129.

[14] D.S. Alexander, W.A. Arbaugh, M.W. Hicks, P. Kakkar, A.D. Keromytis, J.T. Moore, C.A. Gunter, S.M. Nettles, J.M. Smith, The switchware active network architecture, IEEE Network 12 (3) (1998) 29–36.

[15] K.L. Calvert, S. Bhattacharjee, E. Zegura, J. Sterbenz, Directions in active networks, IEEE Commun. Mag. 36 (10) (1998) 72–78.

[16] A.T. Campbell, H.G. De Meer, M.E. Kounavis, K. Miki, J.B. Vicente, D. Villela, A survey of programmable networks, ACM SIGCOMM CCR 29 (2) (1999) 7.

[17] J.E. van der Merwe, S. Rooney, L. Leslie, S. Crosby, The tempest-a practical framework for network programmability, IEEE Network 12 (3) (1998) 20–28.

[18] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, J. van der Merwe, Design and implementation of a routing control platform, in: USENIX NSDI, 2005, pp. 15–28.

[19] A. Farrel, J.-P. Vasseur, J. Ash, A path computation element (PCE)-based architecture, in: RFC 4655, IETF, 2006.

[20] T. Lakshman, T. Nandagopal, R. Ramjee, K. Sabnani, T. Woo, The softrouter architecture, in: ACM SIGCOMM HotNets, 2004.

[21] J. Van der Merwe, A. Cepleanu, K. D'Souza, B. Freeman, A. Greenberg, D. Knight, R. McMillan, D. Moloney, J. Mulligan, H. Nguyen, et al., Dynamic connectivity management with an intelligent route service control point, in: Proceedings of the SIGCOMM workshop on Internet network management, 2006, pp. 29–34.

[22] M. Casado, T. Garfinkel, A. Akella, M.J. Freedman, D. Boneh, N. McKeown, S. Shenker, Sane: a protection architecture for enterprise networks, in: USENIX Security Symposium, 2006.

[23] M. Casado, M.J. Freedman, J. Pettit, J. Luo, N. McKeown, S. Shenker, Ethane: taking control of the enterprise, in: ACM SIGCOMM CCR, vol. 37, 2007, pp. 1–12.

[24] A. Greenberg, G. Hjalmtysson, D.A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, H. Zhang, A clean slate 4D approach to network control and management, ACM SIGCOMM CCR 35 (5) (2005) 41–54.

[25] C. Mascolo, W. Emmerich, H. De Meer, An XML based programmable network platform, in: ICSE Workshop on Software Engineering and Mobility, 2001.

[26] Devolved Control of atm Networks. <http://www.cl.cam.ac.uk/research/srg/netos/old-projects/dcan/>.

[27] Network Configuration Protocol (netconf). <http://tools.ietf.org/html/rfc624>.

[28] K. Greene, TR10: Software-Defined Networking – MIT Technology Review, 2009. <http://goo.gl/SWpjW3>.

[29] Develop with the junos sdk. <http://goo.gl/jIQ3Wd>.

[30] Cisco Open Network Environment. <http://www.cisco.com/web/solutions/trends/open_network_environment>.

[31] N. Chowdhury, R. Boutaba, A survey of network virtualization, in: Elsevier Computer Networks, 2010.

[32] European Telecommunications Standards Institute, Network Functions Virtualisation, 2012. <http://portal.etsi.org/NFV/NFV_White_Paper.pdf>.

[33] N. Handigol, S. Seetharaman, M. Flajslik, R. Johari, N. McKeown, Aster*x: Load-balancing as a network primitive, in: Proceedings of ACLD, 2010.

[34] S. Agarwal, M. Kodialam, T. Lakshman, Traffic engineering in software defined networks, in: IEEE INFOCOM, 2013, pp. 2211–2219.

[35] S. Shirali-Shahreza, Y. Ganjali, FleXam: flexible sampling extension for monitoring and security applications in OpenFlow, in: ACM SIGCOMM HotSDN, 2013, pp. 167–168.

[36] K. Takagiwa, S. Ishida, H. Nishi, SoR-Based Programmable Network for Future Software-Defined Network, in: IEEE COMPSAC, 2013, pp. 165–166.

[37] Z.A. Qazi, J. Lee, T. Jin, G. Bellala, M. Arndt, G. Noubir, Application-awareness in SDN, in: Proceedings of the ACM SIGCOMM, 2013, pp. 487–488.

