European Journal of Computer Science and Information Technology
Vol.10, No.2, pp.27-48, 2022
Print ISSN: 2054-0957 (Print),
Online ISSN: 2054-0965 (Online)

Application of Software-Defined Networking

Sunday, Uzoma I. & Akhibi, Samuel D.
Paul löbe Straße 25, Ilmenau, Germany

ABSTRACT: *Software-Defined Networking (SDN) is an architecture purporting to be dynamic, manageable cost-effective, and adaptable, seeking to be suitable for the high bandwidth, dynamic nature of today's applications. SDN architectures decouple network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstract from application and network services. In this paper, the advantages and challenges of Software-Defined Networking (SDN) are discussed.However, the Software Defined Networking application (SDN App) and the application of SDN are also discussed in detail.*

KEYWORDS: networking, scalability, software, controller, packet

INTRODUCTION

A wise variety of applications with diverse requirements on the network services has been enabled by rapid advancement in networking and computing technologies [1]The highly diverse and dynamic network services demanded by current and emerging applications bring in new challenges to service provisioning in future networks. Software-defined networking (SDN) and network functions virtualization (NFV) are two significant recent innovations that are expected to address these challenges [1].
Software-defined networking (SDN) has gained a lot of attention in recent years, because it addresses the lack of programmability in existing networking architectures and enables easier and faster network innovation. SDN clearly separates the data plane from the control plane and facilitates software implementations of complex networking applications on top [2].

SDN separates network control and data forwarding functionalities to enable centralized and programmable network control [1]. Key components of the SDN architecture include a data plane consisting of network resources for data forwarding, a control plane comprising SDN controller(s) providing centralized control of network resources, and control/management applications that program network operations through a controller. The control-resource interface between the control and data planes is called the southbound interface, while the control-application interface is called the northbound inter-face. Advantages promised by SDN include simplified and enhanced network control, flexible and efficient network management, and improved network service performance.

Network virtualization introduces an abstraction of the underlying infrastructure upon which virtual networks with alternative architecture may be constructed to meet diverse service requirements [3]. More recently, the European Tele-Communications Standards Institute (ETSI) developed NFV.The Software-Defined Networking is a new networking paradigm that tries to improve flexibility, and programmability and reduces management complexity by separating the control plane from the data plane [4], [5]. By having a centralized control plane, it is possible to have a global view of the network providing the opportunity for simplified control applications. However, as with any centralized architecture, scalability can become a major performance degrading factor as the network grows in size. Therefore, understanding the scalability issues is extremely important in the design of carrier-grade SDNs and their widespread adoption. Towards this end, some studies, predominantly experimental in nature, have been carried out. Yeganeh et al. [3] points out several scalability concerns and argue that they are not specific to SDNs. In [4], the authors evaluated the first SDN controller, NOX, and reported that it can only serve 30,000 flow initiation requests while maintaining the flow setup time of less than 10 msec. The scalability of different control plane architectures (centralized, de-centralized and hierarchical) is compared using simple queuing models in [5]. Solutions ranging from a distributed controller architecture [6] to moving some control functions back to the switches [7] are proposed to alleviate the scalability issues.

In the context of SDNs, scalability has always been evaluated based on the number of flow initiation requests that can be supported while guaranteeing a Quality of Service (QoS) factor such as the flow setup time1. While the above metric for scalability is important, it is more appropriate for static networks and does not capture issues that surface when the network topology changes frequently.

Characterizing the throughput loss due to topology changes is hard without detailed information about the topology. However, we can use, as a first approximation, the level of inconsistency between the controller's view of the topology and the actual network as a proxy for the throughput degradation [10]. Note that having a link in the actual network, but not in the controller's view of the topology, is not an issue because the flow table will not have that additional link [11]. However, the inconsistency caused by a link failure could lead to throughput degradation if that link is present in the path of an already established flow. It is straight-forward to compute the expected flow setup time when the size of a network in terms of the number of switches and links is known. In this paper, we define scalability as the number of switches that can be controlled while maintaining the inconsistency below a certain number of links and maintaining the expected flow setup time below a certain threshold when the links are characterized by the distributions of their lifetimes and down-times [10].

REVIEW OF RELEVANT LITERATURE

The past few years have witnessed exciting progress in SDN technologies and their applications in various networking scenarios [12], including wireless networks [14]. On the other hand, researchers have noticed some issues of the current SDN approach that may limit its ability to fully support future network services [15, 16]. To meet the evolving diverse service requirements, SDN data plane devices need to fully perform general flow matching and packet forwarding, which may significantly increase complexity and cost of SDN switches. On the control plane, current SDN architecture lacks sufficient support of interoperability among heterogeneous SDN controllers, and thus limits its ability to provision flexible end-to-end services across autonomous domains.

A root reason for the limitation of current SDN design to achieve its full potential for service provisioning is the tight coupling between network architecture and infrastructure on both data and control planes. Separation between data and control planes alone in the current SDN architecture is not sufficient to overcome this obstacle. Another dimension of abstraction to decouple service functions and network infra-structures is needed in order to unlock SDN's full potential. Therefore, applying the insights of NFV in SDN may further enhance the latter's capability of flexible service provisioning.

On the other hand, many technical challenges must be addressed for realizing the NFV paradigm. Management and orchestration have been identified as key components in the ETSI NFV architecture. Much more sophisticated control and management mechanisms for both virtual and physical resources are required by the highly dynamic networking environment enabled by NFV, in which programmatic network control is indispensable. Employing the SDN principle — decoupling control intelligence from the controlled resources to enable a logically centralized programmable control/management plane — in the NFV architecture may greatly facilitate realization of NFV [17].

Recent research efforts toward combining SDN and NFV to enhance network service provisioning have been made from various aspects. Hypervisor and container-based virtualization mechanisms have been applied to support multi-tenant virtual SDN networks. For example, the network hypervisor Flow-Visor [18] allows multiple controllers to share an Open Flow platform and slice data plane infrastructure. FlowN [18] offers container-based virtualization solution in which each tenant may run its own control application on a shared SDN controller. Some network system designs have explored utilizing capabilities of both SDN and NFV. For example, Woods et al. [19] presented NetVM, a high-performance virtual server platform for supporting NFV, and discussed design guidelines for combining SDN controllers with NetVM to provide coordinated network management. Ding et al. [10] designed an open platform for service chain as a service by using capabilities of SDN together with NFV. The progressive evolution from SDN-agnostic NFV initiative to SDN-enabled NFV solution was discussed in [11]. Relevant standardization organizations are also actively conducting the related study. The Open Network Foundation (ONF) recently released a report on the relationship of SDN and NFV [20], and ETSI NFV ISG is currently

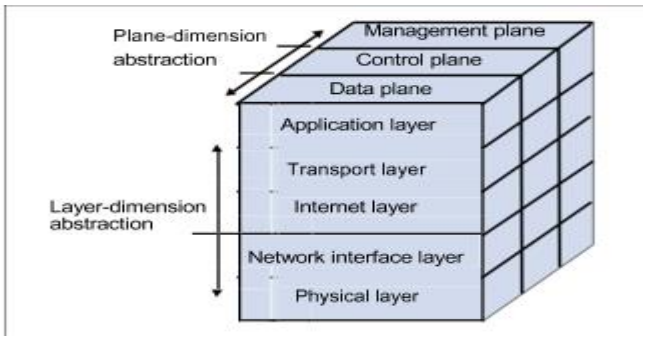

Figure 1.2 A two-dimensional model of layer-plane abstraction in future networking.

Recent research efforts toward combining SDN and NFV to enhance network service provisioning have been made from various aspects. Hypervisor and container-based virtualization mechanisms have been applied to support multi-tenant virtual SDN networks. For example, the network hypervisor Flow-Visor [17] allows multiple controllers to share an OpenFlow platform and slice data plane infrastructure. FlowN [18] offers a container-based virtualization solution in which each tenant may run its own control application on a shared SDN controller. Some network system designs have explored utilizing capabilities of both SDN and NFV. For example, Woods et al. [19] presented NetVM, a high-performance virtual server platform for supporting NFV, and discussed design guidelines for combining SDN controllers with NetVM to provide coordinated network management. Ding et al. [20] designed an open platform for service chain as a service by using capabilities of SDN together with NFV. The progressive evolution from SDN-agnostic NFV initiative to SDN-enabled NFV solution was discussed in [21]. Relevant standardization organizations are also actively conducting the related study. The Open

Network Foundation (ONF) recently released a report on the relationship of SDN and NFV [12], and ETSI NFV ISG is currently working on a draft report regarding SDN usage in the NFV architecture [23]. Although encouraging progress has been made toward combining SDN and NFV, research in this area is still in its infant stage. Current works address the problem from various aspects, including hypervisors for virtual SDN networks, usage of SDN controllers in NFV architecture, and SDN/NFV hybrid solutions for service provisioning. It is desirable to have a high-level framework that provides a holistic vision of how SDN and NFV principles may naturally fit into unified network architecture, which may greatly facilitate the research and technical development in this area. This motivates the work presented in the rest of this article.

