Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices

Article *in* Future Internet · May 2014

DOI: 10.3390/fi6020302

2 authors:

Wolfgang Braun
University of Tuebingen
9 PUBLICATIONS **435** CITATIONS

SEE PROFILE

Michael Menth
University of Tuebingen
291 PUBLICATIONS **3,235** CITATIONS

SEE PROFILE

Future Internet **2014**, *6*, 302-336; doi:10.3390/fi6020302

OPEN ACCESS

future internet

ISSN 1999-5903

www.mdpi.com/journal/futureinternet

Article

Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices

Wolfgang Braun * and Michael Menth

Department of Computer Science, University of Tuebingen, Sand 13, Tuebingen 72076, Germany; E-Mail: menth@uni-tuebingen.de

* Author to whom correspondence should be addressed; E-Mail: wolfgang.braun@uni-tuebingen.de; Tel.: +49-7071-29-70509.

Received: 16 January 2014; in revised form: 12 April 2014 / Accepted: 25 April 2014 / Published: 12 May 2014

Abstract: We explain the notion of software-defined networking (SDN), whose southbound interface may be implemented by the OpenFlow protocol. We describe the operation of OpenFlow and summarize the features of specification versions 1.0–1.4. We give an overview of existing SDN-based applications grouped by topic areas. Finally, we point out architectural design choices for SDN using OpenFlow and discuss their performance implications.

Keywords: software-defined networking; OpenFlow; control plane; network applications

---

1. Introduction

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. There is the hope for less specific and cheaper hardware that can be controlled by software applications through standardized interfaces. Additionally, there is the expectation for more flexibility by dynamically adding new features to the network in the form of networking applications. This concept is known from mobile phone operating systems, such as Apple's iOS and Google's Android, where "apps" can dynamically be added to the system.

In this paper, we introduce the notion of SDN using the definition from the Open Networking Foundation (ONF). OpenFlow is just an option for a control protocol in SDN, but it is the predominant one. As OpenFlow currently evolves, several versions of its specification exist; newer releases are more powerful, as their feature sets support IPv6, Multiprotocol Label Switching (MPLS), rerouting, metering, policing and more scalable control. However, most existing applications are based on version 1.0, whose feature set is rather limited. We will highlight how SDN contributes to network innovation by various networking applications that were implemented and analyzed in recent years. These applications fall in the areas of network management and traffic engineering, security, network virtualization and routing optimization. Several surveys of SDN already exist [1–3]; this contribution differs from them through an analysis of architectural design choices for OpenFlow-based SDN networks. We discuss performance issues of OpenFlow-based SDN and refer to studies regarding these aspects.

Section 2 explains the concept of SDN, and Section 3 gives an overview of the OpenFlow protocol, which is currently the major protocol to control network elements in SDN. Section 4 gives a survey of networking applications and groups them by topic areas. Section 5 discusses architectural design choices for SDN. Performance issues of OpenFlow-based SDN are discussed in Section 6, and Section 7 summarizes OpenFlow-based SDN. Finally, Section 8 concludes this work.

2. Software-Defined Networking

In this section, we give an overview of SDN. We review the interfaces of SDN and discuss potential control protocols for SDN. We delineate SDN from the active and programmable networks, which resemble it in some aspects.

2.1. Definition

We discuss SDN along the definition given by the ONF [4]. It is the most accepted SDN definition worldwide, because most global players in the networking industry and many IT corporations participate in the ONF.

Figure 1. A three-layer software-defined networking (SDN) architecture.

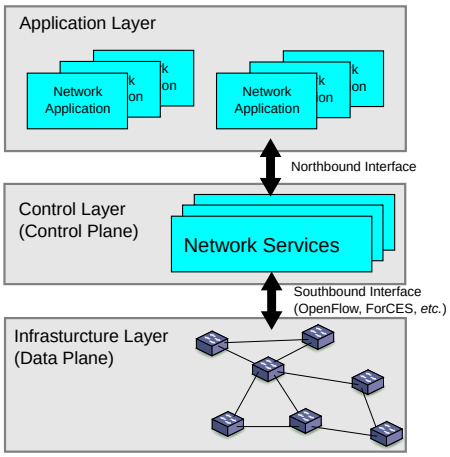


Figure 1 illustrates the SDN framework, which consists of three layers. The lowest layer is the infrastructure layer, also called the data plane. It comprises the forwarding network elements. The responsibilities of the forwarding plane are mainly data forwarding, as well as monitoring local information and gathering statistics.

One layer above, we find the control layer, also called the control plane. It is responsible for programming and managing the forwarding plane. To that end, it makes use of the information provided by the forwarding plane and defines network operation and routing. It comprises one or more software controllers that communicate with the forwarding network elements through standardized interfaces, which are referred to as southbound interfaces. We review protocol options for the southbound interface in Section 2.2. OpenFlow, which is one of the mostly used southbound interfaces, mainly considers switches, whereas other SDN approaches consider other network elements, such as routers. A detailed description of the OpenFlow protocol is given in Section 3.

The application layer contains network applications that can introduce new network features, such as security and manageability, forwarding schemes or assist the control layer in the network configuration. Examples of network applications will be discussed in Section 4. The application layer can receive an abstracted and global view of the network from the controllers and use that information to provide appropriate guidance to the control layer. The interface between the application layer and the control layer is referred to as the northbound interface. For the latter, no standardized API exists today, and in practice, the control software provides its own API to applications. We briefly discuss the northbound interface in Section 2.3.

2.2. Protocol Options for the Southbound Interface

The most common southbound interface is OpenFlow, which is standardized by the Open Networking Foundation (ONF). OpenFlow is a protocol that describes the interaction of one or more control servers with OpenFlow-compliant switches. An OpenFlow controller installs flow table entries in switches, so that these switches can forward traffic according to these entries. Thus, OpenFlow 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 OpenFlow 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 SoftRouter 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 SoftRouter 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.

2.3. 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].

2.4. 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 AN 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.

3. 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 [21] 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.

3.1. Overview

The OpenFlow architecture consists of three basic concepts. (1) The network is built up by OpenFlow-compliant switches that compose the data plane; (2) 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*.

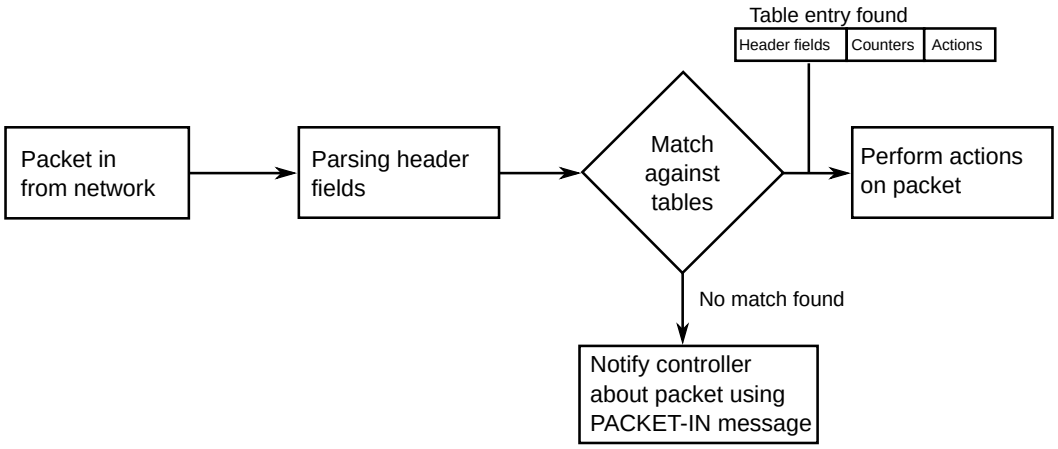
Figure 2. Flow table entry for OpenFlow 1.0.

