CHAPTER

SDN Open Source

11

In Chapter 5 we presented an overview of OpenFlow, ostensibly the single most important and fully open component of the SDN framework. The OpenFlow standard is developed by the *Open Networking Foundation* (ONF); however, the ONF does not provide a working implementation of either the switch or the controller. In this chapter we provide an inventory of a number of open source components that are available for use in research, including source code for OpenFlow itself. Some of these are available under licenses that make them suitable for commerical exploitation. Many of these initiatives will speed the standardization and rate of adoption of SDN. This chapter provides a survey of these initiatives and explains their relationship and roles within the broader SDN theme.

We have noted in earlier chapters that accepted use of the term SDN has grown well beyond its original scope. Indeed, the definition of SDN has blurred into most of the current work on network virtualization. Much of this work, though, particularly in Open SDN, has occurred in the academic world. As such, there is a significant library of open source available for those who want to experiment or build commercial products in the areas of SDN and network virtualization. Beyond academic contributions, many companies are contributing open source to SDN projects. Some examples that were mentioned in earlier chapters are Big Switch's Indigo, VMware's *Open vSwitch* (OVS), and the many companies cited in Section 9.9.2 contributing to OpenDaylight. This survey of SDN-related open source projects was greatly aided by the compilations already performed by SDN Central [3] and others [16,12].

In Figure 11.1 we map this wide selection of open source to the various network components. The major components, from the bottom up, are:

- OpenFlow device
- OpenFlow controller
- Applications
- Network virtualization
- Test and simulation

The last category, test and simulation, is shown to the side of the diagram because it does not fit neatly into a bottom-to-top hierarchy. For each of these categories we show in the callout ovals the notable available open source that contributes to that category. In the sections that follow we provide an inventory of this software. Some of these have surfaced as the basis for commercial deployments of network virtualization and SDN, and we explore that open source product in greater depth. As we move through the different network components in Sections 11.4–11.10, we encourage the reader to look back at Figure 11.1 to place those open source projects in the context of the SDN ecosystem.

The manner in which open source software may be commercially exploited is dependent on the nature of the open source license under which it is provided, and these licenses are by no means created

Software Defined Networks. http://dx.doi.org/10.1016/B978-0-12-416675-2.00011-5
© 2014 Elsevier Inc. All rights reserved.