A Two-Dimensional Abstraction Model for SDN and NFV Integration In this section, we present a two-dimensional abstraction model to show how SDN and NFV principles are related to each other and how they may fit in unified network architecture.

As shown in Fig. 1.2, this abstraction model has layers as well as planes with clear distinction between these two concepts. Both layers and planes offer abstraction in network architecture but in different dimensions. Abstraction provided by layers is in the vertical dimension in the model, starting with underlying hardware and then adding a sequence of layers, each providing a higher (more abstract) level of service. A key property of layering is that the functions of a higher layer rely on the services provided by the lower layers, therefore forming a stack of layers for offering services to applications on the top. On the other hand, plane abstraction is in the horizontal dimension in that functions performed on a plane do not necessarily rely on functions of another plane; therefore, there is no higher or lower plane. Instead, each plane focuses on a particular aspect of the entire network system, such as data transport, network control, and system management. Each plane may comprise multiple layers from physical hardware to application software and collaborates with other planes for network service provisioning.
Traditional circuit-switching-based telecommunication systems embraced plane-dimension abstraction (separating data, control, and management planes) without clear abstraction on the layer dimension. For example, Signal System No. 7 was logically separated from voice channels, and the intelligent network (IN) had service control points (SCPs) decoupled from the data transportation platform [24].

PROTOCOL OPTIONS FOR THE SOUTHBOUND INTERFACE

The most common southbound interface is Open Flow, which is standardized by the Open Networking Foundation (ONF). Open Flow is a protocol that describes the interaction of one or more control servers with Open Flow-compliant switches. An Open Flow controller installs flow table entries in switches, so that these switches can forward traffic according to these entries. Thus, Open Flow switches depend on configuration by controllers. A flow is classified by match fields that are similar to access control lists (ACLs) and may contain wildcards. In Section 3, we provide a detailed description of Open Flow and describe the features offered by different versions of the protocol.

Another option for the southbound interface is the Forwarding and Control Element Separation (ForCES) [5,6] which is discussed and has been standardized by the Internet Engineering Task Force (IETF) since 2004. ForCES is also a framework, not only a protocol; the ForCES framework also separates the control plane and data plane, but is considered more flexible and more powerful than OpenFlow [7,8]. Forwarding devices are modeled using logical function blocks (LFB) that can be composed in a modular way to form complex forwarding mechanisms. Each LFB provides a given functionality, such as IP routing. The LFBs model a forwarding device and cooperate to form even more complex network devices. Control elements use the ForCES protocol to configure the interconnected LFBs to modify the behavior of the forwarding elements.

The Soft Router architecture [9] also defines separate control and data plane functionality. It allows dynamic bindings between control elements and data plane elements, which allows a dynamic assignment of control and data plane elements. In [9], the authors present the SoftRouter architecture and highlight its advantages on the Border Gateway Protocol (BGP) with regard to reliability.

ForCES and Soft Router have similarities with OpenFlow and can fulfill the role of the southbound interface. Other networking technologies are also discussed, as well as possible southbound interfaces in the IETF. For instance, the Path Computation Element (PCE) [10] and the Locator/ID Separation Protocol (LISP) [11] are candidates for southbound interfaces.

NORTHBOUND APIS FOR NETWORK APPLICATIONS

As we will highlight in Section 3, the OpenFlow protocol provides an interface that allows a control software to program switches in the network. Basically, the controller can change the forwarding behavior of a switch by altering the forwarding table. Controllers often provide a similar interface to applications, which is called the northbound interface, to expose the programmability of the network. The northbound interface is not standardized and often allows fine-grained control of the switches. Applications should not have to deal with the details of the southbound interface, e.g., the application does not need to know all details about the network topology, etc. For instance, a traffic engineering network applications should tell the controller the path layout of the flows, but the controller should create appropriate commands to modify the forwarding tables of the switches. Thus, network programming languages are needed to ease and automate the configuration and management of the network.

The requirements of a language for SDN are discussed in [12]. The authors focus on three important aspects. (1) The network programming language has to provide the means for querying the network state. The language runtime environment gathers the network state and statistics, which is then provided to the application; (2) The language must be able to express network policies that define the packet forwarding behavior. It should be possible to combine policies of different network applications. Network applications possibly construct conflicting network policies, and the network language should be powerful enough to express and to resolve such conflicts; (3) The reconfiguration of the network is a difficult task, especially with various network policies. The runtime environment has to trigger the update process of the devices to guarantee that access control is preserved, forwarding loops are avoided or other invariants are met.

Popular SDN programming languages that fulfill the presented requirements are Frenetic [13], its successor, Pyretic [14], and Procera [15]. These languages provide a declarative syntax and are based on functional reactive programming. Due to the nature of functional reactive programming, these languages provide a composable interface for network policies. Various other proposals exist, as well. The European FP7 research project, NetIDE, addresses the northbound interfaces of SDN networks[16].

SDN, ACTIVE AND PROGRAMMABLE NETWORKS

In the past, various technologies were developed to enable programmability in communication networks. Active networks (AN) [17] were developed in the 1990s. The basic idea of active networks is to inject programs into data packets. Switches extract and execute programs from data packets. With this method, new routing mechanisms and network services can be implemented without the modification of the forwarding hardware. However, this approach has not prevailed, due to security concerns and performance issues that can occur on executing programs in the forwarding devices. For example, an attacker can inject malicious programs into network packets and forward them into the network.

Programmable networks (PN) [18,19] also provide a means for programmability in the network by executing programs on packets similar to AN. However, programs are not included in the data packets as with AN. The programs are installed inside the network nodes, which execute the programs on the packets. This clearly reduced security concerns that occur with AN, because a network node only accepts programs after a prior signaling and node setup. Various proposals for PN were made in the past. For example, xbind [20] was proposed for asynchronous transfer mode (ATM) networks that are tailored towards quality of service (QoS) and multimedia applications. The aim of xbind was to overcome the complexity associated with ATM and to allow easier service creation by separating the control algorithms from the ATM protocol, which was enabled by the programmability of the forwarding devices through programming languages.

Both approaches, AN and PN, introduce new flexibility by allowing programs to be executed within the network. They are more flexible than an OpenFlow-based SDN approach, because they allow one to program the data plane in an extensible way. New data plane functionality can be implemented through programs that are either part of data packets with AN or installed inside network nodes with PN. An OpenFlow-based SDN approach cannot extend the data plane functionality without an upgrade of the switches, due to the fact that OpenFlow only provides a fixed set of network operations. The OpenFlow controller is only able to program the switch with its supported set of operations.

SYSTEM DESCRIPTION

In traditional IP networks, the packet routers are in charge of determining the routing policies and other control functions as well as forwarding the packets. That is, the control plane and data plane are tightly coupled in traditional networking architectures. The key idea behind SDNs is to separate the control plane, managed by a controller, from the data plane consisting of a set of forwarding elements or switches. The basic SDN architecture that separates control plane from data plane is shown in Fig. 1.6. In the figure, the services provided by the network operator, such as security, are termed as the application, and they communicate with the SDN controller using interfaces called northbound APIs [80] (for e.g., Procera and RESTFul [70]). The SDN controller controls and programs the switches to perform different functionality using APIs that are termed as southbound APIs (the most common interfaces are OpenFlow [9], ForCES, and NetConf [2]).

We start by describing the flow setup process in SDNs. A more detailed description can be found in [10]. As a part of the start-up procedure, an SDN switch will communicate its IP address and all its available switch ports to the controller. When a packet from a new flow arrives, the switch first checks if there is an entry in its flow table corresponding to the packet it received. If there is no forwarding rule available in the flow table for the received packet, then the packet is forwarded to the controller (for example, in OpenFlow the packet is encapsulated in a Packet_In message). The controller, upon receiving the message from the switch, computes or fetches the pre-computed forwarding rules and installs them in all the concerned switches along the path of the flow. Once the forwarding rules are installed in the switches, the switches can forward packets without any intervention from the controller. The controller should have an up-to-date view of the network in order to compute and install appropriate control functions in the switches. If the controller's view of the network is not consistent with the actual topology, the forwarding rules installed by the controller could be incorrect and could lead to packet drops causing significant performance degradation. Therefore, the controller performs the topology discovery process periodically to update its view of the network.