Header Fields	Counters	Actions

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. 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 controller 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.

Figure 3. Basic packet forwarding with OpenFlow in a switch.



The basic packet forwarding mechanism with OpenFlow is illustrated in Figure 3. 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.

3.2. 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.

3.2.1. 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 3 illustrates the packet handling mechanism of OpenFlow 1.0 as described in Section 3.1. 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.

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.

3.2.2. 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 2 and 3. 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 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. 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.

Figure 4. Flow table entry for OpenFlow 1.1 and later.

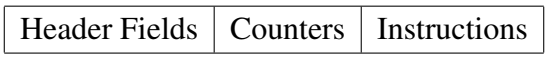


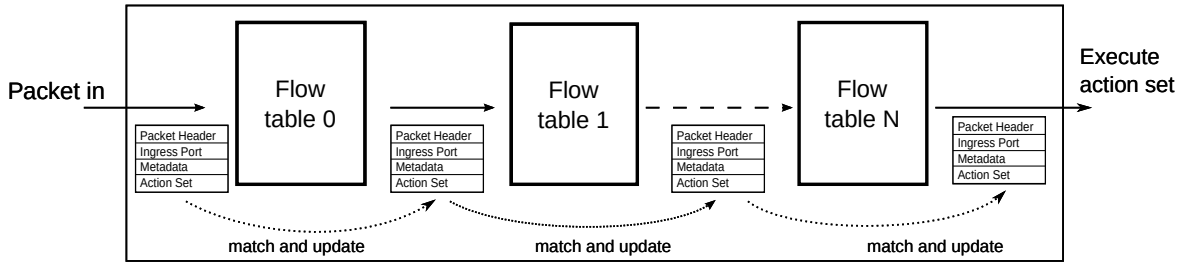
Table 1. List of instructions for OpenFlow 1.1.

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 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 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 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.

Figure 6. Group table entries for OpenFlow 1.1 and later.

Group Identifier	Group Type	Counters	Action Buckets

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 [24] or the IP fast reroute [25] 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.

3.2.3. 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.

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.

3.2.4. 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.

Figure 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 extensions 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.

3.2.5. OpenFlow 1.4

OpenFlow 1.4 [28] 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.

3.2.6. 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 [30] 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 [31] controller was also developed at Stanford University, but is written in Java. The controller is available under a BSD license. The Floodlight controller [32] is a fork of the Beacon controller and is sponsored by Big Switch Networks. It is licensed under the APL. The

Maestro controller [33] 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 [34] controller is written in Java and is based on the Node.JS library. It is available under the MIT license. The Trema [35] 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 [36] 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.

3.2.7. OF-CONFIG

The OF-CONFIG protocol adds configuration and management support to OpenFlow switches. It is also standardized by the ONF. OF-CONFIG provides configuration of various switch parameters that are

not handled by the OpenFlow protocol. This includes features like setting the administrative controllers, configuration of queues and ports, *etc*. The mentioned configuration possibilities are part of OF-CONFIG 1.0 [37], 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.

4. Innovation through SDN-Based Network Applications

SDN has been a vehicle for network innovation in recent years, because it allows for more flexibility and extensibility than existing network architectures. New features or services that are added to an SDN-based network often require only a manageable number of control elements to be upgraded. Upgrades on forwarding elements are often unnecessary depending on the southbound interface. Thus, the separation of the control and data plane allows both planes to evolve more independently of each other compared to other network architectures. Network features and services can be dynamically added in the form of network applications, which facilitates their deployment. Multiple network applications have already been proposed and evaluated by the SDN research community, and we survey them in the following. They can be grouped in the areas of network management and traffic engineering, application server load balancing and network access control, SDN security, network virtualization and inter-domain routing.

4.1. SDN Network Management and Traffic Engineering

The most well-known deployed SDN network in industry is Google's OpenFlow-based WAN, which is used as a production network. As presented at the Open Networking Summit 2012 and in their SDN experience paper [38] at SIGCOMM 2013, Google describes their internal global network that interconnects their data centers worldwide. They discuss the actual implementation of their network and show how traffic engineering can be supported by routing decisions. Their traffic engineering server schedules operations depending on the available bandwidth in the network. They show significant effects in resource utilization that can be achieved with SDN and OpenFlow in the context of data center synchronization and communication. This is possible in the data center environment of Google, but is not generally applicable for other use cases.

SDN eases the introduction of new protocols in the network. In [39], IP multicast is implemented in the control plane using OpenFlow by handling multicast subscribe requests within the control software. The control software installs the forwarding entries in the switches according to the multicast application. This is achieved without modification of the forwarding devices, because their OpenFlow switches supported the required forwarding operations. However, the OpenFlow data plane has to be upgraded if a new protocol requires other operations than those offered by the OpenFlow specification in use. For SDN solutions that support a programmable data plane, e.g., FLARE [40] or programmable networks, the control and data plane can evolve more independently from each other.

Some researchers consider network updates and the consequences of mismanaged updates. Inconsistent routing, e.g., routing loops, can occur when flow tables are updated in the wrong order. In [41], the authors consider consistent network updates by determining the correct switch update order.

First, they introduce powerful abstractions for network updates and define various properties, such as "no security wholes", "connection affinity" and a "loop-free property". Then, the authors verify that their abstractions are valid and lead to the expected network-wide behavior according to the properties. They also discuss the update process on a "per-packet" and "per-flow" basis, as well as the additional state requirements of their proposal. Their work allows network operators to focus on the state before and after a configuration change without worrying about the transition in between.

The creation of user-friendly interfaces for SDNs is discussed in [42]. The authors present a remote network management interface to OpenFlow networks, called OMNI. It eases the management of OpenFlow-based networks, which can be difficult, due to the number of monitoring variables and multiple network configuration options. OMNI monitors and configures the dynamic creation of flows. To that end, it collects data statistics of the network devices and provides an easy to understand network-wide visualization of the current network state.