[38] P. Sharma, S. Banerjee, S. Tandel, R. Aguiar, R. Amorim, D. Pinheiro, Enhancing network management frameworks with SDN-like control, in: IFIP/IEEE IM, 2013, pp. 688–691.

[39] H. Kim, N. Feamster, Improving network management with software defined networking, IEEE Commun. Mag. 51 (2) (2013) 114–119.

[40] R. Bennesby, P. Fonseca, E. Mota, A. Passito, An inter-as routing component for software-defined networks, in: IEEE NOMS, 2012, pp. 138–145.

[41] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, R. Johari, Plug-n-serve: Load-balancing web traffic using OpenFlow, in: ACM SIGCOMM Demo, 2009.

[42] A. Khurshid, W. Zhou, M. Caesar, P. Godfrey, Veriflow: verifying network-wide invariants in real time, ACM SIGCOMM CCR 42 (4) (2012) 467–472.

[43] P. Kazemian, G. Varghese, N. McKeown, Header space analysis: static checking for networks, in: USENIX NSDI, 2012, pp. 9–9.

[44] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, V. Sekar, Making middleboxes someone else's problem: network processing as a cloud service, ACM SIGCOMM CCR 42 (4) (2012) 13–24.

[45] Z.A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, M. Yu, SIMPLE-fying Middlebox Policy Enforcement Using SDN, in: ACM SIGCOMM, 2013.

[46] S. Fayazbakhsh, V. Sekar, M. Yu, J. Mogul, Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions, ACM SIGCOMM HotSDN.

[47] Y. Nishida, A. Nakao, In-network ad-targeting through wifi ap virtualization, in: 2012 International Symposium on Communications and Information Technologies (ISCIT), IEEE, 2012, pp. 1092–1097.

[48] J. Lee, J. Tourrilhes, P. Sharma, S. Banerjee, No more middlebox: integrate processing into network, ACM SIGCOMM CCR 41 (4) (2011) 459–460.

[49] M. SHIMAMURA, T. IKENAGA, M. TSURU, A design and prototyping of in-network processing platform to enable adaptive network services, IEICE Trans. Inf. Syst. E96-D (2) (2013) 238–248.

[50] J.H. Jafarian, E. Al-Shaer, Q. Duan, OpenFlow random host mutation: transparent moving target defense using software defined networking, in: ACM SIGCOMM HotSDN, 2012, pp. 127–132.

[51] S.A. Mehdi, J. Khalid, S.A. Khayam, Revisiting traffic anomaly detection using software defined networking, in: Recent Advances in Intrusion Detection, Springer, 2011, pp. 161–180.

[52] Resonance. <http://resonance.noise.gatech.edu/>.

[53] M. Suenaga, M. Otani, H. Tanaka, K. Watanabe, Opengate on OpenFlow: system outline, in: 2012 Fourth International Conference on Intelligent Networking and Collaborative Systems, IEEE, 2012, pp. 491–492.

[54] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, M. Tyson, FRESCO: modular composable security services for software-defined networks, in: Proceedings of Network and Distributed Security Symposium, 2013.

[55] S. Ghorbani, M. Caesar, Walk the line: consistent network updates with bandwidth guarantees, in: ACM SIGCOMM HotSDN, 2012, pp. 67–72.

[56] E. Keller, S. Ghorbani, M. Caesar, J. Rexford, Live migration of an entire network (and its hosts), in: ACM SIGCOMM HotNets, 2012, pp. 109–114.

[57] Y. Nakagawa, K. Hyoudou, T. Shimizu, A management method of ip multicast in overlay networks using OpenFlow, in: ACM SIGCOMM HotSDN, 2012, pp. 91–96.

[58] M. Casado, N. McKeown, The virtual network system, in: ACM SIGCSE Bulletin, vol. 37, 2005, pp. 76–80.

[59] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, N. McKeown, ElasticTree: Saving Energy in Data Center Networks, in: USENIX NSDI, vol. 3, 2010, pp. 19–21.

[60] K. Kannan, S. Banerjee, Scissors: Dealing with header redundancies in data centers through SDN, in: IEEE CNSM, 2012, pp. 295–301.

[61] V. Mann, K. Kannan, A. Vishnoi, A.S. Iyer, Ncp: Service replication in data centers through software defined networking, in: IFIP/IEEE IM, 2013, pp. 561–567.

[62] R. Wang, D. Butnariu, J. Rexford, OpenFlow-based server load balancing gone wild, in: USENIX Hot-ICE, 2011, pp. 12–12.