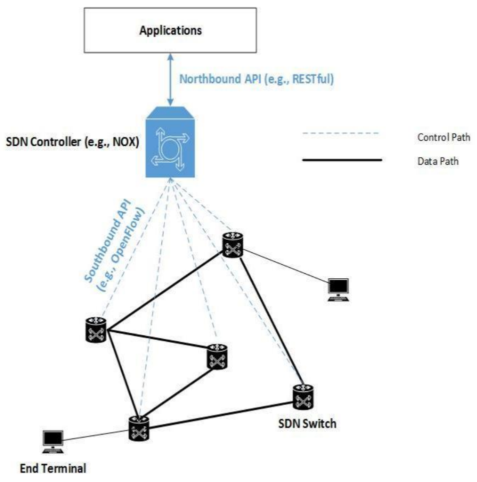
Applications
Northbound API (e.g., RESTful)
SDN Controller (e.g., NOX)
Control Path
Data Path
Southbound API (e.g., OpenFlow)
SDN Switch
End Terminal

Fig. 1.6. Architecture of a software defined network.

THE OPENFLOW PROTOCOL

The OpenFlow protocol is the most commonly used protocol for the southbound interface of SDN, which separates the data plane from the control plane. The white paper about OpenFlow [71] points out the advantages of a flexibly configurable forwarding plane. OpenFlow was initially proposed by Stanford University, and it is now standardized by the ONF [4]. In the following, we first give an overview of the structure of OpenFlow and then describe the features supported by the different specifications.

Overview

The OpenFlow architecture consists of three basic concepts. (110) The network is built up by OpenFlow-compliant switches that compose the data plane; (102) the control plane consists of one or more OpenFlow controllers; (3) a secure control channel connects the switches with the control plane. In the following, we discuss OpenFlow switches and controllers and the interactions among them.

An OpenFlow-compliant switch is a basic forwarding device that forwards packets according to its flow table. This table holds a set of flow table entries, each of which consists of match fields, counters and instructions, as illustrated in Figure 2. Flow table entries are also called flow rules or flow entries. The “header fields” in a flow table entry describe to which packets this entry is applicable. They consist of a wildcard-capable match over specified header fields of packets. To allow fast packet forwarding with OpenFlow, the switch requires ternary content addressable memory (TCAM) that allows the fast lookup of wildcard matches. The header fields can match different protocols depending on the OpenFlow.

specification, e.g., Ethernet, IPv4, IPv6 or MPLS. The “counters” are reserved for collecting statistics about flows. They store the number of received packets and bytes, as well as the duration of the flow. The “actions” specify how packets of that flow are handled. Common actions are “forward”, “drop”, “modify field”, etc.

Header Fields	Counters	Action

Figure 1.7.1. Flow table entry for OpenFlow 1.0.

A software program, called the controller, is responsible for populating and manipulating the flow tables of the switches. By insertion, modification and removal of flow entries, the controller can modify the behavior of the switches with regard to forwarding. The OpenFlow specification defines the protocol that enables the controller to instruct the switches. To that end, the controller uses a secure control channel.

Three classes of communication exist in the OpenFlow protocol: controller-to-switch, asynchronous and symmetric communication (90). The controller-to-switch communication is responsible for feature detection, configuration, programming the switch and information retrieval. Asynchronous communication is initiated by the OpenFlow-compliant switch without any solicitation from the controller. It is used to inform the controller about packet arrivals, state changes at the switch and errors. Finally, symmetric messages are sent without solicitation from either side, i.e., the switch or the controllers are free to initiate the communication without solicitation from the other side. Examples for symmetric communication are hello or echo messages that can be used to identify whether the control channel is still live and available (115).

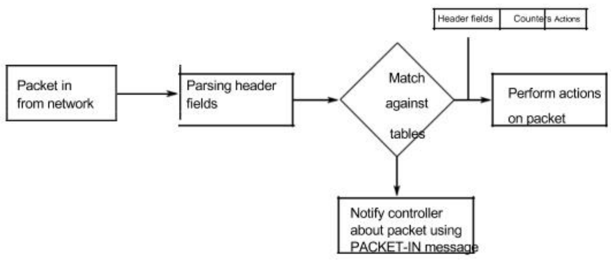

Figure 1.7.2. Basic packet forwarding with OpenFlow in a switch.

The basic packet forwarding mechanism with OpenFlow is illustrated in Figure 1.7.2. When a switch receives a packet, it parses the packet header, which is matched against the flow table. If a flow table entry is found where the header field wildcard matches the header, the entry is considered. If several such entries are found, packets are matched based on prioritization, i.e., the most specific entry or the wildcard with the highest priority is selected. Then, the switch updates the counters of that flow table entry. Finally, the switch performs the actions specified by the flow table entry on the packet, e.g., the switch forwards the packet to a port. Otherwise, if no flow table entry matches the packet header, the switch generally notifies its controller about the packet, which is buffered when the switch is capable of buffering. To that end, it encapsulates either the unbuffered packet or the first bytes of the buffered packet using a PACKET-IN message and sends it to the controller; it is common to encapsulate the packet header and the number of bytes defaults to 128. The controller that receives the PACKET-IN notification identifies the correct action for the packet and installs one or more appropriate entries in the requesting switch. Buffered packets are then forwarded according to the rules; this is triggered by setting the buffer ID in the flow insertion message or in explicit PACKET-OUT messages. Most commonly, the controller sets up the whole path for the packet in the network by modifying the flow tables of all switches on the path (96).

OpenFlow Specifications

We now review the different OpenFlow specifications by highlighting the supported operations and the changes compared to their previous major version and summarize the features of the different versions. Finally, we briefly describe the OpenFlow Configuration and Management Protocol OF-CONFIG protocol, which adds configuration and management support to OpenFlow switches (84).

OpenFlow 1.0

The OpenFlow 1.0 specification [22] was released in December, 2009. As of this writing, it is the most commonly deployed version of OpenFlow. Ethernet and IP packets can be matched based on the source and destination address. In addition, Ethernet-type and VLAN fields can be matched for Ethernet, the differentiated services (DS) and Explicit Congestion Notification (ECN) fields, and the protocol field can be matched for IP. Moreover, matching on TCP or UDP source and destination port numbers is possible.

Figure 1.7.2 illustrates the packet handling mechanism of OpenFlow 1.0. The OpenFlow standard exactly specifies the packet parsing and matching algorithm. The packet matching algorithm starts with a comparison of the Ethernet and VLAN fields and continues if necessary with IP header fields. If the IP type signals TCP or UDP, the corresponding transport layer header fields are considered.

Several actions can be set per flow. The most important action is the forwarding action. This action forwards the packet to a specific port or floods it to all ports. In addition, the controller can instruct the switch to encapsulate all packets of a flow and send them to the controller. An action to drop packets is also available. This action enables the implementation of network access control with OpenFlow. Another action allows modifying the header fields of the packet, e.g., modification of the VLAN tag, IP source, destination addresses, etc.

Statistics can be gathered using various counters in the switch. They may be queried by the controller. It can query table statistics that contain the number of active entries and processed packets. Statistics about flows are stored per flow inside the flow table entries. In addition, statistics per port and per queue are also available (73).

OpenFlow 1.0 provides basic quality of service (QoS) support using queues, and OpenFlow 1.0 only supports minimum rate queues. An OpenFlow-compliant switch can contain one ore more queues, and each queue is attached to a port. An OpenFlow controller can query the information about queues of a switch. The “Enqueue” action enables forwarding to queues. Packets are treated according to the queue properties. Note that OpenFlow controllers are only able to query, but not to set, queue properties. The OF-CONFIG protocol allows one to modify the queue properties, but requires OpenFlow 1.2 and later. We present OF-CONFIG in Section 3.2.7.

OpenFlow 1.1

OpenFlow 1.1 [23] was released in February, 2011. It contains significant changes compared to OpenFlow 1.0. Packet processing works differently now. Packets are processed by a pipeline of multiple flow tables. Two major changes are introduced: a pipeline of multiple flow tables and a group table.