4.2. Load Balancing for Application Servers

Various SDN-based applications have been proposed for enterprise networks. A common task is to load balance online requests to particular service replicas. This is typically implemented by using a dedicated middlebox that performs the load balancing. In [43], incoming packets are directly forwarded towards service replicas. An OpenFlow switch automatically distributes the traffic to different servers. A simple solution to establish a flow table entry for each client does not scale. Therefore, the authors provide a scalable and efficient load-balancing application. They achieve that by exploiting the wildcard expressions of the flow entries and, thus, do not require external middleboxes for load-balancing purposes.

4.3. Security and Network Access Control

Traditional enterprise network security relies on securing the hosts and the application of middleboxes, such as firewalls, intrusion detection systems and network address translators. However, middleboxes are often placed at the edges of the network, due to the lack of network routing control. In [44], the authors allow middleboxes to be placed freely in the network, e.g., in a virtual machine on any physical host. OpenFlow-based SDN is used to steer traffic directly to the responsible middleboxes. Furthermore, in [45], middlebox traversal depending on network policies is controlled by SDN.

Network function virtualization (NFV) [46] is a highly related topic and is standardized at the European Telecommunications Standards Institute (ETSI). Network functions, which are often provided by specialized hardware, can be virtualized with NFV. Then, virtual network functions can reside as virtual machines on general purpose servers within the network. As with middleboxes, SDN can be used to steer traffic to virtualized middleboxes and network functions. Service function chaining (SFC) is used to combine various network services. SFC is standardized in the IETF [47] and was previously called network service chaining (NSC) in the IETF. In [48], the authors point out future research directions in NSC. As illustrated in [44,45], SDN can achieve service function chaining by steering traffic to middleboxes depending on network policies.

In [49], the authors propose the system "Resonance" that provides dynamic access control policies. Their solution is based on OpenFlow to implement network access control in network elements. Due to the flexibility and the possibility of fine-grained entries, they can achieve reactive and dynamic network access control without the use of specialized middleboxes. They also highlight the disadvantages and limitation of their current VLAN-based access control architecture and show how the SDN approach overcomes those limitations and problems.

A similar topic, the enforcement of network-wide policies, is discussed in [50]. The authors propose the VeriFlow architecture, which can be used to check and enforce network policies and invariants in real time. Such invariants can include checks for forwarding loops, suboptimal routing and access control violations.

Yao *et al.* [51] implement address validation based on SDN so that address spoofing can be efficiently prohibited. The approach is based on selective installation of flow entries that either allow or drop packets. If a packet arrives that does not match any installed entry, the controller is queried. Then, the controller checks if the given address is authorized to send the packets. When spoofing is detected, explicit drop rules are installed in the switches to prevent further actions by a malicious intruder.

Moving target defense prevents attacks that rely on static IP address assignments, such as stealthy scanning and worm propagation. The basic idea of moving target defense is to frequently change IP addresses of hosts over time. In [52], an SDN-based moving target defense method is presented. A host has a persistent IP address and SDN translates this IP address to different IP addresses over time. Thereby, the host operates on the same IP address, while from the outside world, it appears to frequently change its IP address. This method effectively prevents scanning of the host from external, as well as internal attackers.

The detection and defense against Denial of Service (DoS) and distributed DoS (DDoS) attacks have been investigated several times on the basis of SDNs. Chu *et al.* [53] detect DDoS attacks by measuring the frequency of the traffic of a flow. If the flow frequency exceeds a specified threshold, they assume that a DoS attack is happening. In such a case, the controller instructs the switches to drop the packets that belong to the malicious flow to mitigate the effects of the attack. Another attack proposed by Braga *et al.* [54] is based on self-organizing maps to classify traffic patterns. When traffic is classified as malicious, the controller reactively programs the network to drop the malicious traffic.

The control plane in SDN is more centralized, powerful and open than in existing network architectures. Therefore, the control plane is obviously vital and must not be compromised. Therefore, FortNOX is proposed in [55], which is an extension of the NOX OpenFlow controller. It can be used in OpenFlow-based SDN networks to improve the security of critical control components by providing role-based authorization of network applications and security constraint enforcement within the controller. This is achieved by a conflict detection engine that mediates flow rule insertion that can be triggered by various network applications. Flow table entry installations are enforced by "least privilege" to ensure high integrity. Thus, the authors propose a solution that allows for network applications, which may request conflicting flow installations.

4.4. Network Testing, Debugging and Verification

The SDN framework changes network configuration and operation. Traditionally, network operators configure the network using device-centric configurations, which can be hard to manage and maintain for large networks. With SDN, various software modules and network applications are responsible for programming the network automatically. Network operators will use the abstractions and user interfaces of SDN software to specify network-centric configurations and policies. Obviously, it is vital that the software programs are correct and produce valid device-centric configurations that provide the desired networking behavior. However, such complex software is often susceptible to bugs, and network application programmers must be able to detect and remove bugs from the programs.

Many debugging utilities for software programs exist that aid programmers to detect and remove bugs from the code. Inspired by the GNU debugger, *gdb*, the authors of [56] propose a network debugger, called *ndb*. Programmers annotate programs with breakpoints; *gdb* will stop a running process at the breakpoints, and the programmer can inspect the current program state. *ndb* provides the same functionality with respect to the networking domain. Network breakpoints can be defined, and *ndb* generates packet backtraces belonging to the breakpoint. The authors present how *ndb* modifies flow tables in switches to generate packet traces. This allows network application programmers to reconstruct the sequence of events that caused packets to be falsely forwarded.

The OFRewind framework's [57] main goal is to apply network debugging technologies, such as the Simple Network Management Protocol (SNMP), *tcpdump* and the sampling of network data to OpenFlow-based networks. The tools provided by the framework can be used to reproduce software errors, to identify hardware limitations or to determine the location of configuration errors. The tools provide the means to correctly record the traffic that is sent over the network and allows one to reproduce failure scenarios by replaying the recorded data traffic in a specific test environment. The authors describe the challenge for providing such a scalable and correct platform and how their solution leverages the centralized OpenFlow control plane to implement a controlled testing environment.

The authors of [58] propose a solution to test OpenFlow switches. Within each OpenFlow switch, an OpenFlow agent communicates with an OpenFlow controller and accepts programs in the form of flow table entries. There may be various OpenFlow agents in a network, e.g., the switches in the network are provided by different vendors or different OpenFlow versions are installed. The SOFT approach aims to identify inconsistencies between different agents and firmware versions by performing symbolic execution. Then, the results of different OpenFlow agents are crosschecked against each other. The authors were able to identify several inconsistencies between the publicly available reference OpenFlow switch and OpenVSwitch implementations.

In [59], the authors propose NICE, a testing method OpenFlow controller application. The authors performed model checking and concolic execution to test unmodified OpenFlow controller applications for the correct behavior. The authors systematically explored the network behavior and network state under various network event orderings. They solved scalability issues by simulating simplified switches and end hosts. The authors improved their bug detection rate by interleaving network events. The effectiveness of NICE is shown by detecting several bugs in three real network applications.