[63] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, B4: Experience with a globally-deployed software defined WAN, in: ACM SIGCOMM, 2013, pp. 3–14.

[64] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, R. Wattenhofer, Achieving high utilization with software-driven WAN, in: ACM SIGCOMM, 2013, pp. 15–26.
[65] Z. Liu, Y. Li, L. Su, D. Jin, L. Zeng, M2cloud: software defined multi-site data center network control framework for multi-tenant, in: ACM SIGCOMM, 2013, pp. 517–518.
[66] Optical Transport Working Group in ONF. <http://goo.gl/YkdUP6>.
[67] S. Gringeri, N. Bitar, T.J. Xia, Extending software defined network principles to include optical transport, IEEE Commun. Mag. 51 (3) (2013) 32–40.
[68] G. Wang, T. Ng, A. Shaikh, Programming your network at run-time for big data applications, in: ACM SIGCOMM HotSDN, 2012, pp. 103–108.
[69] M. Shirazipour, W. John, J. Kempf, H. Green, M. Tatipamula, Realizing packet-optical integration with SDN and OpenFlow 1.1 extensions, in: IEEE ICC, 2012, pp. 6633–6637.
[70] S. Azodolmolky, R. Nejabati, E. Escalona, R. Jayakumar, N. Efstathiou, D. Simeonidou, Integrated OpenFlow-gmpls control plane: an overlay model for software defined packet over optical networks, in: European Conference and Exposition on Optical Communications, Optical Society of America, 2011.
[71] D.E. Simeonidou, R. Nejabati, M. Channegowda, Software defined optical networks technology and infrastructure: enabling software-defined optical network operations, in: Optical Fiber Communication Conference, Optical Society of America, 2013.
[72] J. Zhang, J. Zhang, Y. Zhao, H. Yang, X. Yu, L. Wang, X. Fu, Experimental demonstration of OpenFlow-based control plane for elastic lightpath provisioning in flexi-grid optical networks, Optics express 21 (2) (2013) 1364–1373.
[73] A.N. Patel, P.N. Ji, T. Wang, QoS-aware optical burst switching in OpenFlow based Software-Defined Optical Networks, in: ONDM, 2013, pp. 275–280.
[74] V.R. Gudla, S. Das, A. Shastri, G. Parulkar, N. McKeown, L. Kazovsky, S. Yamashita, Experimental demonstration of OpenFlow control of packet and circuit switches, in: Optical Fiber Communication Conference, Optical Society of America, 2010.
[75] L. Liu, T. Tsuritani, I. Morita, H. Guo, J. Wu, OpenFlow-based wavelength path control in transparent optical networks: a proof-of-concept demonstration, in: IEEE ECOC, 2011, pp. 1–3.
[76] A. Sadasivarao, S. Syed, P. Pan, C. Liou, A. Lake, C. Guok, I. Monga, Open transport switch – a software defined networking architecture for transport networks, in: ACM SIGCOMM HotSDN, 2013.
[77] L. Suresh, J. Schulz-Zander, R. Merz, A. Feldmann, T. Vazao, Towards programmable enterprise wlans with odin, in: ACM SIGCOMM HotSDN, 2012, pp. 115–120.
[78] M. Bansal, J. Mehlman, S. Katti, P. Levis, Openradio: a programmable wireless dataplane, in: ACM SIGCOMM HotSDN, 2012, pp. 109–114.