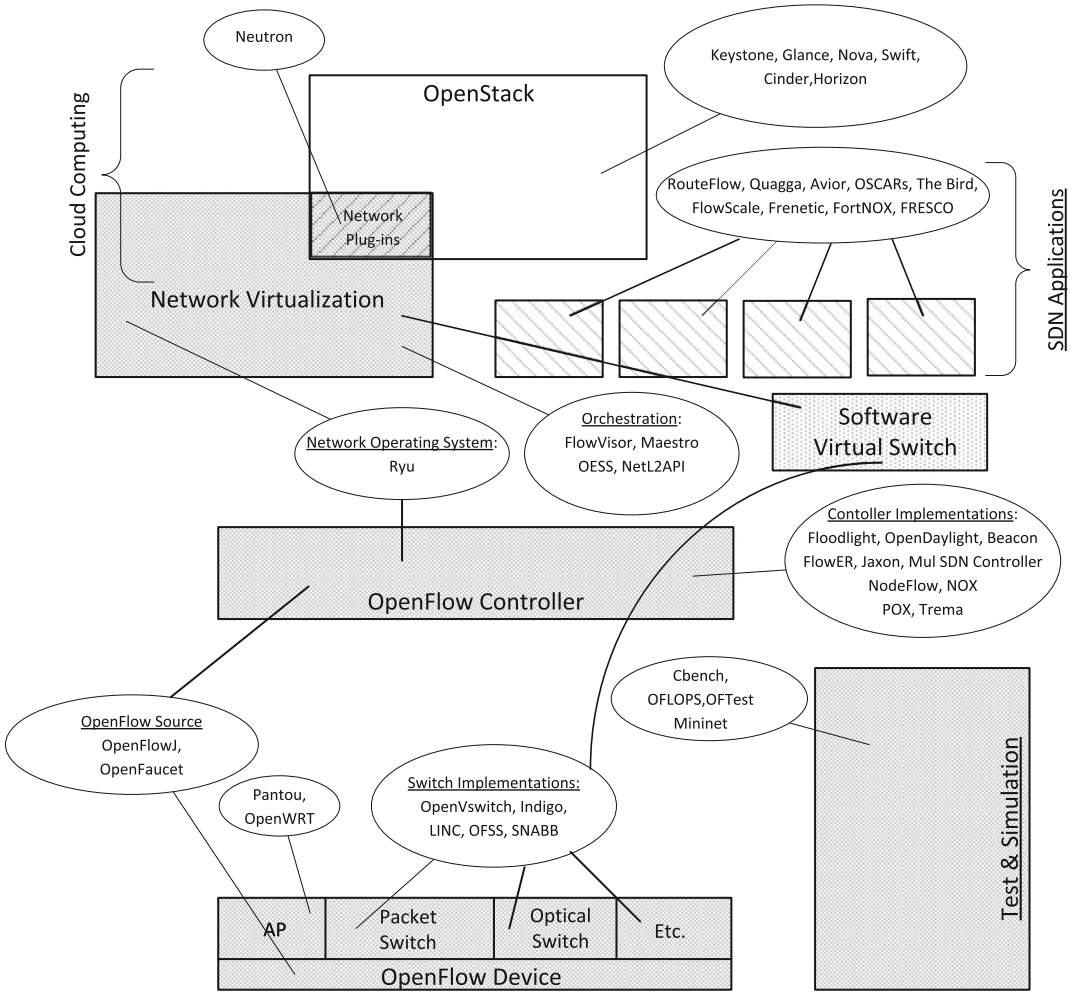


FIGURE 11.1

SDN open source landscape.

equal. So, before discussing the software modules themselves, let's discuss the prevalent types of open source licenses in use today.

11.1 Chapter-Specific Terminology

In this chapter we introduce a term commonly used by the open source development community: *forking*. If we view a source code base as the trunk of a tree, there are development branches in different locations

around the world where innovation and experimentation occur. In many cases, successful and important innovations are contributed back to the evolving master copy of the code and appear in a future release higher up on the trunk of the source code tree, to continue with the analogy. When a new code base is forked from an existing release, it cuts the connection back to the trunk. This means that the continuing innovation on that fork will not be contributed back to the original code base.

11.2 Open Source Licensing Issues

Unless a person has been intimately involved in building a commercial product that incorporates open source software, it is unlikely that he or she understands the complexity of this field of open source licensing. A large number of different open source licenses is extant today. At the time of this writing, *gnu.org* [4] lists a total of 88 free software licenses.

There are many nuances between these free software licenses that may seem arcane to a nonlaw professional. Slight differences about permissions or requirements about using the name of the originator of the license or requirements about complying with U.S. export laws may be some of the only differences between them. Enterprises that offered free source code in the past did so with their organization's name in the license, where the name itself may be the only substantive difference between one company's license and another already existing license. We provide a brief description of the different licenses under which the source mentioned in this chapter has been released, paying particular attention to the business ramifications of choosing to incorporate the software under each of these licenses in a commercial product.

An accepted common denominator of open source is that it is source code provided without charge and that you may use and modify for a defined set of purposes. There is open source available that only allows non-commercial use. We believe that this greatly limits the impact that such software will have, so we do not discuss this further here. Even when commercial use is allowed, however, there are significant differences in the obligations implicit in the use of such software.

Some major categories of open source licensing in widespread commercial use today are the GNU *General Public License* (GPL), *BSD style*, and *Apache*. There are different organizations that opine on whether a given license is truly open source. These groups include the *Free Software Foundation* (FSF) and the *Open Source Initiative*. They do not always agree.

GPL is an extremely popular licensing form. The Linux operating system is distributed under the GPL license. It allows users to copy and modify the software for virtually any purpose. For many companies, though, GPL is too restrictive in one major facet. It incorporates the notion of *copyleft*, whereby if any derivative works are distributed, they must be distributed under the same license. From a practical standpoint, this means that if a commercial company extends a GPL open source code base to provide some new function that it wants to offer to the market, the modifications that are made must be made freely available to the open source community under GPL terms. If the feature in question here embodies core intellectual property that is central to that company's value proposition, this is rarely acceptable. If the function provided by the GPL open source can be used with little added intellectual property, GPL code may be suitable in commercial products. Ensuring that this distinction is kept clear can have massive consequences on the valuation that a startup receives from investors. For this reason, GPL open source is used with caution in industry. Nonetheless, GPL remains the most widely used free software license. There is a GPL V.2 and GPL V.3 that are somewhat more restrictive than the initial version.

The BSD-family licensing model is more permissive. This style of license was first used for the *Berkeley Software Distribution* (BSD), a Unix-like operating system. There have been a number of variants of this original license since its first use; hence we use here the term BSD family of licenses. Unlike GPL code, the user is under no obligation to contribute derivative works back to the open source community. This makes it much less risky to incorporate BSD-licensed open source into commercial products. In some versions of this style of license there is an *advertising clause*. This clause was originally required to acknowledge the use of U.C. Berkeley code in any advertisement of the product. Although this requirement has been eliminated in the latest versions of the BSD license, there are still many free software products licensed with a similar restriction. An example is the widely used OpenSSL encryption library, which requires acknowledgment of the author in product advertising.

The Apache license is another permissive license form for open source. The license was developed by the *Apache Software Foundation* (ASF) and is used for software produced by the ASF as well as some non-ASF entities. Software provided under the Apache license is unrestricted in use except for the requirement that the copyright notice and disclaimer be maintained in the source code itself. Because this restriction has minimal impact on potential commercialization of such software, many commercial entities build products freely incorporating Apache-licensed source code today.