A systematic workflow for debugging SDNs is proposed in [60]. The goal of the workflow is to detect errant behavior in the network and to locate its cause. Especially, failure location is a difficult and tedious problem, but is required to fix or work around a bug. Therefore, the workflow tries to classify failures to different layers. For example, a tenant isolation breach can originate from the hypervisor caused by a mis-translation from virtual networks towards flow table entries in physical devices. The authors describe the overall picture, the considered layers and illustrate their workflow with examples from real network operation. In addition, the authors discuss network troubleshooting tools for traditional and SDN-based networks. They clarify that their workflow is not only applicable on SDN-based networks, but emphasize the advantages of SDN for network debugging. Finally, the authors describe unresolved issues of network debugging and how the process can be possibly optimized in future work.

4.5. SDN Inter-Domain Routing

Various efforts exist to integrate SDN and OpenFlow in carrier networks that often require network routing protocols. The authors of [61] integrate the routing suite, Quagga, into the NOX OpenFlow controller. This enhanced NOX controller is called QuagFlow. The authors highlight the advantages of reusing existing software rather than rewriting a routing suite in a centralized fashion for OpenFlow. Their main argument is that new software tends to have more bugs than an equivalent software that was already tested and applied in real networks. Their solution is based on virtualized routers that run the Quagga routing suite. The QuagFlow architecture ensures that routing control messages are correctly delivered from OpenFlow switches to virtual Quagga routers and *vice versa*. This avoids inconsistencies between the IP and the OpenFlow network. The RouteFlow architecture [62] is based on QuagFlow. In addition, RouteFlow is transparent with regard to the routing engine, e.g., Quagga can be replaced with any appropriate routing software. RouteFlow allows for flexible mapping and operation between virtual elements and physical switches. For instance, each virtual router may control a single or multiple OpenFlow switches. Both approaches, QuagFlow and RouteFlow, enable SDN to interact with IP networks and allow for incremental migration towards SDN-based networks.

The authors of [63] discuss how SDN can perform inter-domain routing. They provide an inter-domain routing component to an SDN control plane. They implement inter-AS routing in the OpenFlow NOX controller. Finally, concepts of centralized BGP control platforms are re-evaluated again. Routing Control Platforms (RCP) [64] were discussed in the mid-2000s to implement a scalable and extensible control platform for BGP. However, RCP focused mainly on the BGP control plane and its integration in IP networks running Interior Gateway Protocols (IGPs), such as Open Shortest Path First (OSPF) or Intermediate System to Intermediate System (IS-IS). The integration in IP networks is enabled through an "IGP Viewer" component. The authors of [65] discuss the concept of RCP in combination with OpenFlow-based networks. They describe how RCP can be embedded as an SDN-based network application and discuss the advantages of this approach, as well as potential difficulties. A centralized BGP control plane can scale for a high number of exterior BGP sessions, as shown in the past. The authors discuss how a centralized BGP control plane simplifies the edge architecture by replacing expensive IP border routers with simplified SDN switches. The RCP architecture is also believed to provide more stability, security and improved policy management than common BGP architectures,

such as route reflectors and confederations. However, the authors discuss scalability issues with OpenFlow-based RCP that are due to the flow table limitations, as well as the required backup flow installations. The availability of a centralized BGP control plane can also be problematic, and the authors discuss potential solutions based on BGP SHIM to avoid the single point of failure problem.

4.6. SDN-Based Network Virtualization

Multiprotocol Label Switching (MPLS) is a network technology that provides various network features with explicit tunneling. The MPLS tunnels may be used for traffic engineering (known as MPLS-TE), as well as for MPLS-based VPNs (MPLS-VPN). The former is often enabled by the Resource Reservation Protocol (RSVP-TE), and the latter often appears in the context of BGP and MPLS/IP virtualization. In [66], the authors discuss how MPLS-TE and MPLS-VPN solutions can be implemented as plugins to the NOX controller. The authors highlight the advantages of the OpenFlow-based approach that can effectively reuse network functionality from the OpenFlow protocol to implement the complex MPLS-TE and MPLS-VPN functionality with little effort. Generalized MPLS (GMPLS) is an extended variant of the MPLS control plane. It can create tunnels on the optical network layer. The authors of [67] provide a unified OpenFlow/GMPLS control plane that can be used to provide GMPLS-specific features to OpenFlow networks.

Network virtualization can also be achieved with pure SDN-based solutions. They are commonly based on the idea of explicit flow table entry installation based on virtual network definitions. The work in [68] describes a virtual network definition and highlights how the virtual network definitions are converted to OpenFlow entries. The authors describe the SDN control plane decisions and how they verify the isolation properties of virtual networks. Hierarchical policies for network virtualization are discussed in [69]. The authors propose PANE, a system that creates flow-specific entries out of virtual network definitions. PANE exposes an API through the northbound interface that can be used to request virtual networks. The system is able to detect conflicts when new virtual networks are requested. PANE can resolve conflicting virtual networks automatically. Finally, the authors of [70] implement an SDN network virtualization component for Openstack [71]. Openstack is an open-source platform for building and maintaining private and public clouds. Their prototype is called Meridian; it is integrated in Openstack, and provides network-as-a-service features to Openstack. This allows Openstack to provide a virtual network for a cloud that has specific network requirements. For example, a complex cloud application can consist of a firewall, a database server and several web-servers. Meridian allows one to define a virtual network that connects the elements and provide consistent routing for external and internal traffic.

5. Design Choices for OpenFlow-Based SDN

Today, SDN is mostly used for flexible and programmable data centers. There is a need for network virtualization, energy efficiency and dynamic establishment and enforcement of network policies. An important feature is the dynamic creation of virtual networks, commonly referred to as network-as-a-service (NaaS). Even more complex requirements arise in multi-tenant data center environments. SDN can provide these features easily, due to its flexibility and programmability.

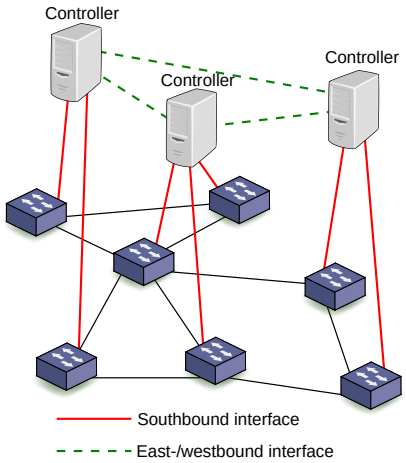
However, SDN is also discussed in a network or Internet service provider (ISP) context. Depending on the use case, the design of SDN architectures varies a lot. In this section, we point out architectural design choices for SDN. We will discuss their implications with regard to performance, reliability and scalability of the control and data plane and refer to research on these topics.

5.1. Control Plane: Physically vs. Logically Centralized

Typically, a centralized control plane is considered for SDN. It provides a global view and knowledge of the network and allows for optimization and intelligent control. It can be implemented in a single server, which is a physically centralized approach. Obviously, a single controller is a single point of failure, as well as a potential bottleneck. Thus, a single control server is most likely not an appropriate solution for networks, due to a lack of reliability and scalability.