[79] K.-K. Yap, M. Kobayashi, R. Sherwood, T.-Y. Huang, M. Chan, N. Handigol, N. McKeown, Openroads: empowering research in mobile networks, ACM SIGCOMM CCR 40 (1) (2010) 125–126.
[80] K.-K. Yap, R. Sherwood, M. Kobayashi, T.-Y. Huang, M. Chan, N. Handigol, N. McKeown, G. Parulkar, Blueprint for introducing innovation into wireless mobile networks, in: ACM SIGCOMM VISA, 2010, pp. 25–32.
[81] P. Dely, A. Kassler, N. Bayer, OpenFlow for wireless mesh networks, in: IEEE ICCCN, 2011, pp. 1–6.
[82] A. Gudipati, D. Perry, L.E. Li, S. Katti, SoftRAN: Software Defined Radio Access Network, in: ACM SIGCOMM HotSDN, 2013.
[83] M. Yang, Y. Li, D. Jin, L. Su, S. Ma, L. Zeng, Openran: a software-defined ran architecture via virtualization, in: ACM SIGCOMM, 2013, pp. 549–550.
[84] L.E. Li, Z.M. Mao, J. Rexford, Toward software-defined cellular networks, in: IEEE EWSDN, 2012, pp. 7–12.
[85] C. Guimaraes, D. Corujo, R.L. Aguiar, F. Silva, P. Frosi, Empowering software defined wireless Networks through Media Independent Handover management, in: IEEE GLOBECOM, 2013, pp. 2204–2209.
[86] R. Mortier, T. Rodden, T. Lodge, D. McAuley, C. Rotsos, A.W. Moore, A. Koliousis, J. Sventek, Control and understanding: Owning your home network, in: IEEE COMSNETS, 2012, pp. 1–10.
[87] K.L. Calvert, W.K. Edwards, N. Feamster, R.E. Grinter, Y. Deng, X. Zhou, Instrumenting home networks, ACM SIGCOMM CCR 41 (1) (2011) 84–89.
[88] N. Feamster, Outsourcing home network security, in: ACM SIGCOMM HomeNets, 2010, pp. 37–42.
[89] N. Blefari-Melazzi, A. Detti, G. Mazza, G. Morabito, S. Salsano, L. Veltri, An OpenFlow-based testbed for information centric networking, Future Network Mobile Summit (2012) 4–6.
[90] S. Salsano, N. Blefari-Melazzi, A. Detti, G. Morabito, L. Veltri, Information centric networking over SDN and OpenFlow: Architectural aspects and experiments on the OFELIA testbed, Elsevier Comput. Networks 57 (16) (2013) 3207–3221.
[91] L. Veltri, G. Morabito, S. Salsano, N. Blefari-Melazzi, A. Detti, Supporting information-centric functionality in software defined networks, in: IEEE ICC, 2012, pp. 6645–6650.
[92] D. Syrivelis, G. Parisis, D. Trossen, P. Flegkas, V. Sourlas, T. Korakis, L. Tassiulas, Pursuing a software defined information-centric network, in: IEEE EWSDN, 2012, pp. 103–108.
[93] X.N. Nguyen, D. Saucez, T. Turletti, Efficient caching in content-centric networks using OpenFlow, in: INFOCOM Student Workshop, 2013.
[94] J. Suh, H. Jung, T. TaekyoungKwon, Y. Choi, C-flow: Content-oriented networking over OpenFlow, in: Open Networking Summit, 2012.
[95] T. Tsou, X. Shi, J. Huang, Z. Wang, X. Yin, Analysis of Comparisons between OpenFlow and Forces. <http://tools.ietf.org/html/draft-wang-forces-compare-openflow-forces-00/>.
[96] OpenFlow Switch Specification. <http://goo.gl/1DYxw6>.
[97] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, S. Shenker, NOX: towards an operating system for networks, ACM SIGCOMM CCR 38 (3) (2008) 105–110.
[98] POX: A Python-Based OpenFlow Controller, author = Mccauley, J. <http://www.noxrepo.org/pox/about-pox/>.