The *Eclipse Public License* (EPL) [7] can be viewed as a compromise between the strong *copyleft* aspects of the GPL and the commercially friendly Apache license. Under certain circumstances, using EPL-licensed source code can result in an obligation to contribute one's own modifications back to the open source community. This is not a universal requirement, though, so developers can often incorporate this software into commercial products without risk of being forced to expose their own intellectual property to the entire open source community. The requirement comes into effect when the modifications are not a separate software module but contain a significant part of the EPL-licensed original and are thus a derivative work. Even when the requirement to release extensions to the original is not in effect, the user is required to grant a license to anyone receiving their modification to any patent the user may hold over those modifications.

The Stanford University *Free and Open Source Software* (FOSS) license [8] is noteworthy in the manner in which it attempts to address the common problem with commercial use of GPL-licensed source code due to its copyleft provision. Stanford's motivation in developing this concept was to avoid the situation where developers would hesitate to develop applications for the Beacon controller (see Section 11.6). Because of the way in which applications interface with the Beacon controller, if it were licensed under GPL without the FOSS exception, such developers would be obliged to contribute their work as GPL open source. This license is specific to the Beacon controller, though the concept can be applied more generally. In essence, Stanford wants to release the Beacon controller code under the terms of GPL. FOSS specifies that a user of the code must continue to treat the Beacon code itself under the terms of GPL, but the code they write as a Beacon application can be released under any of a number of approved FOSS licenses.

These approved licenses are the most common of the open source licenses discussed in this chapter. Notably, an open source application can be released under a license that is more commercially friendly, such as Apache, which does not require that the derivative works on the application code be released to the open source community, thus allowing the developer to guard his newly generated intellectual property.

11.3 Profiles of SDN Open Source Users

When judging the relevance of open source to the SDN needs of one's own organization, the answer will be highly dependent on the role to which the organization aspires on the SDN playing field. To be sure, the early SDN adopters were almost exclusively researchers, many of whom worked in an academic environment. For such researchers, keeping flexibility and the ability to continually tinker with the code is always paramount compared to concerns about support or protection of intellectual property. For this reason, open source OpenFlow controllers and switch code were and continue to be very important in research environments. Similarly, a startup hoping to build a new OpenFlow switch or controller will find it very tempting to leverage the work present in an existing open source implementation, provided that does not inhibit the development and commercial exploitation of the company's *own* intellectual property. If one is a cloud computing operator, tinkering with the OpenFlow controller is far less important than having a reliable support organization that one can call in the event of network failure.

Thus, we propose three broad categories of likely users of SDN-related open source software:

- Research
- Commercial developers
- Network operators

Assuming that a given piece of open source software functionally meets a user's requirements, each of these user classes will have a different overriding concern. A developer in a commercial enterprise will focus on ensuring that he/she does not incorporate any open source that will limit his/her company's ability to exploit its products commercially and ensuring that it will protect its intellectual property. The GPL is often problematic when the open source will be modified, and the modifications must thus be made publicly available under GPL. Generally, researchers have the greatest flexibility and simply desire that the source code be available free of charge and can be modified and distributed for any purpose. Often, though, researchers are also strong proponents of the growth of open source and, thus, may prefer a copyleft license that obliges the users to contribute their modifications back to the open source community. Finally, the network operator will be most concerned about maintaining the network in an operational state and therefore will focus on how the open source product will be supported and how robust it is.

Because of the fundamental noncommercial nature of open source, it will always be a challenge to identify whom to hold responsible when a bug surfaces. This will give rise to companies that will effectively commercialize the open source, where the value they add is not development of new features but to provide well-tested releases and configurations as well as a call desk for support. Some operators will simply fall back on the old habit of purchasing technology from the mainstream NEMs. Whether or not their NEM provider uses open source is almost irrelevant to the operators, since they really are purchasing equipment plus an insurance policy and it matters little where the NEM obtained this technology. On the other hand, whereas a cloud operator may not want to assume responsibility for support of its switching infrastructure, the open source applications, test, or monitoring software might serve as useful starting points for locally supported projects.

In terms of the software projects discussed in this chapter, some examples of how different user types might use some of the software include:

- A hardware switch company that wants to convert a legacy switch into an OpenFlow-enabled switch may use the OpenFlow switch code presented in Section 11.5 as a starting point.

- A company that wants to offer a commercial OpenFlow controller might start its project with one of the OpenFlow controllers we list in Section 11.6, add features to it, and harden the code.
- A cloud operator might use an open source SDN application such as those we describe in Section 11.7 to begin to experiment with OpenFlow technology prior to deciding to invest in this technology.

As we review several different categories of software projects in the following sections, for each we list which of these three classes of users are likely to adopt a given software offering. We now provide an inventory of some of the SDN-specific open source that is used by researchers, developers, and operators. Since the number of these open source projects is quite large, we group them into the basic categories of switches, controllers, orchestration, applications, and simulation and test, as depicted in Figure 11.1. We summarize the different open source contributions in each of these categories in a number of tables in the following sections. Only in an instance where the particular project has played an unusually important role in the evolution of SDN do we describe it in further detail.