As an alternative, a logically centralized control plane may be used to provide more reliability and scalability. It consists of physically distributed control elements that interface with each other through the so-called east- and west-bound interface which is illustrated in Figure 8. Since that distributed control plane interfaces with other layers, like a centralized entity, the data plane and network applications see only a single control element. A challenge for logically centralized control is the consistent and correct network-wide behavior. Another common term for the SDN logically centralized control plane is the "network operating system" (network OS).

Figure 8. Logically centralized control plane.



Several studies investigated the feasibility, scalability and reliability of logically centralized control planes. One research issue is the placement of the distributed control elements inside the network. In [72], the importance of limited latency for control communication in OpenFlow-based networks is highlighted. To meet these latency constraints, they propose a number of required controllers with their position in the network. Hock *et al.* [73] optimize the placement of controllers with regard to latency, as well as controller load, reliability and resilience. Their proposed method can be used to implement a scalable and reliable SDN control plane.

The authors of [74] propose a control plane named HyperFlow that is based on the NOX OpenFlow controller. They discuss the management of network applications and the consistent view of the

framework in detail. For example, when a link fails, one controller notices the failure, but other controllers may not be aware of the link failure. The HyperFlow architecture ensures in such cases that network applications operate on a consistent state of the network, even though control elements may not share identical knowledge about the network. A hierarchical control platform called Kandoo is proposed in [75], which organizes controllers in lower and higher layers. Controllers in lower layers handle local network events and program the local portions of the network under their control. Controllers on higher layers make network-wide decisions. In particular, they instruct and query the local controllers at lower layers.

Another study [76] compares the performance of network applications that run on a distributed control platform. Network applications that are aware of the physical decentralization showed better performance than applications that assume a single network-wide controller.

Jarschel *et al.* [77] model the performance of OpenFlow controllers by a M/M/1 queuing system. They estimate the total sojourn time of a packet in a controller, which is mostly affected by its processing speed. Furthermore, they calculate the packet drop probability of a controller.

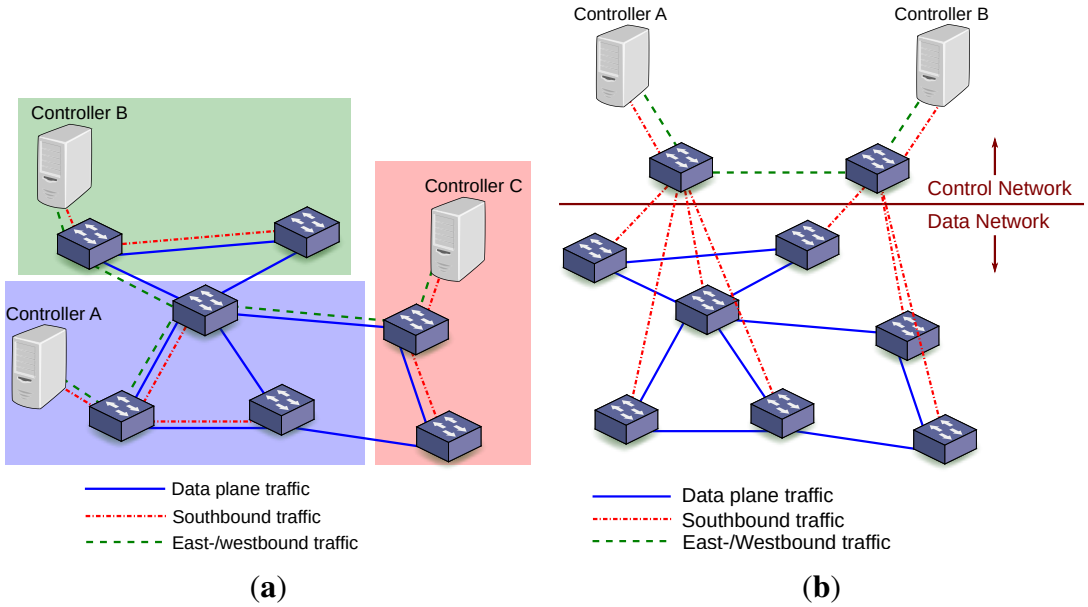
5.2. Control Plane: In-Band vs. Out-of-Band Signaling

In the following, we will discuss the communication between the control elements and the forwarding devices on the data plane. This control channel has to be secure and reliable. In SDN-based data center networks, this control channel is often built as a separate physical network in parallel with the data plane network. Carrier and ISP networks have different requirements. They often span over a country or a continent. Therefore, a separate physical control network might not be cost-efficient or viable at all. Thus, two main design options for the control channel exist: in-band control plane and out-of-band control plane.

With in-band control, control traffic is sent like data traffic over the same infrastructure. This is shown in Figure 9a. This variant does not require an additional physical control network, but has other major disadvantages. Firstly, the control traffic is not separated from the data traffic, which raises security concerns. Failures of the data plane will also affect the control plane. Thus, it is possible that a failure disconnects the switch from its control element, which makes the restoration of the network more complicated.

Out-of-band control requires a separate control network in addition to the data network, as illustrated in Figure 9b. This is a common approach in data centers that are limited in geographical size. In the data center context, the maintenance and cost of an additional control network is usually acceptable. This may be different for wide-ranging networks, such as carrier networks, where a separate control network can be costly with regard to CAPEX and OPEX. The advantages of an out-of-band control plane are that the separation of data and control traffic improves security. Data plane network failures do not affect control traffic, which eases network restoration. Moreover, a separate control plane can be implemented more securely and reliably than the data plane. This ensures the high availability of the control plane and can be crucial for disruption-free network operation.

Figure 9. (**a**) In-band signaling; (**b**) Out-of-band signaling.



A possible approach that combines in-band and out-of-band control planes for carrier networks is based on the control planes in optical networks, such as synchronous optical network (SONET) or synchronous digital hierarchy (SDH) and optical transport networks (OTN). In such networks, an optical supervisory channel (OSC) may be established on a separate wavelength, but on the same fiber over which data traffic is carried. In a similar way, a dedicated optical channel could be allocated for SDN control traffic when an optical layer is available. Furthermore, other lower layer separation techniques may be used to implement separated control and data networks over the same physical infrastructure.