We first explain the pipeline. With OpenFlow 1.0, the result of the packet matching is a list of actions that are applied to the packets of a flow. These actions are directly specified by flow table entries, as shown in Figures 1.7.5 and 1.7.6. With OpenFlow 1.1, the result of the pipeline is a set of actions that are accumulated during pipeline execution and are applied to the packet at the end of the pipeline. The OpenFlow table pipeline requires a new metadata field, instructions and action sets. The metadata field may collect metadata for a packet during the matching process and carry them from one pipeline step to the next. Flow table entries contain instructions instead of actions, as shown in Figure 1.7.4. The list of possible instructions for OpenFlow 1.1 are given in Table 1. The "Apply-Actions" instruction directly applies actions to the packet. The specified actions are not added to the action set. The "Write-Actions" instruction adds the specified actions to the action set and allows for incremental construction of the action set during pipeline execution. The "Clear-Actions" instruction empties the action set (77). The "Write-Metadata" instruction updates the metadata field by applying the specified mask to the current metadata field. Finally, the "Goto" instruction refers to a flow table, and the matching process continues with this table. To avoid loops in the pipeline, only tables with a higher ID than the current table are allowed to be referenced. Thus, the matching algorithm will deterministically terminate. If no "Goto" instruction is specified, the pipeline processing stops, and the accumulated action set is executed on the packet (105).

Header Fields	Counters	Instructions

Figure 1.7.4. Flow table entry for OpenFlow 1.1 and later.

Table 1. List of instructions for OpenFlow 1.1.

Instruction	Argument	Semantic
Apply-Actions	Action(s)	Applies actions immediately without adding them to the action set
Write-Actions	Action(s)	Merge the specified action(s) into the action set
Clear-Actions	-	Clear the action set
Write-Metadata	Metadata mask	Updates the metadata field
Goto-Table	Table ID	Perform matching on the next table

Figure 1.7.5 illustrates the packet processing of the pipeline. Before the pipeline begins, the metadata field and the action set for a packet are initialized as empty. The matching process starts on the first flow table. The packet is matched against the consecutive flow tables from each of which the highest-priority matching flow table entry is selected. The pipeline ends when no matching flow table entry is found or no "Goto" instruction is set in the matching flow table entry. The pipeline supports the definition of complex forwarding mechanisms and provides more flexibility compared to the switch architecture of OpenFlow 1.0.

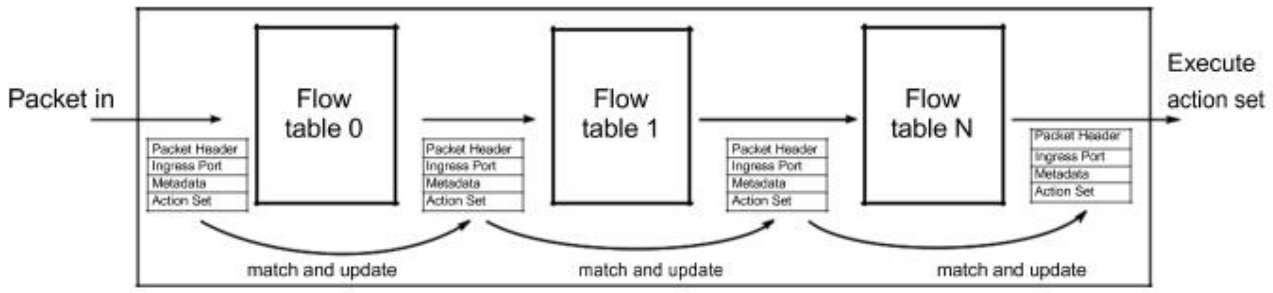

Figure 1.7.5. OpenFlow pipeline.

The second major change is the addition of a group table. The group table supports more complex forwarding behaviors, which are possibly applied to a set of flows. It consists of group table entries, as shown in Figure 1.7.6. A group table entry may be performed if a flow table entry uses an appropriate instruction that refers to its group identifier. In particular, multiple flow table entries can point to the same group identifier, so that the group table entry is performed for multiple flows.

Group Identifier	Group Type	Counters	Action Buckets

Figure 1.7.6. Group table entries for OpenFlow 1.1 and later.

The group table entry contains a group type, a counters field and a field for action buckets. The counters are used for collecting statistics about packets that are processed by this group. A single action bucket contains a set of actions that may be executed, depending on the group type. There are possibly multiple action buckets for a group table entry. The group types define which of them are applied. Four group types exist, and we illustrate the use of two of them.

The group type "all" is used to implement broadcast and multicast. Packets of this group will be processed by all action buckets. The actions of each bucket are applied to the packet consecutively. For example, a group table entry with the type "all" contains two action buckets. The first bucket consists of the action "forward to Port 1". The second bucket consists of the action "forward to Port 2". Then, the switch sends the packet both to Port 1 and Port 2.

The group type "fast failover" is used to implement backup paths. We first explain the concept of liveness for illustration purposes. An action bucket is considered live if all actions of the bucket are considered live. The liveness of an action depends on the liveness of its associated port. However, the liveness property of a port is managed outside of the OpenFlow protocol. The OpenFlow standard only specifies the minimum requirement that a port should not be considered live if the port or the link is down. A group with the type "fast failover" executes the first live action bucket, and we explain this by the following example. The primary path to a flow's destination follows Port 3, while its backup path follows Port 4. This may be configured by a group table with the first action bucket containing the forwarding action towards Port 3 and a second action bucket containing the forwarding action towards Port 4. If Port 3 is up, packets belonging to this group are forwarded using Port 3; otherwise, they are forwarded using Port 4. Thus, the "fast failover" group type supports reroute decisions that do not require immediate controller interaction. Thus, a fast reroute mechanisms can be implemented that ensures minimum packet loss in failure cases. Fast reroute mechanisms, such as the MPLS fast reroute [69] or the IP fast reroute [70] can be implemented with OpenFlow group tables.

As an optional feature, OpenFlow 1.1 performs matching of MPLS labels and traffic classes. Furthermore, MPLS-specific actions, like pushing and popping MPLS labels, are supported. In general, the number of supported actions for OpenFlow 1.1 is larger than for OpenFlow 1.0. For example, the Time-To-Live (TTL) field in the IP header can be decremented, which is unsupported in OpenFlow 1.0. OpenFlow 1.1 provides additional statistics fields due to the changed switch architecture. Controllers can query statistics for the group table and group entries, as well as for action buckets (99).

OpenFlow 1.2
OpenFlow 1.2 [26] was released in December, 2011. It comes with extended protocol support, in particular for IPv6. OpenFlow 1.2 can match IPv6 source and destination addresses, protocol number, flow label, traffic class and various ICMPv6 fields. Vendors have new possibilities to extend OpenFlow by themselves to support additional matching capabilities. A type-length-value (TLV) structure, which is called OpenFlow Extensible Match (OXM), allows one to define new match entries in an extensible way (59).

With OpenFlow 1.2, a switch may simultaneously be connected to more than a single controller, i.e., it can be configured to be administrated by a set of controllers. The switch initiates the connection, and the controllers accept the connection attempts. One controller is defined master and programs the switch. The other controllers are slaves. A slave controller can be promoted to the master role, while the master is demoted to the slave role. This allows for controller failover implementations (109).

OpenFlow 1.3
OpenFlow 1.3 [27] introduces new features for monitoring and operations and management (OAM). To that end, the meter table is added to the switch architecture. Figure 7 shows the structure of meter table entries. A meter is directly attached to a flow table entry by its meter identifier and measures the rate of packets assigned to it. A meter band may be used to rate-limit the associated packet or data rate by dropping packets when a specified rate is exceeded. Instead of dropping packets, a meter band may optionally recolor such packets by modifying their differentiated services (DS) field. Thus, simple or complex QoS frameworks can be implemented with OpenFlow 1.3 and later specifications.

Meter Identifier	Meter Bands	Counters

Figure 1.7.7. Meter table entry.

The support for multiple controllers is extended. With OpenFlow 1.2, only fault management is targeted by a master/slave scheme. With OpenFlow 1.3, arbitrary auxiliary connections can be used to supplement the connection with the master controller and the switch. Thereby, better load balancing in the control plane may be achieved. Moreover, per-connection event filtering is introduced. This allows controllers to subscribe only to message types they are interested in. For example, a controller responsible for collecting statistics about the network can be attached as the auxiliary controller and subscribes only to statistics events generated by the switches.

OpenFlow 1.3 supports IPv6 extension headers. This includes, e.g., matching on the encrypted security payload (ESP) IPv6 header, IPv6 authentication header, or hop-by-hop IPv6 header. Furthermore, support for Provider Backbone Bridge (PBB) is added, as well as other minor protocol enhancements.

OpenFlow 1.4

OpenFlow 1.4 [88] was released in October 2013. The ONF improved the support for the OpenFlow Extensible Match (OXM). TLV structures for ports, tables, and queues are added to the protocol, and hard-coded parts from earlier specifications are now replaced by the new TLV structures. The configuration of optical ports is now possible. In addition, controllers can send control messages in a single message bundle to switches. Minor improvements of group tables, flow eviction on full tables and monitoring features are also included.