[99] SNAC. <http://www.openflowhub.org/display/Snac/>.
[100] E. Ng, Maestro: A System for Scalable OpenFlow Control. <https://code.google.com/p/maestro-platform/>.
[101] Ryu, An Operating System for Software Defined Network. <http://osrg.github.com/ryu/>.
[102] Mul. <http://sourceforge.net/p/mul/wiki/Home/>.
[103] Beacon. <https://openflow.stanford.edu/display/Beacon/Home/>.
[104] Floodlight. <http://floodlight.openflowhub.org/>.
[105] IRIS. <http://openiris.etri.re.kr/>.
[106] OESS. <https://code.google.com/p/nddi/>.
[107] Jaxon. <http://jaxon.onuos.org/>.
[108] Nodeflow. <http://garyberger.net/?p=537/>.
[109] ovs-controller. <http://openvswitch.org/cgi-bin/ovsman.cgi?page=utilities%2Fovs-controller.8/>.
[110] Open vSiwtch. <http://openvswitch.org/>.
[111] Flowvisor. <https://github.com/OPENNETWORKINGLAB/flowvisor/wiki/>.
[112] Routeflow. <https://sites.google.com/site/routeflow/>.
[113] Helios by NEC. <http://www.nec.com/>.
[114] H. Pan, H. Guan, J. Liu, W. Ding, C. Lin, G. Xie, The flowadapter: enable flexible multi-table processing on legacy hardware, in: ACM SIGCOMM HotSDN, 2013, pp. 85–90.
[115] N. Kang, J. Reich, J. Rexford, D. Walker, Policy transformation in software defined networks, in: Proceedings of the ACM SIGCOMM, 2012, pp. 309–310.
[116] Y. Kanizo, D. Hay, I. Keslassy, Palette: Distributing tables in software-defined networks, in: IEEE INFOCOM Mini-conference, 2013.
[117] N. Kang, Z. Liu, J. Rexford, D. Walker, Optimizing the one big switch abstraction in software-defined networks, in: ACM CoNEXT, 2013.
[118] Network OS: ONOS. <http://onlab.us/tools.html#os/>.
[119] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, Onix: A distributed control platform for large-scale production networks, in: Proceedings of USENIX OSDI, vol. 10, 2010, pp. 1–6.
[120] L. Vanbever, J. Reich, T. Benson, N. Foster, J. Rexford, Hotswap: Correct and efficient controller upgrades for software-defined networks, in: ACM SIGCOMM HotSDN, 2013.
[121] A. Tootoonchian, Y. Ganjali, Hyperflow: A distributed control plane for OpenFlow, in: USENIX INM/WREN, 2010, pp. 3–3.
[122] P. Peresini, M. Kuzniar, N. Vasic, M. Canini, D. Kostic, Of. cpp: Consistent packet processing for OpenFlow, Tech. rep., EPFL, 2013.
[123] D. Levin, A. Wundsam, B. Heller, N. Handigol, A. Feldmann, Logically centralized?: state distribution trade-offs in software defined networks, in: ACM SIGCOMM HotSDN, 2012, pp. 1–6.
[124] N.P. Katta, J. Rexford, D. Walker, Incremental consistent updates, in: ACM SIGCOMM HotSDN, 2013, pp. 49–54.
[125] N.P. Katta, J. Rexford, D. Walker, Time-Based Updates in Software Defined Networks, in: ACM SIGCOMM HotSDN, 2013.
[126] M. Yu, J. Rexford, M.J. Freedman, J. Wang, Scalable flow-based networking with DIFANE, ACM SIGCOMM CCR 41 (4) (2011) 351–362.
[127] A.R. Curtis, J.C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, S. Banerjee, DevoFlow: Scaling flow management for high-