5.3. Management of Flow Entries: Proactive vs. Reactive

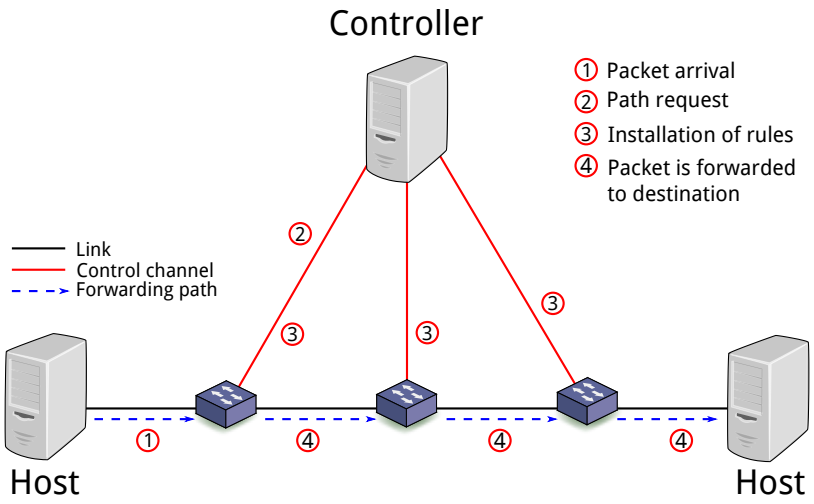
We first explain why flow tables in OpenFlow switches are limited in size, and then, we discuss two approaches for flow table entry management.

In the SDN architecture, the control plane is responsible for the configuration of the forwarding devices. With OpenFlow, the controller installs flow table entries in the forwarding tables of the switches. As discussed in Section 3, an entry consists of match fields, counters and forwarding actions. The OpenFlow match fields are wildcards that match to specific header fields in the packets. Wildcards are typically installed in ternary content-addressable memory (TCAM) to ensure fast packet matching and forwarding. However, TCAM is very expensive, so that it needs to be small; as a consequence, only a moderate number of flow entries can be accommodated in the flow table. In the following, we will describe two flow management approaches: proactive and reactive flow management. Both variants are not mutually exclusive: it is common in OpenFlow networks to install some flows proactively and the remaining flows reactively.

The controller is able to install flow entries permanently and in particular before they are actually needed. This approach is referred to as proactive flow management [78]. However, this approach has a disadvantage: flow tables must hold many entries that might not fit into the expensive TCAM.

To cope with small flow tables, flow entries can also be installed reactively, *i.e.*, installed on demand. This is illustrated in Figure 10. (1) Packets can arrive at a switch where no corresponding rule is installed in the table, and therefore, the switch cannot forward the packet on its own; (2) The switch notifies the controller about the packet; (3) The controller identifies the path for the packet and installs appropriate rules in all switches along the path; (4) Then, the packets of that flow can be forwarded to their destination.

Figure 10. Reactive flow management.



The mechanisms to enable reactive flow management in OpenFlow are based on timeouts. The controller sets an expiry timer that defaults to one second. The switch tracks the duration to the last match for all entries. Unused entries are removed from the switch. When more packets of an expired flow arrive, the controller must be queried for path installation again.

Both flow management approaches have different advantages and disadvantages. The reactive approach holds only the recently used flow entries in the table. On the one hand, this allows one to cope with small forwarding tables. On the other hand, controller interaction is required if packets of a flow arrive in a switch that has no appropriate entry in the flow table. After some time, the correct entry will be eventually installed in the switch, so that packets can be forwarded. The resulting delay depends on the control channel and the current load of the controller. Thus, reactive flow management reduces the state in the switches and relaxes the need for large flow tables, but it increases the delay and reliability requirements of the control channel and control plane software. Especially, failures of the control channel or the controller will have significant impact on the network performance if flow entries cannot be installed in a timely manner.

With a proactive flow management approach, all required flow entries are installed in the switches by the controller. Depending on the use case and network, a huge amount of flows must be installed in switches, e.g., BGP routing tables can contain hundreds of thousands IP prefixes. This may require switch memory hierarchy optimizations that are not needed with reactive flow management. While proactive flow management increases state requirements in switches, it relaxes the requirements on the control plane and software controller performance and is more robust against network failures in the

control plane. That means that if the controller is overloaded or the communication channel fails, the data plane is still fully functional.

The problem with limited TCAM and proactive flow management is a special concern in carrier networks. They mostly use the Border Gateway Protocol (BGP), which has significantly high state requirements [79]. Thus, analyses of the feasibility and practicability of SDN in that context with regard to different flow management approaches and state-heavy protocols are relevant, and viable solutions are needed.

The authors of [80] use a reactive approach to SDN and BGP. They leverage the knowledge of the traffic distribution that they have measured on real data to offload flows with a high traffic portion, called heavy hitters, to the forwarding plane. Entries for flows with low packet frequency are not held in the flow table when space is not available. Those flow entries are kept in the controller and packets belonging to low-frequency flows are sent from the switch to the controller, which has the complete forwarding knowledge of all flows. The proposed SDN software router is presented in [81]. Other approaches try to reduce the flow table size. For example, source routing techniques can be leveraged to significantly reduce the number of flow table entries, as shown by Soliman *et al.* [82]. In their source routing method, the ingress router encodes the path in the form of interface numbers in the packet header. The routers on the path forward the packet according to the interface number of the path in the header. Since packets of source-routed flows contain the path in their headers, the switches on the path do not require a flow table entry for them. However, their method requires some minor changes to the OpenFlow protocol to support the source routing. Source routing methods for SDN are also discussed in the IRTF [83].

The forwarding state for BGP is also a concern in traditional networks without the challenging limitations of forwarding tables. Various approaches to solve this problem exist today. The authors of [84] try to improve the scaling of IP routers using tunneling and virtual prefixes for global routing. Virtual prefixes are selected in such a way that prefixes are aggregated efficiently. Their solution requires a mapping system to employ the virtual prefixes. Ballani *et al.* [85] have a similar solution, where they remove parts of the global routing table with virtual prefixes. They show that this technique can reduce the load on BGP routers. The idea of virtual prefixes is currently standardized in the IETF [86]. Distributed hash tables can also be used to effectively reduce the size of BGP tables, as shown in [87]. Other methodologies are based on the idea of efficient compression of the forwarding information state, as shown in [88]. The authors use compression algorithms for IP prefix trees in such a way that lookups and updates of the FIB can be done in a timely manner. However, match fields in OpenFlow may contain wildcards and, therefore, have more freedom with regard to aggregation than prefixes in IP routing tables. Appropriate methodologies must be developed and tested to reduce the state in OpenFlow switches.

5.4. Data Plane: Resilience

Link or node failures are common in networking, and routing protocols are responsible for the recovery from network failures. Failure recovery is mainly classified by restoration and protection; the latter is often referred to as fast reroute. Restoration is provided by almost all network protocols, but generally requires some time until the network has recovered. Protection schemes provide backup paths that locally bypass failed network elements to minimize the packet loss and, in general, have to be

activated within 50 ms. Thus, protection schemes are highly desirable. A variety of protection schemes exist on various network layers: optical networks often rely on ring protection; IP fast reroute [25] is usually based on shortest paths and encapsulation, and even protection mechanisms for transport layer protocols, e.g., multipath TCP [89], exist.

However, protection schemes always require additional resources to be allocated in the network. Optical networks must reserve additional bandwidth on separate wavelengths, IP fast reroute requires additional forwarding entries in the devices, *etc*. Most importantly, additional forwarding entries can be crucial for OpenFlow networks, due to the fact that forwarding tables have a limited size. Thus, it is important to understand the restoration process in OpenFlow networks, as well as the options and advantages of protection schemes for OpenFlow.