Summary of OpenFlow Specifications and Controllers

Table 2 provides the supported protocols and available match fields of the discussed OpenFlow versions. Table 3 compiles the types of statistics collected by a switch, which can be queried by a controller.

Table 2. OpenFlow (OF) match fields.

	OF 1.0	OF 1.1	OF 1.2	OF 1.3 & OF 1.4
Ingress Port	X	X	X	X
Metadata		X	X	X
Ethernet: src, dst, type	X	X	X	X
IPv4: src, dst, proto, ToS	X	X	X	X
TCP/UDP: src port, dst port	X	X	X	X
MPLS: label, traffic class		X	X	X
OpenFlow Extensible Match (OXM)			X	X
IPv6: src, dst, flow label, ICMPv6			X	X
IPv6 Extension Headers				X

A list of open source controllers is given in Table 4. The NOX controller [29] was initially developed at Stanford University and can be downloaded from [30]. It is written in C++ and licensed under the GNU General Public License (GPL). The NOX controller was used in many research papers. The POX [90] controller is a rewrite of the NOX controller in Python and can be used on various platforms. Initially, POX was also published under the GPL, but has been available under the Apache Public License (APL) since November, 2013. The Beacon [91] controller was also developed at Stanford University but is written in Java. The controller is available under a BSD license. The Floodlight controller [92] is a fork of the Beacon controller and is sponsored by Big Switch Networks. It is licensed under the APL.The Maestro controller [93] was developed at Rice University and written in Java. The authors emphasize the use of multi-threading to improve the performance of the controller in larger networks. It is licensed under the GNU Lesser General Public License (LGPL). The NodeFLow [114] controller is written in Java and is based on the Node.JS library. It is available under the MIT license. The Trema [115] controller is written in C and Ruby. It is possible to write plugins in C and in Ruby for that controller. It is licensed under the GPL and developed by NEC. Finally, the OpenDaylight controller [116] is written in Java and hosted by the Linux Foundation. The OpenDaylight controller has no restriction on the operating system and is not bound to Linux. The controller is published under the Eclipse Public License (EPL).

Table 3. Statistics are measured for different parts of the OpenFlow switch.

	OF 1.0	OF 1.1	OF 1.2	OF 1.3 & OF 1.4
Per table statistics	X	X	X	X
Per flow statistics	X	X	X	X
Per port statistics	X	X	X	X
Per queue statistics	X	X	X	X
Group statistics		X	X	X
Action bucket statistics		X	X	X
Per-flow meter				X
Per-flow meter band				X

Table 4. List of available open source OpenFlow controllers. LGPL, Lesser General Public License; EPL, Eclipse Public License.

Name	Programming language	License	Comment
NOX[29]	C++	GPL	Initially developed at Stanford University. NOX can be downloaded from [30].
POX[30]	Python	Apache	Forked from the NOX controller. POX is written in Python and runs under various platforms.
Beacon [31]	Java	BSD	Initially developed at Stanford.
Floodlight [32]	Java	Apache	Forked from the Beacon controller and sponsored by Big Switch Networks.
Maestro [33]	Java	LGPL	Multi-threaded OpenFlow controller developed at Rice University.
NodeFLow [34]	JavaScript	MIT	JavaScript OpenFlow controller based on Node.JS.
Trema [35]	C and Ruby	GPL	Plugins can be written in C and in Ruby. Trema is developed by NEC.
OpenDaylight [36]	Java	EPL	OpenDaylight is hosted by the Linux Foundation, but has no restrictions on the operating system.

OF-CONFIG

The OF-CONFIG protocol adds configuration and management support to OpenFlow switches. It is also standardized by the ONF. OF-CONFIG provides the configuration of various switch parameters that are not handled by the OpenFlow protocol. This includes features like setting the administrative controllers, the configuration of queues and ports, etc. The mentioned configuration possibilities are part of OF-CONFIG 1.0 [67], which was released in December, 2011. The initial specification requires OpenFlow 1.2 and later. As of this writing, the current specification is OF-CONFIG 1.1.1 and supports the configuration of OpenFlow 1.3 switches (68).

CONCLUSION

SDN can be expanded beyond the actual match lactation paradigm. For example, it could integrate middleboxes or programmable custom packet processors.
This integration could offer new services like on-the-fly encryption, transcoding, or traffic classification. This requires coordination, consensus, and vendor support.
In the control plane, the composing and complying of heterogeneous components are still difficult. For example, compose applications using Beacon, POX, or floodlight simultaneously.
Finally, remark the SDN is a tool. They create new innovative services and applications.

References

[1] Qiang duan, Nirwan Ansari, and Mehmet Toy; software defined Network Virtualization: An Architectural. Framework for integrating SDN and NFV for service provisioning in Future networks
[2] Astuto, B.N.; Mendonça, M.; Nguyen, X.N.; Obraczka, K.; Turletti, T.A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. IEEE Commun. Surv. Tutor. 2014, doi:10.1109/SURV.2014.012214.00180.
[3] Jain, R.; Paul, S. Network Virtualization and Software Defined Networking for Cloud Computing: A Survey. IEEE Commun. Mag. 2013, 51, 24–31. Open Networking Foundation. Available online: https://www.opennetworking.org/ (accessed on 22 July 2013).
[4] Doria, A.; Salim, J.H.; Haas, R.; Khosravi, H.; Wang, W.; Dong, L.; Gopal, R.; Halpern, J. Forwarding and Control Element Separation (ForCES) Protocol Specification. RFC 5810 (Proposed Standard), 2010. Available online: https://datatracker.ietf.org/doc/rfc5810/ (accessed on 22 July 2013).
[5] Yang, L.; Dantu, R.; Anderson, T.; Gopal, R.Forwarding and Control Element Separation (ForCES) Framework. RFC 3746 (Informational), 2004. Available online: https://datatracker.ietf.org/doc/rfc3746/ (accessed on 22 July 2013).
[6] Hares, S. Analysis of Comparisons between OpenFlow andForCES. Internet Draft (Informational), 2012. Available online: https://datatracker.ietf.org/doc/draft-hares-forces-vs-openflow/ (accessed on 17 February 2014).
[7] Haleplidis, E.; Denazis, S.; Koufopavlou, O.; Halpern, J.; Salim, J.H. Software-Defined Networking: Experimenting with the Control to Forwarding Plane Interface. In Proceedings of the European Workshop on Software Defined Networks (EWSDN), Darmstadt, Germany, 25–26 October 2012; pp. 91–96.
[8] Lakshman, T.V.; Nandagopal, T.; Ramjee, R.; Sabnani, K.; Woo, T. The SoftRouter Architecture. In Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets), San Diego, CA, USA, 15–16 November 2004.