11.4 OpenFlow Source Code

As Figure 11.2 shows, both the switch and controller implementations can leverage an implementation of the OpenFlow protocol. There are multiple open source implementations of OpenFlow. We use a format of two paired tables to provide a succinct description of these implementations. In Table 11.1 we provide a basic description of available implementations; in Table 11.2 we list detailed information regarding the provider of the code, the type of license, the source code in which it is written (where relevant), and the likely class(es) of users. We have assigned these target classes of users using a number of criteria, including: (1) the nature of the license, (2) the maturity of the code, and (3) actual users,

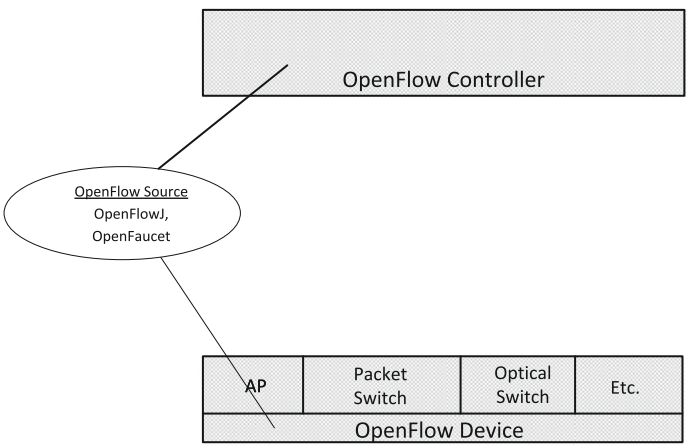


FIGURE 11.2

Applications of OpenFlow source.

Table 11.1 Open Source Implementations of OpenFlow: Description

Product Name	Description
OpenFlow Reference	Reference code base that tracks the specification.
OpenFaucet	Source code implementation of OpenFlow protocol 1.0, used in both controllers and switches.
OpenFlowJ	Source code implementation of OpenFlow protocol. Both Beacon and FlowVisor incorporate this code.

Table 11.2 Open Source Implementations of OpenFlow: Details

Product Name	Source	License	Language	Target User
OpenFlow Reference	Stanford	BSD-style	C	Research
OpenFaucet	Midokura	Apache	Python	Research
OpenFlowJ	Stanford	Apache	Java	Research

when known. OpenFlowJ is notable in that the Beacon controller, itself the fount of a number of other controller implementations, uses the OpenFlowJ code.

We continue to use this format of pairing two tables to provide the description of notable open source projects in each of the major categories we discuss in this chapter. In some cases the license or language may not be known. We indicate this by placing a hyphen in the corresponding table cell. We also mark the table cell with a hyphen for languages in the event that the project uses multiple languages.

11.5 Switch Implementations

In this section we discuss some of the more important available open source SDN switch implementations. The open source SDN switch implementations are summarized in Tables 11.3 and 11.4. Looking back at Figure 11.1 we can see that the open source switch implementations can potentially apply both to OpenFlow hardware device implementations as well as software-based virtual switches. In the following paragraphs we examine the two most widely discussed switch implementations, OVS and Indigo, as well as the only OpenFlow wireless access point code base listed, Pantou.

As of this writing, OVS is the most mature and feature-complete open source implementation of the switch side of an OpenFlow implementation. OVS has been used as the control logic inside genuine hardware switches as well as a virtual switch running within a hypervisor to implement network virtualization. It is in the latter context that the developer of OVS, Nicira, has been for some time shipping OVS with its NVP product. Since the acquisition of Nicira by VMware, the NVP product is now marketed under the name NSX. OVS uses OpenFlow for control of flows and OVSDB for configuration. OVS uses the sFlow protocol for packet sampling and monitoring. The code has been ported to multiple switching chipsets and has thus found a home in several hardware switches as well. Earlier, in Section 6.6.3, we presented an example of OVS being used for SDN via hypervisor-based overlays.

Table 11.3 Open Source OpenFlow Switches: Description

Project	Description
Open vSwitch	Production-quality multilayer virtual switch designed for programmatic network automation using standard management interfaces as well as OpenFlow. Also called OVS.
Indigo	OpenFlow switch implementation designed to run on physical switches and use their ASICs to run OpenFlow at line rates. Note that Switch Light is Big Switch's commercial switch software product and is based on the second generation of the Indigo platform.
LINC	OpenFlow 1.2 Softswitch.
OFSS	OpenFlow 1.1 Softswitch.
of13softswitch	User-space OpenFlow 1.3 Softswitch. Based on Ericsson TrafficLabs 1.0 softswitch code.
Pantou	Wireless router implementation based on OpenWRT with OpenFlow capabilities (see Section 6.4 for more information about OpenWRT.)