When an OpenFlow switch detects that a link has failed, it notifies its controller about the failure. The controller then takes action to reroute the affected traffic by installing different forwarding entries in appropriate nodes, so that the failure is bypassed. This has two major effects: (1) the restoration is delayed due to the necessary controller interaction and round trip time; and (2) causes additional load on the control plane. Since this may happen simultaneously with multiple flows, this may cause overload on the controller and heavily affect the network performance, especially with reactive flow management. The authors of [90] have analyzed OpenFlow in carrier-grade networks and also investigated the restoration process with OpenFlow. In their test setup, they measured a restoration time between 80 and 130 ms. They also emphasize that the restoration time will be a magnitude higher for large networks, and thus, protection schemes are required that mitigate the effect of the separated control and data plane.

As an alternative, data plane resilience may be achieved by protection. A protection scheme implies that a switch can forward traffic affected by a failure over an alternative local interface. This minimizes service interruption and makes longer reaction times by the controller acceptable. Section 3.2.2 describes how such backup paths can be pre-established in OpenFlow networks. Backup paths require at least OpenFlow 1.1 and have to be installed in the OpenFlow group table. Each flow having a backup path must refer to a group table entry with the group type "fast-failover". The actions for the primary path are installed in the first action bucket of the group entry. The next action bucket describes the actions for the first backup path. Additional backup paths can also be installed in consecutive action buckets of the group entry. The switch will execute the first live action bucket, *i.e.*, in the failure-free case, the primary path, and in the failure case, the first live backup path.

In [91], two main contributions are made toward the protection for OpenFlow networks. The first contribution handles the liveness of ports, actions and action buckets. The OpenFlow standard states that the liveness of ports and links can be managed outside of the OpenFlow protocol. The authors use Bidirectional Forwarding Detection (BFD) to implement their protection mechanism. A BFD component is added to the OpenFlow switch, and it detects link failures in less than 50 ms and disables the port. Then, the switch activates the backup path defined in the group table entry. The second contribution of their work is a protection scheme based on MPLS fast reroute [24]. Appropriate MPLS labels identify backup paths and are pre-established by the controller. The failure detecting switch ensures that the backup path label is applied to the MPLS label stack and forwards the affected packets along the path; the switches along the backup path require additional flow table entries that match on the backup path labels.

The previously discussed work [91] shows that protection schemes can be a critical design choice for large OpenFlow networks, due to the required additional flow entries. It is important to evaluate state requirements for protection schemes and carefully consider whether protection is applicable to a network of a certain size. Therefore, resilience in OpenFlow-based SDN can be a crucial factor that either significantly improves network performance in failure scenarios or causes further scalability concerns, in particular in failure-free scenarios.

6. Performance of OpenFlow-Based SDNs

In the following, we discuss the scalability and feasibility aspects of OpenFlow-based networks and OpenFlow-compliant switches. Several works discuss data plane performance issues that arise with fine-grained flow control, limited flow table sizes and limited sizes of OpenFlow counters that can be used to implement various operations and management (OAM) features.

To improve scalability issues with flow counters, Mogul and Congdon [92] increased the performance by proposing alternate switch designs that implement software-defined counters on a generic CPU within the switch. As a side effect, this design allows flow tables with more flow table entries. In reference [93], the authors improved the performance of switches by differentiating between coarse-grained and fine-grained flows and by managing both flow classes independently. They showed a significant increase in switching performance and the possibility of handling more flows within a switch, while being fully compliant with OpenFlow 1.0. Using the CPU as a co-processing unit in switches can improve the flexibility and switching performance of OpenFlow devices [94]. The results of this work are validated using a prototype implementation, where the switch design is modified by adding a general purpose CPU interconnected to an application-specific integrated circuit (ASIC) with an internal high-bandwidth link. Source routing techniques are applied to SDN networks in [95]. The forwarding path is encoded in the packets to significantly reduce the number of required flow table entries. Finally, the authors of [96] discuss the feasibility of OpenFlow on Linux-based systems and analyze the influence of small packets on the forwarding performance.

Other approaches aim to minimize the number of entries in TCAM by aggregating the matches for entries with the same action. This is a common procedure on prefix-based matches, which can be optimally solved in polynomial time, as shown in [97]. The compression of wildcard entries is more complex. The "TCAM Razor", proposed in [98], is a heuristic for the compression of wildcard matches. A set of TCAM entries is converted into a smaller semantically equivalent set of entries. In this work, they also discuss the compression of multi-dimensional entries, as found in ACLs and OpenFlow-based match rules. They show significant compression ratios compared to prefix-based aggregation and evaluate the runtime. The authors emphasize that their proposal does not require any modifications to existing packet classification systems.

Another method to compress wildcard matches is presented in [99]. The authors provide a compression heuristic that has a polynomial run time with regard to the amount of TCAM rules. The method is based on bit weaving, *i.e.*, two rules are aggregated if they have the same action and differ in only one bit. The authors also consider incremental updates in their method and compare their solution

with other compression methods, such as the TCAM Razor. The bit-weaving method improves the runtime significantly, but decreases the compression efficiency.

A different approach for wildcard aggregation with TCAM is proposed in [100]. The authors perform rule-set minimization based on two-level logic minimization. The algorithm is based on exact logic function minimization algorithms, which are applied in the process of creating large integrated circuits. The authors provide an exact solution for the minimization problem, as well as a number of heuristics and show higher compression ratios with their methods compared to others. However, the proposed method has scalability problems with the number of TCAM entries that must be aggregated.

The "Doing It Fast And Easy" (DIFANE) [101] approach examines the scalability issues that arise with OpenFlow in large networks and with many fine-grained flow entries. Scalability concerns can be classified by (1) the number of flow entries inside the switches and (2) the load on the controller that is caused when many flows have to be installed simultaneously. DIFANE installs all forwarding information in the fast path, *i.e.*, in TCAM, in a few selected switches, called "authority switches". This is achieved by wildcard match fields and the intelligent distribution of flow table entries on the authority switches. The other switches forward traffic that cannot be resolved by their own flow table towards authority switches. The authors show the applicability of DIFANE in large networks by evaluating their proposal on various metrics, such as the number of required TCAM entries, packet loss caused by failures, *etc*.

7. Discussion of OpenFlow-Based SDN

In this work, we have shown that OpenFlow-based SDN provides advantages, but also creates technical challenges. SDN provides more flexibility and programmability than conventional networking architectures, so that new features and network applications can be added to networks more easily. Researchers have analyzed OpenFlow-based SDN in various networking areas and showed improvements, even for complex networking tasks and features. We presented network applications in the fields of network security, traffic engineering, network management, middlebox networking, virtualization and inter-domain routing in Section 4.