[9] Zheng, H.; Zhang, X. Path Computation Element to Support Software-Defined Transport Networks Control Internet-Draft (Informational). 2014. Available online: https://datatracker.ietf.org/doc/draft-zheng-pce-for-sdn-transport/ (accessed on 2 March 2014).
[10] Rodriguez-Natal, A.; Barkai, S.; Ermagan, V.; Lewis, D.; Maino, F.; Farinacci, D. Software Defined Networking Extensions for the Locator/ID Separation Protocol. Internet Draft (Experimental), 2014. Available online: http://wiki.tools.ietf.org/id/ draft-rodrigueznatal-lisp-sdn-00.txt (accessed on 2 March 2014).
[11] Rexford, J.; Freedman, M.J.; Foster, N.; Harrison, R.; Monsanto, C.; Reitblatt, M.; Guha, A.; Katta, N.P.; Reich, J.; Schlesinger, C. Languages for Software-Defined Networks. IEEE Commun. Mag. 2013, 51, 128–134.
[12] Foster, N.; Harrison, R.; Freedman, M.J.; Monsanto, C.; Rexford, J.; Story, A.; Walker, D. Frenetic: A Network Programming Language. In Proceedings of the ACM SIGPLAN International Conference on Functional Programming, Tokyo, Japan, 19–21 September 2011.
[13] Monsanto, C.; Reich, J.; Foster, N.; Rexford, J.; Walker, D. Composing Software-Defined Networks. In Proceedings of the USENIX Syposium on Networked Systems Design & Implementation (NSDI), Lombard, IL, USA, 2–5 April 2013; pp. 1–14.
[14] Voellmy, A.; Kim, H.; Feamster, N. Procera: A Language for High-Level Reactive Network Control. In Proceedings of the ACM Workshop on Hot Topics in Software Defined Networks (HotSDN), Helsinki, Finland, 13–17 August 2012; pp. 43–48.
[15] Facca, F.M.; Salvadori, E.; Karl, H.; Lopez, D.R.; Gutierrez, P.A.A.; Kostic, D.; Riggio, R. NetIDE: First Steps towards an Integrated Development Environment for Portable Network Apps. In Proceedings of the European Workshop on Software Defined Networks (EWSDN), Berlin, Germany, 10–11 October 2013; pp. 105–110.
[16] Tennenhouse, D.L.; Wetherall, D.J. Towards an Active Network Architecture. ACM SIGCOMM Comput. Commun. Rev. 1996, 26, 5–18.
[17] Campbell, A.T.; De Meer, H.G.; Kounavis, M.E.; Miki, K.; Vicente, J.B.; Villela, D. A Survey of Programmable Networks. ACM SIGCOMM Comput. Commun. Rev. 1999, 29, 7–23.
[18] Feamster, N.; Rexford, J.; Zegura, E. The Road to SDN: An Intellectual History of Programmable Networks. ACM Queue 2013, 12, 20–40.
[19] Chan, M.C.; Huard, J.F.; Lazar, A.A.; Lim, K.S. On Realizing a Broadband Kernel for Multimedia Networks. In Proceedings of the International COST 237 Workshop on Multimedia Telecommunications and Applications, Barcelona, Spain, 25–27 November 1996; pp. 56–74.
[20] McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S.; Turner, J. OpenFlow: Enabling Innovation in Campus Networks. ACM SIGCOMM Comput. Commun. Rev. 2008, 38, 69–74.
[21] OpenFlow Switch Consortium and Others. OpenFlow Switch Specification Version 1.0.0. 2009. Available online: https://www.opennetworking.org/images/stories/downloads/sdn-resources/ onf-specifications/openflow/openflow-spec-v1.3.0.pdf (accessed on 25 November 2013).
[22] OpenFlow Switch Consortium and Others. OpenFlow Switch Specification Version 1.1.0. 2011. Available online: http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf (accessed on 25 November 2013).
[23] Pan, P.; Swallow, G.; Atlas, A. RFC4090: Fast Reroute Extensions to RSVP-TE for LSP Tunnels, 2005. Available online: https://datatracker.ietf.org/doc/rfc4090/ (accessed on 22 July 2013).
[24] Atlas, A.; Zinin, A. RFC5286: Basic Specification for IP Fast Reroute: Loop-Free Alternates, 2008. Available online: https://tools.ietf.org/html/rfc5286 (accessed on 22 July 2013).
[25] OpenFlow Switch Consortium and Others. OpenFlow Switch Specification Version 1.2.0. 2011. Available online: https://www.opennetworking.org/images/stories/downloads/sdn-resources/ onf specifications/openflow/openflow-spec-v1.2.pdf (accessed on 25 November 2013).

[26] OpenFlow Switch Consortium and Others. OpenFlow Switch Specification Version 1.3.0. 2012. Available online: https://www.opennetworking.org/images/stories/downloads/sdn-resources/ onf-specifications/openflow/openflow-spec-v1.3.0.pdf (accessed on 25 November 2013).
[27] OpenFlow Switch Consortium and Others. OpenFlow Switch Specification Version 1.4.0. 2013. Available online: https://www.opennetworking.org/images/stories/downloads/sdn-resources/ onf-specifications/openflow/openflow-spec-v1.4.0.pdf (accessed on 12 January 2014).
[28] Gude, N.; Koponen, T.; Pettit, J.; Pfaff, B.; Casado, M.; McKeown, N.; Shenker, S. NOX: Towards an Operating System for Networks. ACM SIGCOMM Comput. Commun. Rev. 2008, 38, 105–110.
[29] NOXrepo.org. Available online: http://www.noxrepo.org (accessed on 16 November 2013).
[30] Erickson, D. The Beacon OpenFlow Controller. In Proceedings of the ACM Workshop on Hot Topics in Software Defined Networks (HotSDN), Hong Kong, China, 12–16 August 2013;13–18.
[31] Project Floodlight: Open Source Software for Building Software-Defined Networks. Available online: http://www.projectfloodlight.org/floodlight/ (accessed on 16 November 2013).
[32] Cai, Z.; Cox, A.L.; Eugene Ng, T.S. Maestro: Balancing Fairness, Latency and Throughput in the OpenFlow Control Plane; Technical Report; Rice University: Houston, TX, USA, 2011. NodeFLow OpenFlow Controller. Available online: https://github.com/gaberger/NodeFLow (accessed on 16 November 2013).
[33] Trema: Full-Stack OpenFlow Framework in Ruby and C. Available online: http://trema.github.io/trema/ (accessed on 16 November 2013).
[34] OpenDaylight. Available online: http://www.opendaylight.org/ (accessed on 22 February 2014).
[35] OpenFlow Switch Consortium and Others. Configuration and Management Protocol OF-CONFIG 1.0. 2011. Available online: https://www.opennetworking.org/images/ stories/downloads/sdn-resources/onf-specifications/openflow-config/of-config1dot0-final.pdf (accessed on 25 November 2013).
[36] Jain, S.; Kumar, A.; Mandal, S.; Ong, J.; Poutievski, L.; Singh, A.; Venkata, S.; Wanderer, J.; Zhou, J.; Zhu, M.; et al. B4: Experience with a Globally-Deployed Software Defined WAN. In Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, Hong Kong, China, 13–17 August 2013; pp. 3–14.
[37] Kotani, D.; Suzuki, K.; Shimonishi, H. A Design and Implementation of OpenFlow Controller handling IP Multicast with Fast Tree Switching. In Proceedings of the IEEE/IPSJ International Symposium on Applications and the Internet (SAINT), Izmir, Turkey, 16–20 July 2012; 60–67.
[38] Nakao, A. FLARE: Open Deeply Programmable Network Node Architecture. Available online: http://netseminar.stanford.edu/seminars/10_18_12.pdf 2012. (accessed on 24 January 2014).
[39] Reitblatt, M.; Foster, N.; Rexford, J.; Schlesinger, C.; Walker, D. Abstractions for Network Update. In Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, Helsinki, Finland, 13–17 August 2012; pp. 323–334.
[40] Mattos, D.; Fernandes, N.; da Costa, V.; Cardoso, L.; Campista, M.; Costa, L.; Duarte, O. OMNI: OpenFlow MaNagement Infrastructure. In Proceedings of the International Conference on the Network of the Future (NOF), Paris, France, 28–30 November 2011; pp. 52–56.
[41] Wang, R.; Butnariu, D.; Rexford, J. OpenFlow-Based Server Load Balancing Gone Wild. In Proceedings of the USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE), Boston, MA, USA, 29 March 2011; pp.12–12.
[42] Gember, A.; Prabhu, P.; Ghadiyali, Z.; Akella, A. Toward Software-Defined Middlebox Networking. In Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets), Redmond, WA, USA, 29–30 October 2012; pp. 7–12.