performance networks, in: ACM SIGCOMM CCR, vol. 41, 2011, pp. 254–265.
[128] S. Hassas Yeganeh, Y. Ganjali, Kandoo: a framework for efficient and scalable offloading of control applications, in: ACM SIGCOMM HotSDN, 2012, pp. 19–24.
[129] M. Othman, K. Okamura, Hybrid control model for flow-based networks, in: IEEE COMPSAC, 2013, pp. 765–770.
[130] D. Drutskoy, E. Keller, J. Rexford, Scalable network virtualization in software-defined networks, IEEE Internet Comput. 17 (2) (2013) 20–27.
[131] Z. Bozakov, P. Papadimitriou, Autoslice: automated and scalable slicing for software-defined networks, in: ACM CoNEXT student workshop, 2012, pp. 3–4.
[132] S.H. Yeganeh, A. Tootoonchian, Y. Ganjali, On scalability of software-defined networking, IEEE Commun. Mag. 51 (2) (2013) 136–141.
[133] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, R. Kompella, Towards an elastic distributed sdn controller, in: ACM SIGCOMM HotSDN, 2013, pp. 7–12.
[134] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, M. Horowitz, Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN, in: Proceedings of the ACM SIGCOMM, 2013, pp. 99–110.
[135] J.C. Mogul, P. Congdon, Hey, you darned counters!: get off my ASIC!, in: ACM SIGCOMM HotSDN, 2012, pp. 25–30.
[136] G. Lu, R. Miao, Y. Xiong, C. Guo, Using cpu as a traffic co-processing unit in commodity switches, in: ACM SIGCOMM HotSDN, 2012, pp. 31–36.
[137] K. Kogan, S. Nikolenko, W. Culhane, P. Eugster, E. Ruan, Towards efficient implementation of packet classifiers in SDN/OpenFlow, in: ACM SIGCOMM HotSDN, 2013, pp. 153–154.
[138] E. Kissel, G. Fernandes, M. Jaffee, M. Swany, M. Zhang, Driving Software Defined Networks with XSP, in: IEEE ICC, 2012, pp. 6616–6621.
[139] C. Monsanto, J. Reich, N. Foster, J. Rexford, D. Walker, Composing software defined networks, in: Proceedings of USENIX NDSI.
[140] X. Wen, Y. Chen, C. Hu, C. Shi, Y. Wang, Towards a secure controller platform for OpenFlow applications, in: ACM SIGCOMM HotSDN, 2013, pp. 171–172.
[141] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, G. Gu, A security enforcement kernel for OpenFlow networks, in: ACM SIGCOMM HotSDN, 2012, pp. 121–126.
[142] S. Shin, G. Gu, Attacking software-defined networks: a first feasibility study, in: ACM SIGCOMM HotSDN, 2013, pp. 165–166.
[143] K. Benton, L.J. Camp, C. Small, OpenFlow vulnerability assessment, in: ACM SIGCOMM HotSDN, 2013, pp. 151–152.
[144] M. Kuzniar, P. Perešini, N. Vasic, M. Canini, D. Kostic, Automatic failure recovery for software-defined networks, in: ACM SIGCOMM HotSDN, 2013.
[145] U.C. Kozat, G. Liang, K. Kokten, Verifying forwarding plane con tivity and locating link failures using static rules in software defined networks, in: ACM SIGCOMM HotSDN, 2013, pp. 157–158.
[146] J. Kempf, E. Bellagamba, A. Kern, D. Jocha, A. Takacs, P. Skoldstrom, Scalable fault management for OpenFlow, in: IEEE ICC, 2012, pp. 6606–6610.
[147] D. Williams, H. Jamjoom, Cementing high availability in OpenFlow with rulebricks, in: ACM SIGCOMM HotSDN, 2013, pp. 139–144.
[148] B. Heller, R. Sherwood, N. McKeown, The controller placement problem, in: ACM SIGCOMM HotSDN, 2012, pp. 7–12.
[149] K. Phemius, M.B. Thales, OpenFlow: Why latency does matter, in: IFIP/IEEE IM, 2013, pp. 680–683.
[150] Y. Hu, W. Wendong, X. Gong, X. Que, C. Shiduan, Reliability-aware controller placement for software-defined networks, in: IFIP/IEEE IM, 2013, pp. 672–675.
[151] Mininet. <http://mininet.org/>.
[152] ns-3. <http://www.nsnam.org/>.
[153] D. Klein, An OpenFlow extension for the omnet++ inet framework.
[154] Estinet 8.0 OpenFlow Network Simulator and Emulator. <http://www.estinet.com/products.php/>.
[155] S. Wang, C. Chou, C.-M. Yang, Estinet OpenFlow network simulator and emulator, IEEE Commun. Mag. 51 (9) (2013) 110–117.
[156] Trema. <http://trema.github.io/trema/>.
[157] Mirage. <http://www.openmirage.org/>.
[158] Cbench. <http://www.openflowhub.org/display/floodlightcontroller/Cbench+(New)/>.
[159] NICE. <https://code.google.com/p/nice-of/>.
[160] M. Canini, D. Venzano, P. Peresini, D. Kostic, J. Rexford, A nice way to test OpenFlow applications, in: USENIX NSDI, 2012, pp. 10–10.