Table 11.4 Open Source OpenFlow Switches: Details

Product Name	Source	License	Language	Target User
Open vSwitch	Nicira	Apache 2.0	C	Researchers Developers
Indigo	Big Switch	EPL	—	Developers
LINC	Infoblox	Apache 2	Erlang	Developers
OFSS	Ericsson	BSD-like	C	Research
of13softswitch	CPqD	BSD-like	C	Research
Pantou	Stanford	—	—	Research

Big Switch Networks offers the *Indigo* switch code base under the Eclipse public license. This was a natural move for Big Switch in its desire to create an ecosystem of hardware switches that work well with its commercial OpenFlow controller product, Big Network Controller. Like OVS, Indigo is targeted for use both on physical switches as well as an OpenFlow hypervisor switch for network virtualization environments. In particular, this code can be used to convert a legacy layer two or three switch into an OpenFlow switch. By making this code base available to switch manufacturers, especially smaller manufacturers that may lack the means to develop their own OpenFlow switch-side control code, Big Switch hopes to expand the market for its controller.[1] Because Indigo is integrated with Ethernet switch ASICs, it is possible to switch flows under the OpenFlow paradigm at line rates. Another distinction between OVS and Indigo is that Indigo is implemented specifically to support OpenFlow, whereas OVS can support other control mechanisms such as OVSDB. In addition, OVS has broader support for affiliated network virtualization features than Indigo and includes richer contributions from the open source community. Nevertheless, after OVS, Indigo remains a more likely candidate for an open source starting point for a commercial OpenFlow physical switch than the alternatives discussed in this section. From a feature-set point of view, OVS seems to be a superset of Indigo's features.

[1] Big Switch is now marketing a commercial version of Indigo called Switch Light for this express purpose.

Pantou is actually designed to turn commodity APs into an OpenFlow-enabled AP. This is accomplished by integrating the NOX OpenFlow implementation with OpenWRT. We explained in Section 6.4 that OpenWRT is an open source project that applies the SDN principle of *opening up the device* to a wide selection of low-cost consumer AP hardware. This union of OpenFlow with basic wireless AP functionality available on low-cost hardware is a boon to researchers who want to experiment with the largely virgin field of SDN applied to 802.11 networks.

11.6 Controller Implementations

In this section we discuss some of the more important available open source SDN controller implementations. The open source SDN controller implementations are summarized in Tables 11.5 and 11.6.

The Beacon [8] controller was based on OpenFlowJ, an early open source implementation of OpenFlow in Java. The OpenFlowJ project is hosted at Stanford University. Beacon is a highly influential controller, both for the large amount of early OpenFlow research and development that was done on that controller as well as being the code base from which the Floodlight controller source code was forked.

Beacon is a cross-platform, modular OpenFlow controller [8]. At the time of this writing, Beacon has been successfully deployed in an experimental data center consisting of 20 physical switches and 100 virtual switches. It runs on many different platforms, including high-end Linux servers.

Table 11.5 Open Source Controllers: Description

Project	Description
NOX	The first OpenFlow controller, used as the basis for many subsequent implementations.
POX	Python version of NOX.
Beacon	Java-based OpenFlow controller that is multi-threaded, fast, and modular; uses OpenFlowJ.
Floodlight	Java-based OpenFlow controller derived from Beacon, supported by large developer community.
Ryu	Network controller with APIs for creating applications; integrates with OpenStack. Manages network devices not only through OpenFlow but alternatives such as NETCONF and OF-Config. More than just a controller, it is sometimes referred to as a network operating system.
OpenDaylight	This controller attempts to be the all-purpose SDN controller. It supports numerous southbound interfaces, including OpenFlow, and provides a generic network abstraction for applications in its northbound API.
FlowER	OpenFlow Controller (still in development).
Jaxon	Thin Java Interface to NOX controller.
Mul SDN Controller	OpenFlow Controller.
NodeFlow	OpenFlow Controller.
Trema	OpenFlow Controller, integrated test and debug. Extensible to distributed controllers.

Table 11.6 Open Source Controllers: Details

Product Name	Source	License	Language	Target User
NOX	ICSI	GPL	C++	Research, operators
POX	ICSI	GPL	Python	Research
Beacon	Stanford University	GPLv2 Stanford FOSS Exception v1.0	Java	Research
Floodlight	Big Switch Networks	Apache	Java	Research, developers, operators
Ryu	NTT Communications	Apache 2.0	Python	Research, developers, operators
OpenDaylight	OpenDaylight	EPL	—	Developers
FlowER	Travelping GmbH	MIT License	Erlang	Operators
Jaxon	University of Tsukuba	GPLv3	Java	Research
Mul SDN	Kulcloud	GPLv2	C	Operators
NodeFlow	Gary Berger	—	Javascript	Research
Trema	NEC	GPLv2	Ruby,C	Research

As mentioned earlier, though the core controller code is protected with GPL-like terms, the FOSS license exception allows developers to extend the controller with applications that may be licensed under more commercially favorable terms. Beacon's stability [6] distinguishes it from other controllers used primarily for research purposes. Modules of Beacon can be started, stopped, and even installed while the other components of Beacon continue operating. Evidence of the impact that this code base has had on the OpenFlow controller industry is found in the names of the two most influential open source controllers being used by industry today, *Floodlight* and *Open Daylight*, both echoing the illumination connotation of the name Beacon. In Sections 10.5–10.7, we discussed the use of Beacon, Floodlight, and OpenDaylight when developing SDN applications.