[43] Initial Thoughts on Custom Network Processing via Waypoint Services. In Proceedings of the Workshop on Infrastrucures for Software/Hardware Co-Design (WISH), Chamonix, France, 2 April 2011; pp. 15–20.
[44] ETSI—Network Functions Industry Specification Group. Network Functions Virtualisation (NFV). 2013. Available online: http://portal.etsi.org/NFV/NFV_White_Paper2.pdf (accessed on 29 October 2013).
[45] Boucadair, M.; Jacquenet, C. Service Function Chaining: Framework & Architecture.
[46] Internet Draft (Intended Status: Standards Track), 2014. Available online: https://tools.ietf.org/search/draft-boucadair-sfc-framework-02 (accessed on 20 February 2014).
[47] John, W.; Pentikousis, K.; Agapiou, G.; Jacob, E.; Kind, M.; Manzalini, A.; Risso, F.; Staessens, D.; Steinert, R.; Meirosu, C. Research Directions in Network Service Chaining. In Proceedings of the IEEE Workshop on Software Defined Networks for Future Networks and Services (SDN4FNS), Trento, Italy, 11–13 November 2013; pp. 1–7.
[48] Nayak, A.; Reimers, A.; Feamster, N.; Clark, R. Resonance: Inference-based Dynamic Access Control for Enterprise Networks. In Proceedings of the Workshop on Research on Enterprise Networking (WREN), Barcelona, Spain, 21 August 2009; pp. 11–18.
[49] Khurshid, A.; Zhou, W.; Caesar, M.; Godfrey, P.B. VeriFlow: Verifying Network-Wide Invariants in Real Time. In Proceedings of the ACM Workshop on Hot Topics in Software Defined Networks (HotSDN), Helsinki, Finland, 13–17 August 2012; pp. 49–54.
[50] Yao, G.; Bi, J.; Xiao, P. Source Address Validation Solution with OpenFlow/NOX Architecture. In Proceedings of the IEEE International Conference on Network Protocols (ICNP), Vancouver, BC, Canada, 17–20 October 2011; pp. 7–12.
[51] Jafarian, J.H.; Al-Shaer, E.; Duan, Q. Openflow Random Host Mutation: Transparent Moving Target Defense Using Software Defined Networking. In Proceedings of the ACM Workshop on Hot Topics in Software Defined Networks (HotSDN), Helsinki, Finland, 13–17 August 2012; pp. 127–132.
[52] YuHunag, C.; MinChi, T.; YaoTing, C.; YuChieh, C.; YanRen, C. A Novel Design for Future On-Demand Service and Security. In Proceedings of the International Conference on Communication Technology (ICCT), Nanjing, China, 11–14 November 2010; pp. 385–388.
[53] Braga, R.; Mota, E.; Passito, A. Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow. In Proceedings of the IEEE Conference on Local Computer Networks (LCN), Denver, CO, USA, 11–14 October 2010; pp. 408–415.
[54] Porras, P.; Shin, S.; Yegneswaran, V.; Fong, M.; Tyson, M.; Gu, G. A Security Enforcement Kernel for OpenFlow Networks. In Proceedings of the ACM Workshop on Hot Topics in Software Defined Networks (HotSDN), Helsinki, Finland, 13–17 August 2012; pp. 121–126.
[55] Handigol, N.; Heller, B.; Jeyakumar, V.; Maziéres, D.; McKeown, N. Where is the Debugger for My Software-defined Network? In Proceedings of the ACM Workshop on Hot Topics in Software Defined Networks (HotSDN), Helsinki, Finland, 13–17 August 2012; pp. 55–60.
[56] Wundsam, A.; Levin, D.; Seetharaman, S.; Feldmann, A. OFRewind: Enabling Record and Replay Troubleshooting for Networks. In Proceedings of the USENIX Annual Technical Conference, Portland, OR, USA, 15–17 June 2011.
[57] Kuzniar, M.; Peresini, P.; Canini, M.; Venzano, D.; Kostic, D. A SOFT Way for Openflow Switch Interoperability Testing. In Proceedings of the ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT), Nice, France, 10–13 December 2012; pp. 265–276.
[58] Canini, M.; Venzano, D.; Perešíni, P.; Kostic,´ D.; Rexford, J. A NICE Way to Test Openflow Applications. In Proceedings of the USENIX Syposium on Networked Systems Design & Implementation (NSDI), San Jose, CA, USA, 25–27 April 2012.

[59] Heller, B.; Scott, C.; McKeown, N.; Shenker, S.; Wundsam, A.; Zeng, H.; Whitlock, S.; Jeyakumar, V.; Handigol, N.; McCauley, J.; et al. Leveraging SDN Layering to Systematically Troubleshoot Networks. In Proceedings of the ACM Workshop on Hot Topics in Software Defined Networks (HotSDN), Hong Kong, China, 12–16 August 2013; pp. 37–42.
[60] Nascimento, M.R.; Rothenberg, C.E.; Salvador, M.R.; Magalhães, M.F. QuagFlow: Partnering Quagga with OpenFlow. ACM SIGCOMM Comput. Commun. Rev. 2010, 40, 441–442.
[61] Nascimento, M.R.; Rothenberg, C.E.; Salvador, M.R.; Corrêa, C.N.A.; de Lucena, S.; Magalhães, M.F. Virtual Routers as a Service: The Routeflow Approach Leveraging Software-Defined Networks. In Proceedings of the International Conference on Future Internet Technologies (CFI), Seoul, Korea, 13–15 June 2011; pp. 34–37.
[62] Bennesby, R.; Fonseca, P.; Mota, E.; Passito, A. An Inter-AS Routing Component for Software-Defined Networks. In Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS), Maui, HI, USA, 16–20 April 2012; pp. 138–145.
[63] Caesar, M.; Caldwell, D.; Feamster, N.; Rexford, J.; Shaikh, A.; van der Merwe, J. Design and Implementation of a Routing Control Platform. In Proceedings of the USENIX Syposium on Networked Systems Design & Implementation (NSDI), Boston, MA, USA, 2–4 May 2005; pp. 15–28.
[64] Rothenberg, C.E.; Nascimento, M.R.; Salvador, M.R.; Corrêa, C.N.A.; de Lucena, S.C.; Raszuk, R. Revisiting Routing Control Platforms with the Eyes and Muscles of Software-Defined Networking. In Proceedings of the ACM Workshop on Hot Topics in Software Defined Networks (HotSDN), Helsinki, Finland, 13–17 August 2012; pp. 13–18.
[65] Sharafat, A.R.; Das, S.; Parulkar, G.; McKeown, N. MPLS-TE and MPLS VPNS with OpenFlow. ACM SIGCOMM Comput. Commun. Rev. 2011, 41, 452–453.
[66] Azodolmolky, S.; Nejabati, R.; Escalona, E.; Jayakumar, R.; Efstathiou, N.; Simeonidou, D. Integrated OpenFlow-GMPLS Control Plane: An Overlay Model for Software Defined Packet Over Optical Networks. In Proceedings of the European Conference and Exposition on Optical Communications, Geneva, Switzerland, 18–22 September 2011.
[67] Gutz, S.; Story, A.; Schlesinger, C.; Foster, N. Splendid Isolation: A Slice Abstraction for Software-Defined Networks. In Proceedings of the ACM Workshop on Hot Topics in Software Defined Networks (HotSDN), Helsinki, Finland, 13–17 August 2012; pp. 79–84.
[68] Ferguson, A.D.; Guha, A.; Liang, C.; Fonseca, R.; Krishnamurthi, S. Hierarchical Policies for Software Defined Networks. In Proceedings of the ACM Workshop on Hot Topics in Software Defined Networks (HotSDN), Helsinki, Finland, 13–17 August 2012; pp. 37–42.
[69] Banikazemi, M.; Olshefski, D.; Shaikh, A.; Tracey, J.; Wang, G. Meridian: An SDN Platform for Cloud Network Services. IEEE Commun. Mag. 2013, 51, 120–127.
[70] The OpenStack Foundatation. 2013. Available online: http://www.openstack.org/ (accessed on 22 July 2013).
[71] Heller, B.; Sherwood, R.; McKeown, N. The Controller Placement Problem. In Proceedings of the ACM Workshop on Hot Topics in Software Defined Networks (HotSDN), Helsinki, Finland, 13–17 August 2012; pp. 7–12.
[72] Hock, D.; Hartmann, M.; Gebert, S.; Jarschel, M.; Zinner, T.; Tran-Gia, P. Pareto-Optimal Resilient Controller Placement in SDN-based Core Networks. In Proceedings of the 25th International Teletraffic Congress (ITC), Shanghai, China, 10–12 September 2013; pp. 1–9.
[73] Tootoonchian, A.; Ganjali, Y. HyperFlow: A Distributed Control Plane for OpenFlow. In Proceedings of the USENIX Workshop on Research on Enterprise Networking (WREN), San Jose, CA, USA, 27 April 2010; p. 3.
[74] Yeganeh, S.H.; Ganjali, Y. Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications. In Proceedings of the ACM Workshop on Hot Topics in Software Defined Networks (HotSDN), Helsinki, Finland, 13–17 August 2012; pp. 19–24.