[161] SDN Troubleshooting Simulator (STS). <http://ucb-sts.github.io/sts/>.
[162] Oflops. <http://archive.openflow.org/wk/index.php/Oflops/>.
[163] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, A.W. Moore, Oflops: An open framework for OpenFlow switch evaluation, in: Passive and Active Measurement, Springer, 2012, pp. 85–95.
[164] OFTest. <http://archive.openflow.org/wk/index.php/OFTestTutorial/>.
[165] OFRewind. <http://archive.openflow.org/wk/index.php/OFRewind/>.
[166] A. Wundsam, D. Levin, S. Seetharaman, A. Feldmann, OFRewind: Enabling Record and Replay. Troubleshooting for Networks, in: USENIX ATC, 2011.
[167] N. Handigol, B. Heller, V. Jeyakumar, D. Mazieres, N. McKeown, Where is the debugger for my software-defined network? in: ACM SIGCOMM HotSDN, 2012, pp. 55–60.
[168] R.C. Scott, A. Wundsam, K. Zarifis, S. Shenker, What, Where, and When: Software Fault Localization for SDN, Tech. rep., UCB/EECS-2012-178, UC Berkeley, 2012.
[169] H. Zeng, P. Kazemian, G. Varghese, N. McKeown, Automatic test packet generation, in: Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies, 2012, pp. 241–252.
[170] A. Voellmy, J. Wang, Y.R. Yang, B. Ford, P. Hudak, Maple: Simplifying SDN programming using algorithmic policies, in: ACM SIGCOMM, 2013, pp. 87–98.
[171] M. Reitblatt, M. Canini, A. Guha, N. Foster, Fattire: Declarative Fault Tolerance for Software-Defined Networks.
[172] T.L. Hinrichs, N.S. Gude, M. Casado, J.C. Mitchell, S. Shenker, Practical declarative network management, in: ACM SIGCOMM WREN, pp. 1–10.
[173] A. Voellmy, H. Kim, N. Feamster, Procera: a language for high-level reactive network control, in: ACM SIGCOMM HotSDN, 2012, pp. 43–48.
[174] N. Foster, R. Harrison, M.J. Freedman, C. Monsanto, J. Rexford, A. Story, D. Walker, Frenetic: a network programming language, ACM SIGPLAN Notices 46 (9) (2011) 279–291.
[175] M. Reitblatt, N. Foster, J. Rexford, D. Walker, Consistent updates for software-defined networks: change you can believe in!, in: ACM SIGCOMM HotNets, 2011, p. 7.
[176] PlanetLab Europe. <http://www.planet-lab.eu/openflow/>.
[177] OFELIA, OpenFlow in Europe: Linking Infrastructure and Applications. <http://www.fp7-ofelia.eu/about-ofelia/>.
[178] National LambdaRail. <http://www.nlr.net/testbeds.php/>.
[179] Internet2-based NDDI. <http://inddi.wikispaces.com/Internet2-based+NDDI/>.
[180] RISE testbed. <http://www.jgn.nict.go.jp/english/info/technologies/openflow.html/>.
[181] SURFnet. <http://goo.gl/QDPm7W>.
[182] COTN, The California OpenFlow Testbed Network. <http://www.cenic.org/page_id=143/>.
[183] The FIBER Project. <http://www.fibre-ict.eu>.
[184] M.F. Schwarz, M.A. Rojas, C.C. Miers, F.F. Redigolo, T.C. Carvalho, Emulated and software defined networking convergence, in: IFIP/IEEE IM, 2013, pp. 700–703.
[185] Open Networking Foundation. <https://www.opennetworking.org/index.php?lang=en/>.
[186] Open Networking Research Center. <https://onrc.stanford.edu>.
[187] OpenDayLight. <http://www.opendaylight.org/>.
[188] OpenStack. <http://www.openstack.org/>.
[189] Apache CloudStack. <http://cloudstack.apache.org>.
[190] Interface to the Routing System (I2RS). <http://datatracker.ietf.org/wg/i2rs>.
[191] Software-Defined Networking Research Group. <http://irtf.org/sdnrg/>.
[192] ITU-T. <http://www.itu.int/en/ITU-T/Pages/default.aspx/>.
[193] Metro Ethernet Forum (MEF). <http://metroethernetforum.org>.
[194] OpenFlow Reference Implementation. <http://archive.openflow.org/wk/index.php/Source_Code_Exploration>.
[195] Dpqd ofsoftswitch13. <https://github.com/CPqD/ofsoftswitch13/>.
[196] Indigo. <http://www.projectfloodlight.org/indigo/>.
[197] OpenFaucet. <http://rlenglet.github.io/openfaucet/index.html/>.
[198] Pantou. <http://archive.openflow.org/wk/index.php/Pantou_:_OpenFlow_1.0_for_OpenWRT>.
[199] Y. Luo, P. Cascon, E. Murray, J. Ortega, Accelerating OpenFlow switching with network processors, in: ACM/IEEE ANCS, 2009, pp. 70–71.
[200] V. Tanyingyong, M. Hidell, P. Sjödin, Improving PC-based OpenFlow switching performance, in: ACM/IEEE ANCS, 2010, p. 13.