Floodlight was forked from Beacon prior to Beacon being offered under the current GPL/FOSS licensing model [9]. Because Floodlight itself is licensed under the Apache 2.0 license, it is not subject to the copyleft provisions of the Beacon license, and the source is thus more likely to be used by recipients of the code who need to protect the intellectual property of their derivative works on the code base. Floodlight is also integrated with OpenStack (see Section 11.10). A number of commercial OpenFlow controllers are being developed using Floodlight as the starting point. This includes Big Switch's own commercial controller, Big Network Controller. Big Switch's business strategy [5] surrounding the open source controller is complex. The company offers a family of applications that work with the Floodlight controller. Though its commercial controller will not be open source, the company promises to maintain compatibility at the interface level between the commercial and open source versions. This reflects a fairly classical approach to building a software business, which is to donate an open source version of

functionality in order to bootstrap a community of users of the technology, hoping that it gains traction, and then offer a commercial version that can be *upsold* to well-heeled users that require commercial support and extended features. Big Switch joined the *Open Daylight* project, donating its Floodlight controller to the effort in hopes that this action would further propagate Floodlight. These hopes were ultimately not realized, as we explain below. Indeed, as of this writing, Big Switch has tweaked this original business strategy. We discuss the company's *big pivot* in Section 12.9.1.

The OpenDaylight project [10] was formed in early 2013 to provide an open source SDN framework. The platinum-level sponsors included Cisco, Brocade, Ericsson, Citrix, Microsoft, and Big Switch Networks. Unlike the ONF, OpenDaylight only considers OpenFlow to be one of many alternatives to provide software control of networking gear. Part of the OpenDaylight offering includes an open source controller, which is why we include OpenDaylight in this section. As we mentioned, Big Switch initially joined and donated its Floodlight controller to OpenDaylight, anticipating that Floodlight would become the key OpenFlow component of the project. This expectation was quickly shattered as OpenDaylight decided to make the Cisco *Open Networking Environment* (ONE) controller the centerpiece and to merely add Floodlight technology to that core. (Cisco now markets a commercial version of the ONE controller as the *Extensible Network Controller*, or XNC.) Subsequent to this, Big Switch reduced its involvement in the OpenDaylight project and ultimately withdrew completely [1]. The XNC controller [13] may use a variety of methods to control network devices. Whereas OpenFlow is one such method, it supports others, including RESTful APIs and Cisco's *Open Network Environment Platform Kit* (onePK). It is thus noteworthy that both SDN via APIs and Open SDN are present in the same XNC controller. The facts that Big Switch stepped back from its commitment to OpenDaylight, there is a total lack of coordination between the ONF and OpenDaylight, and the platinum sponsors are almost all incumbent network equipment manufacturers have all led to a growing skepticism [11] about the compatibility of OpenDaylight with the ONF. When one considers how alternative definitions of SDN have deviated from the initial Open SDN precepts and that OpenFlow remains tightly coupled to those founding precepts, it is not surprising that such schisms arise.

So, despite the recurring theme of illumination used in the names of the most prominent open source controller implementations, the future of the Beacon, Floodlight, and OpenDaylight controllers remains murky.

11.7 SDN Applications

Chapter 10 was dedicated to an analysis of how Open SDN applications are designed and how they interact with controllers and provided a number of sample designs of different SDN applications. If the reader wants to actually embark on the journey of writing his own SDN application, it is clearly advantageous to have a starting point that can be used as a model for new work. To that end, in this section we present some of the more important available open source SDN applications. The open source SDN applications are summarized in Tables 11.7 and 11.8. The term *applications* by its very nature is general, so it is not possible to list all the possible categories of SDN applications. Nevertheless, four major themes surface that have attracted the most early attention to SDN applications. These are security, routing, network management, and *network functions virtualization* (NFV).

In Table 11.7 there are three examples related to routing: the *BIRD*; *Quagga*, which is a general-purpose open source routing implementation that is suitable for use in SDN environments; and *Routeflow*,

Table 11.7 Open Source SDN Applications: Description

Product Name	Description
Routeflow	Integrates IP routing with OpenFlow controller; based on the earlier Quagflow project (see Section 11.11).
Quagga	Provides IP routing protocols (e.g., BGP, OSPF).
Avior	Management Application for Floodlight.
OSCARS	On-Demand Secure Circuits and Advance Reservation system used for optical channel assignment by SDN; predates SDN.
The BIRD	Provides IP routing protocols (e.g., BGP, OSPF).
FlowScale	Traffic load balancer as a service using OpenFlow.
Frenetic	Provides language to program OpenFlow controller abstracting low-level details related to monitoring, specifying, and updating packet-forwarding policies.
FortNOX	Security Framework, originally coupled with NOX controller, now integrated in SE-Floodlight; extends OpenFlow controller into a security mediation service. Can reconcile new rules against established policies.
FRESCO	Security application integrated with FortNOX; provides security-specific scripting language to rapidly prototype security detection and mitigation modules.