[75] Levin, D.; Wundsam, A.; Heller, B.; Handigol, N.; Feldmann, A. Logically Centralized?: State Distribution Trade-offs in Software Defined Networks. In Proceedings of the ACM Workshop on Hot Topics in Software Defined Networks (HotSDN), Helsinki, Finland, 13–17 August 2012; pp. 1–6.
[76] Jarschel, M.; Oechsner, S.; Schlosser, D.; Pries, R.; Goll, S.; Tran-Gia, P. Modeling and Performance Evaluation of an OpenFlow Architecture. In Proceedings of the International Teletraffic Congress (ITC), San Francisco, CA, USA, 6–8 September 2011; pp. 1–7.
[77] Fernandez, M.P. Comparing OpenFlow Controller Paradigms Scalability: Reactive and Proactive. In Proceedings of the International Conference on Advanced Information Networking and Applications (AINA), Barcelona, Spain, 25–28 March 2013; pp. 1009–1016.
[78] CIDR REPORT. Available online: http://www.cidr-report.org/as2.0/ 2013. (accessed on 22 July 2013).
[79] Sarrar, N.; Uhlig, S.; Feldmann, A.; Sherwood, R.; Huang, X. Leveraging Zipf's Law for Traffic Offloading. ACM SIGCOMM Comput. Commun. Rev. 2012, 42, 16–22.
[80] Sarrar, N.; Feldmann, A.; Uhrig, S.; Sherwood, R.; Huang, X. Towards Hardware Accelerated Software Routers. In Proceedings of the ACM CoNEXT Student Workshop, Philadelphia, PA, USA, 3 December 2010; pp. 1–2.
[81] Soliman, M.; Nandy, B.; Lambadaris, I.; Ashwood-Smith, P. Source Routed Forwarding with Software Defined Control, Considerations and Implications. In Proceedings of the ACM CoNEXT Student Workshop, Nice, France, 10–13 December 2012; pp. 43–44.
[82] Ashwood-Smith, P.; Soliman, M.; Wan, T. SDN State Reduction. Internet Draft (Informational), 2013. Available online: https://tools.ietf.org/html/draft-ashwood-sdnrg-state-reduction-00 (accessed on 25 November 2013).
[83] Zhang, X.; Francis, P.; Wang, J.; Yoshida, K. Scaling IP Routing with the Core Router-Integrated Overlay. In Proceedings of the IEEE International Conference on Network Protocols (ICNP), Santa Barbara, CA, USA, 12–15 November 2006; pp. 147–156.
[84] Ballani, H.; Francis, P.; Cao, T.; Wang, J. ViAggre: Making Routers Last Longer! In Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets), Calgary, AB, Canada, 6–7 October 2008; pp. 109–114.
[85] 86. Francis, P.; Xu, X.; Ballani, H.; Jen, D.; Raszuk, R.; Zhang, L. FIB Suppression with Virtual Aggregation. IETF Internet Draft (Informational), 2012. Available online: http://wiki.tools.ietf.org/html/draft-ietf-grow-va-06 (accessed on 16 November 2013).
[86] Masuda, A.; Pelsser, C.; Shiomoto, K. SpliTable: Toward Routing Scalability through Distributed BGP Routing Tables. IEICE Trans. Commun. 2011, E94-B, 64–76.
[87] Rètvàri, G.; Tapolcai, J.; Kõrösi, A.; Majdàn, A.; Heszberger, Z. Compressing IP Forwarding Tables: Towards Entropy Bounds and Beyond. In Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN), Hong Kong, China, 12–16 August 2013; pp. 111–122.
[88] Ford, A.; Raicu, C.; Handley, M.; Bonaventure, O. RFC6824: TCP Extensions for Multipath Operation with Multiple Addresses. IETF Internet Draft (Experimental), 2013. Available online: http://tools.ietf.org/html/rfc6824 (accessed on 22 February 2014).
[89] Sharma, S.; Staessens, D.; Colle, D.; Pickavet, M.; Demeester, P. OpenFlow: Meeting Carrier-Grade Recovery Requirements. Comput. Commun. 2013, 36, 656–665.
[90] Kempf, J.; Bellagamba, E.; Kern, A.; Jocha, D.; Takàcs, A.; Sköldström, P. Scalable Fault Management for OpenFlow. In Proceedings of the IEEE International Conference on Communications (ICC), Ottawa, ON, Canada, 10–15 June 2012; pp. 6606–6610.

[91] Mogul, J.C.; Congdon, P. Hey, You Darned Counters!: Get off my ASIC! In Proceedings of the ACM Workshop on Hot Topics in Software Defined Networks (HotSDN), Helsinki, Finland, 13–17 August 2012; pp. 25–30.
[92] Narayanan, R.; Kotha, S.; Lin, G.; Khan, A.; Rizvi, S.; Javed, W.; Khan, H.; Khayam, S.A. Macroflows and Microflows: Enabling Rapid Network Innovation through a Split SDN Data Plane. In Proceedings of the European Workshop on Software Defined Networks (EWSDN), Darmstadt, Germany, 25–26 October 2012; pp. 79–84.
[93] Lu, G.; Miao, R.; Xiong, Y.; Guo, C. Using Cpu as a Traffic Co-Processing Unit in Commodity Switches. In Proceedings of the ACM Workshop on Hot Topics in Software Defined Networks
[94] (HotSDN), Helsinki, Finland, 13–17 August 2012; pp. 31–36.
[95] Chiba, Y.; Shinohara, Y.; Shimonishi, H. Source Flow: Handling Millions of Flows on Flow-Based Nodes. In Proceedings of the ACM SIGCOMM, New Delhi, India, 30 August–2 September 2010; pp. 465–466.
[96] Bianco, A.; Birke, R.; Giraudo, L.; Palacin, M. Openflow Switching: Data Plane Performance. In Proceedings of the IEEE International Conference on Communications (ICC), Cape Town, South Africa, 23–27 May 2010; pp. 1–5.
[97] Draves, R.; King, C.; Srinivasan, V.; Zill, B. Constructing Optimal IP Routing Tables. In Proceedings of the IEEE Infocom, New York, NY, USA, 21–25 March 1999; pp. 88–90
[98] Liu, A.X.; Meiners, C.R.; Torng, E. TCAM Razor: A Systematic Approach Towards Minimizing Packet Classifiers in TCAMs. IEEE/ACM Trans. Netw. 2010, 18, 490–500.
[99] Meiners, C.R.; Liu, A.X.; Torng, E. Bit Weaving: A Non-Prefix Approach to Compressing Packet Classifiers in TCAMs. IEEE/ACM Trans. Netw. 2012, 20, 488–500.
[100] McGeer, R.; Yalagandula, P. Minimizing Rulesets for TCAM Implementation. In Proceedings of the IEEE Infocom, Rio de Janeiro, Brazil, 19–25 April 2009; pp. 1314–1322.
[101] Yu, M.; Rexford, J.; Freedman, M.J.; Wang, J. Scalable Flow-Based Networking with DIFANE. ACM SIGCOMM Comput. Commun. Rev. 2010, 40, 351–362 .
[102] D. Kreutz et al., "Software-Defined Networking: A Comprehensive Survey," Proc. IEEE, vol. 103, no. 1, Jan. 2015, pp. 14–76.
[103] NM M. K. Chowdhury and R. Boutaba, "A Survey of Network Virtualiza-tion," Elsevier Comp. Networks J., vol. 54, no. 5, Apr. 2010, pp. 862–76.
[104] ETSI NFV ISG, "Network Function Virtualization — Introduction White Paper," Proc. SDN and OpenFlow World Congress, Oct. 2012.
[105] J. Liu et al., "Device-to-Device Communications for Enhancing Quality of Experience in Software Defined Multi-Tier LTE-A Networks," IEEE Network, vol. 29, no. 4, July 2015, pp. 46–52.
[106] M. Casado et al., "Fabric: A Retrospective on Evolving SDN," Proc. 1st Wksp. Hop Topics in Software-Defined Networks, Aug. 2012.
[107] M. Casado et al., "Software-Defined Internet Architecture: Decoupling Architecture from Infrastructure," Proc. 11th ACM Wksp. Hot Topics in Networks, Oct. 2012.
[108] R. Sherwood et al., "FlowVisor: A Network Virtualization Layer," Open-Flow Switch Consortium, tech. rep, 2009.
[109] D. Drutskoy, E. Keller, and J. Rexford, "Scalable Network Virtualization in Software-Defined Networks," IEEE Internet Comp., vol. 17, no. 2, Mar. 2013, pp. 20–27.
[110] T. Wood et al., "Toward a Software-Based Network: Integrating Software Defined Networking and Network Function Virtualization," IEEE Network, vol. 29, no. 3, May 2015, pp. 36–41.
[111] W. Ding et al., "OpenSCaaS: an Open Service Chain as a Service Plat-form toward the Integration of SDN and NFV," IEEE Network, vol. 29, no. 3, May 2015, pp. 30–35.
[112] J. Matias et al., "Toward an SDN-Enabled NFV Architecture," IEEE Com-mun. Mag., vol. 53, no. 4, Apr. 2015, pp. 187–93.

[113] Open Network Foundation, “ONF Techical Report TR-518: Relationship of SDN and NFV,” Oct. 2015.
[114] ETSI NFV ISG, “NFV-EVE005: SDN Usage in NFV Architectural Frame-work,” Oct. 2015.
[115] Fischer et al., “Virtual Network Embedding: A Survey,” IEEE Commun. Surveys & Tutorials, vol. 15, no. 4, 4th qtr. 2013, pp. 1888–1906
[116] Q. Duan, Y. Yan, and A. V. Vasilakos, “A Survey on Service-Oriented Network Virtualization toward Convergence of Networking and Cloud Computing,” IEEE Trans. Network Service Mgmt., vol. 9, no. 4, Dec. 2012, pp. 373–92