All those network applications are facilitated by the SDN control plane, which is discussed in Section 5.1. It provides a consistent and global view of the network, which enables network control algorithms to be written in a simplified fashion. The control plane is software-based and can run on powerful server hardware with lots of memory and modern CPUs, which enables computation-intensive route calculations in practice, such as traffic engineering and route optimization. Moreover, the number of control elements in OpenFlow-based SDN is usually smaller than the number of forwarding elements, which facilitates upgrades. However, OpenFlow-based SDN possibly requires data plane updates for new protocols, if the set of operations offered by an OpenFlow specification is insufficient. The OpenFlow protocol provides more flexibility since OpenFlow 1.2, because new match fields can be defined using the OpenFlow Extensible Match (OXM), as presented in Section 3.2.3. Other southbound interfaces are more flexible than OpenFlow, e.g., ForCES offers a more programmable data plane.

As the control server sets the flow table entries in OpenFlow switches via the OpenFlow protocol, the frequency of configuration requests by OpenFlow switches may drive the control server to its limit,

so that a bottleneck may occur. Such situations may happen in the presence of a large number of fine-grained flow table entries that occur in large networks with many switches and end-hosts or in the presence of network failures or updates when many flows need to be simultaneously rerouted. We presented various works that discuss and improve the control plane scalability of OpenFlow by leveraging hierarchical control structures or intelligent flow rule distribution.

Software-based network applications enable network innovation. Nonetheless, complex software often contains many bugs, which also holds for network control algorithms. Failures in network control software can cause failures and outages in a network. Therefore, the correctness of network applications, which are applied to critical infrastructure, have to be correct and well tested before they are deployed. We discussed several approaches for the testing and verification of SDN software in Section 4.4.

In addition, the OpenFlow data plane faces scalability issues. OpenFlow switches support wildcard matches that are used to classify packets to flows. Thus, fast packet forwarding for OpenFlow requires special hardware: TCAM is used in switches for the fast lookup of wildcard matches. TCAM size is often very limited, due to the high cost. Serious scalability problems can occur when many flow table entries are needed. The number of necessary flow table entries increases for larger networks, excessive use of fine-grained matches and data plane resilience methods. We discussed several proposals in Section 6 that improve the performance of the OpenFlow data plane, especially with regard to the limited number of flow table entries. This discussion shows that it is recommendable to investigate prior to deployment whether the advantages of OpenFlow-based SDN solutions outweigh its scalability concerns, which both depend on the specific use case.

8. Conclusions

In this paper, we have reviewed the concept of software-defined networking (SDN) and explained the features of the different specifications of the OpenFlow protocol, which is a means to implement SDN-based networks. We illustrated various SDN applications in different areas that improve network management and operation, enterprise networks and middlebox routing, security issues and inter-domain routing. A major contribution of this paper is the consideration of design choices. The controller may be physically or logically centralized; in-band or out-of-band control may be used,; flow table entries may be configured in the switches in a reactive or proactive way; and data plane resilience may be provided. We discussed the advantages and disadvantages of these approaches and summarized the challenges and advantages of OpenFlow-based SDN that have been proposed recently.

Acknowledgments

This work has been supported by the German Federal Ministry of Education and Research (BMBF) under support code 16BP12307 (EUREKA-Project SASER). The authors alone are responsible for the content of the paper.

Conflicts of Interest

The authors declare no conflict of interest.

References

1. Lara, A.; Kolasani, A.; Ramamurthy, B. Network Innovation Using OpenFlow: A Survey. *IEEE Commun. Surv. Tutor.* **2013**, *16*, 1–20.
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.
4. Open Networking Foundation. Available online: https://www.opennetworking.org/ (accessed on 22 July 2013).
5. 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).
6. 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).
7. Hares, S. Analysis of Comparisons between OpenFlow and ForCES. Internet Draft (Informational), 2012. Available online: https://datatracker.ietf.org/doc/draft-hares-forces-vs-openflow/ (accessed on 17 February 2014).
8. 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.
9. 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.
10. 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).
11. 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).
12. 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.
13. 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.

14. 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.
15. 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.
16. 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.
17. Tennenhouse, D.L.; Wetherall, D.J. Towards an Active Network Architecture. *ACM SIGCOMM Comput. Commun. Rev.* **1996**, *26*, 5–18.
18. 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.
19. Feamster, N.; Rexford, J.; Zegura, E. The Road to SDN: An Intellectual History of Programmable Networks. *ACM Queue* **2013**, *12*, 20–40.
20. 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.
21. 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.
22. 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).
23. 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).
24. 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).
25. 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).
26. 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).
27. 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).
28. 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).

29. 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.
30. NOXrepo.org. Available online: http://www.noxrepo.org (accessed on 16 November 2013).
31. 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; pp. 13–18.
32. Project Floodlight: Open Source Software for Building Software-Defined Networks. Available online: http://www.projectfloodlight.org/floodlight/ (accessed on 16 November 2013).
33. 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.
34. NodeFLow OpenFlow Controller. Available online: https://github.com/gaberger/NodeFLow (accessed on 16 November 2013).
35. Trema: Full-Stack OpenFlow Framework in Ruby and C. Available online: http://trema.github.io/trema/ (accessed on 16 November 2013).
36. OpenDaylight. Available online: http://www.opendaylight.org/ (accessed on 22 February 2014).
37. 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).
38. 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.
39. 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; pp. 60–67.
40. 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).
41. 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.
42. 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.
43. 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.

44. 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.
45. 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.
46. 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).
47. Boucadair, M.; Jacquenet, C. Service Function Chaining: Framework & Architecture. Internet Draft (Intended Status: Standards Track), 2014. Available online: https://tools.ietf.org/search/draft-boucadair-sfc-framework-02 (accessed on 20 February 2014).
48. 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.
49. 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.
50. 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.
51. 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.
52. 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.
53. 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.
54. 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.
55. 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.
56. 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.

57. 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.
58. 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.
59. Canini, M.; Venzano, D.; Perešíni, P.; Kostić, 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.
60. 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.
61. 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.
62. 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.
63. 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.
64. 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.
65. 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.
66. 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.
67. 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.
68. 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.

69. 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.
70. 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.
71. The OpenStack Foundatation. 2013. Available online: http://www.openstack.org/ (accessed on 22 July 2013).
72. 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.
73. 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.
74. 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.
75. 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.
76. 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.
77. 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.
78. 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.
79. CIDR REPORT. Available online: http://www.cidr-report.org/as2.0/ 2013. (accessed on 22 July 2013).
80. 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.
81. 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.
82. 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.
83. 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).

84. 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.
85. 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.
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).
87. Masuda, A.; Pelsser, C.; Shiomoto, K. SpliTable: Toward Routing Scalability through Distributed BGP Routing Tables. *IEICE Trans. Commun.* **2011**, *E94-B*, 64–76.
88. 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.
89. 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).
90. Sharma, S.; Staessens, D.; Colle, D.; Pickavet, M.; Demeester, P. OpenFlow: Meeting Carrier-Grade Recovery Requirements. *Comput. Commun.* **2013**, *36*, 656–665.
91. 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.
92. 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.
93. 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.
94. 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 (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. 88–97.

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 .