Table 11.8 Open Source SDN Applications: Details

Product Name	Source	License	Language	Target Users
Routeflow	CPqD (Brazil)	—	—	Research, developers
Quagga	Quagga Routing Project	GPL	C	Research, developers
Avior	Marist College	MIT License	Java	Research
OSCARS	Energy Services Network (U.S. Department of Energy)	New BSD	Java	Research
The BIRD	CERN	GPL	—	Research
FlowScale	InCNTRE	Apache 2.0	Java	Research
Frenetic	Princeton University	GPL	Python	Research
FortNOX	SRI International	—	—	Research
FRESCO	SRI International	—	—	Research

which is SDN-specific. In Section 11.11 we describe a specific example of open source being used to implement a complete SDN network where Routeflow and Quagga are used together. *Avior* is a network management application for the Floodlight controller. OpenFlow has great potential to provide the next generation of network security [15]. The two relevant examples in Tables 11.7 and 11.8 are *FortNOX* and *Fresco*. As we described in Section 8.6.1, NFV applications constitute the class of applications that virtualize a network services function that is performed by a standalone appliance in legacy networks.

Examples of such devices are traffic load balancers and intrusion detection systems. FlowScale is an NFV application that implements a traffic load balancer as an OpenFlow controller application.

Some of the open source applications covered in this section are OpenFlow applications, where domain-specific problems are addressed through applications communicating with the OpenFlow controller via its northbound interface. Not all SDN applications are OpenFlow applications, however.

11.8 Orchestration and Network Virtualization

We introduced the concept of orchestration in Section 3.2.3. There are many compute, storage, and networking resources available in the data center. When virtualization is applied to these resources, the number of assignable resources explodes to enormous scale. Assigning these resources to the tenants of the data center in an efficient and manageable fashion is a complex technology unto itself. This technology is called *orchestration*. In this section we list some of the more important open source implementations that provide orchestration functionality. Network orchestration is directly related to network virtualization. The term *orchestration* came into use as it became clear that virtualizing the networking portion of a complex data center, already running its compute and storage components virtually, involved the precise coordination of many independent parts. Unlike legacy networking, where the independent parts function in a truly distributed and independent fashion, network virtualization required that a centralized entity coordinate their activities at a very fine granularity—much like a conductor coordinates the exact timing of the thundering percussion interlude with the bold entry of a brass section in a symphony orchestra. The open source SDN orchestration and network virtualization solutions are summarized in Tables 11.9 and 11.10.

11.9 Simulation, Testing, and Tools

In this section we discuss some of the more important open source implementations related to network simulation, test, and tools pertinent to SDN. The open source SDN solutions available in this area are summarized in Tables 11.11 and 11.12. Some of these projects offer software that may have relevance in nonresearch environments, particularly *Cbench*, *OFLOPS*, and *OFTEST*. Although we have not seen

Table 11.9 Open Source Orchestration Solutions: Description

Product Name	Description
FlowVisor	Creates slices of network resources; delegates control of each slice, that is, allows multiple OpenFlow controllers to share a set of physical switches.
Maestro	Provides interfaces for network control applications to access and modify a network.
OESS	Provides user-controlled VLAN provisioning using OpenFlow switches.
NetL2API	Provides generic API to control layer two switches via vendors' CLIs, not OpenFlow; useable for non-OpenFlow network virtualization.
Neutron	OpenStack's operating system's networking component that supports multiple network plugins, including OpenFlow.

Table 11.10 Open Source Orchestration Solutions: Details

Product Name	Source	License	Language	Likely Users
FlowVisor	ON.LAB	—	Java	Research
Maestro	Rice University	Lesser GPL	Java	Research
OESS	Internet2 Indiana University	Apache 2.0	—	Research
NetL2API	Locaweb	Apache	Python	Research, developers, operators
Neutron	OpenStack Foundation	Apache	—	Operators

Table 11.11 Open Source Test and Simulation: Description

Product Name	Description
Cbench	OpenFlow Controller benchmarker. Emulates a variable number of switches, sends PACKET_IN messages to controller under test from the switches, and observes responses from controller.
OFLOPS	OpenFlow switch benchmarker. A standalone controller that sends and receives messages to/from an OpenFlow switch to characterize its performance and observes responses from the switch.
MiniNet	Simulates large networks of switches and hosts. Not SDN-specific, but widely used by SDN researchers who simulate OpenFlow switches and produce traffic for OpenFlow controllers.
OFTest	Tests switch compliance with OpenFlow protocol versions up to 1.2.