[201] A. Bianco, R. Birke, L. Giraudo, M. Palacin, OpenFlow switching: Data plane performance, in: IEEE ICC, 2010, pp. 1–5.
[202] S. Han, K. Jang, K. Park, S. Moon, PacketShader: a GPU-accelerated software router, ACM SIGCOMM CCR 40 (4) (2010) 195–206.
[203] H. Gill, D. Lin, L. Sarna, R. Mead, K.C.T. Lee, B.T. Loo, SP4: scalable programmable packet processing platform, in: Proceedings of the ACM SIGCOMM, 2012, pp. 75–76.
[204] Akihiro Nakao, FLARE: Open Deeply Programmable Switch, in: The 16th GENI Engineering Conference, 2012. <http://www.pilab.jp/ipop2013/exhibition/panel/iPOP2013_uTokyo_panel.pdf>.
[205] M. Dobrescu, K. Argyraki, G. Iannaccone, M. Manesh, S. Ratnasamy, Controlling parallelism in a multicore software router, in: ACM PRESTO, 2010.
[206] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, F. Huici, L. Mathy, Towards high performance virtual routers on commodity hardware, in: Proceedings of ACM CoNEXT, 2008, p. 20.
[207] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, S. Ratnasamy, RouteBricks: exploiting parallelism to scale software routers, in: Proceedings of the ACM SIGOPS SOSP, 2009, pp. 15–28.
[208] Y. Qi, J. Fong, W. Jiang, B. Xu, J. Li, V. Prasanna, Multi-dimensional packet classification on FPGA: 100 Gbps and beyond, in: IEEE ICFPT, 2010, pp. 241–248.
[209] M. Ahmed, F. Huici, A. Jahanpanah, Enabling dynamic network processing with clickos, in: Proceedings of the ACM SIGCOMM, 2012, pp. 293–294.
[210] N. Kim, J.-Y. Yoo, N.L. Kim, J. Kim, A programmable networking switch node with in-network processing support, in: IEEE ICC, 2012, pp. 6611–6615.
[211] M. Dobrescu, K. Argyraki, S. Ratnasamy, Toward predictable performance in software packet-processing platforms, in: USENIX NSDI, 2012, pp. 11–11.
[212] K. Jang, S. Han, S. Han, S. Moon, K. Park, SSLShader: cheap SSL acceleration with commodity processors, in: USENIX NSDI, 2011, pp. 1–1.
[213] H. Song, Protocol-oblivious forwarding: unleash the power of SDN through a future-proof forwarding plane, in: ACM SIGCOMM HotSDN, 2013, pp. 127–132.
[214] H. Farhadi, P. Du, A. Nakao, User-defined actions for SDN, in: ACM CFI, 2014.
[215] H. Farhady, A. Nakao, Tagflow: efficient flow classification in sdn, IEICE Trans. Commun. 97 (11) (2014) 2302–2310.
[216] L. Liang, H. Farhady, D. Ping, A. Nakao, Ouroboros: protocol independent forwarding for sdn, IEICE Trans. Commun. 97 (11) (2014) 2278–2285.
[217] A.D. Ferguson, A. Guha, J. Place, R. Fonseca, S. Krishnamurthi, Participatory networking, in: USENIX Hot-ICE, vol. 12, 2012.
[218] A. Ferguson, A. Guha, C. Liang, R. Fonseca, S. Krishnamurthi, Participatory Networking: An API for Application Control of SDNs, in: ACM SIGCOMM, 2013.
[219] P. Lin, J. Hart, U. Krishnaswamy, T. Murakami, M. Kobayashi, A. Al-Shabibi, K.-C. Wang, J. Bi, Seamless interworking of SDN and IP, in: ACM SIGCOMM, 2013, pp. 475–476.
[220] D. Levin, M. Canini, S. Schmid, A. Feldmann, Incremental SDN Deployment in Enterprise Networks, in: ACM SIGCOMM, 2013.

Hamid Farhadi received the M.S. degree in computer science from Sharif University of Technology, Iran, in 2010. He is currently pursuing his Ph.D. in the University of Tokyo, Japan. His research interests include software-defined networking, network virtualization and cloud computing.

HyunYong Lee received the M.S. degree and the Ph.D. degree in computer science from the Gwangju Institute of Science and Technology (GIST), Korea, in 2003 and 2010, respectively. He is currently Research associate of the University of Tokyo, Japan. His research interests include P2P traffic control and in-network caching and user incentive mechanism of the information-centric networking.

Akihiro Nakao received B.S. (1991) in Physics, M.E. (1994) in Information Engineering from the University of Tokyo. He was at IBM Yamato Laboratory/at Tokyo Research Laboratory/at IBM Texas Austin from 1994 till 2005. He received M.S. (2001) and Ph.D. (2005) in Computer Science from Princeton University. He has been teaching as an Associate Professor in Applied Computer Science, at Interfaculty Initiative in Information Studies, Graduate School of Interdisciplinary Information Studies, the University of Tokyo since 2005. (He has also been an expert visiting scholar/a project leader at National Institute of Information and Communications Technology (NICT) since 2007).