Table 11.12 Open Source Test and Simulation: Details

Product Name	Source	License	Language	Likely Users
Cbench	Stanford University	GPL V.2	C	Research, developers, operators
OFLOPS	Stanford University	Stanford License	C	Research, developers, operators
MiniNet	Stanford University	Stanford License	Python	Research
OFTest	Big Switch Networks	Stanford License	Python	Research, developers, operators

Mininet used in a commercial setting, this tool has seen wide use by SDN researchers to emulate the large networks of switches and hosts and the traffic that can cause a controller, such as an OpenFlow controller, to create, modify, and tear down large numbers of flows.

11.10 OpenStack

We introduced the OpenStack Foundation in Section 9.9.3. The OpenStack open source is a broad open source platform for cloud computing, released under the Apache license. We show the role of OpenStack as well as its components in Figure 11.3. OpenStack provides virtualization of the three main components of the data center: compute, storage, and networking. The compute function is called *Nova*. Nova works with the major available hypervisors to manage pools of virtual machines. Examples of the hypervisors that can be used with OpenStack include KVM, XenServer, and VMware, among others. The storage functions are *Swift* and *Cinder*. Swift provides redundant storage such that storage servers can be replicated or restored at will, with minimum dependency on the commodity storage drives that provide the actual physical storage. Cinder provides OpenStack compute instances with access to file and block storage devices. This access can be used with most of the prevalent storage platforms found in

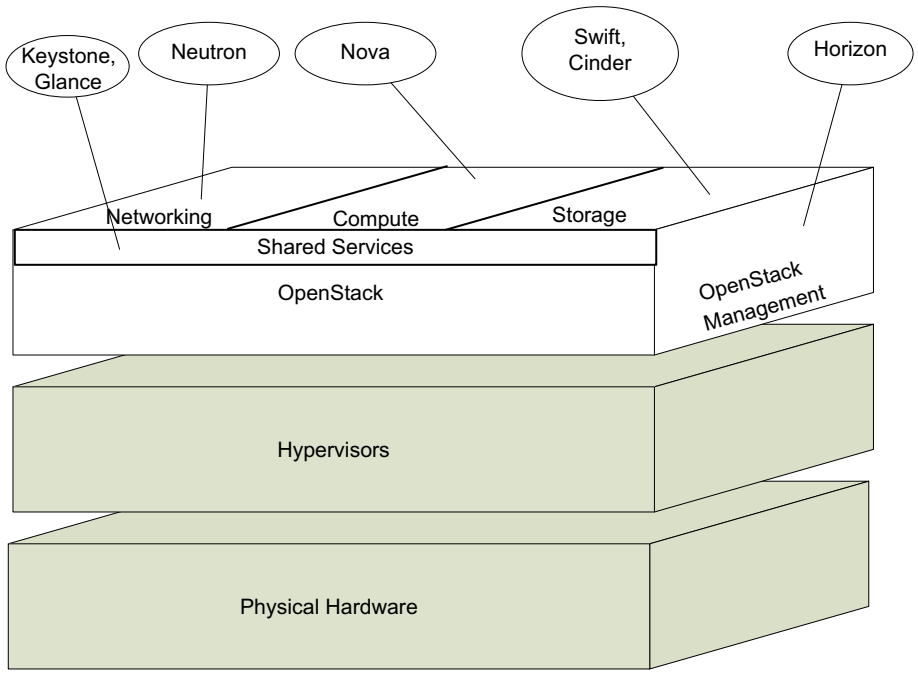


FIGURE 11.3

Openstack components and roles.

cloud computing today. *Horizon* provides a dashboard to access, provision, and manage the cloud-based resources in an OpenStack environment. There are two shared services, *Keystone* and *Glance*. Keystone provides a user authentication service and can integrate with existing identity services such as LDAP. Glance provides the ability to copy and save a server image such that it can be used to replicate compute or storage servers as services are expanded. It also provides a fundamental backup capability for these images. The network virtualization component of OpenStack is provided by *Neutron* and, thus, is the component most relevant to our discussion of SDN. Note that Neutron was formerly known as Quantum.

Architecturally, Neutron's role in OpenStack is embodied as a number of plugins that provide the interface between the network and the balance of the OpenStack cloud computing components. Though OpenStack is not limited to using Open SDN as its network interface, Open SDN is included as one of the networking options. Looking back at Figure 11.1 we see that the Neutron plugin can interface to the northbound API of an OpenFlow controller. Neutron can thus provide the network abstraction layer for an OpenFlow-enabled network. Just as OpenFlow can work with many different network-control applications via a northbound API, so can OpenStack's Neutron have many different kinds of network plugins. Thus, OpenStack and OpenFlow may be married to provide a holistic network solution for cloud computing, but neither is exclusively tied to the other. As shown in Figure 11.4, OpenStack can use Neutron plugins to control legacy network devices, an OpenFlow controller that controls OpenFlow-enabled physical switches, or virtual switches such as OVS [2]. The implementation

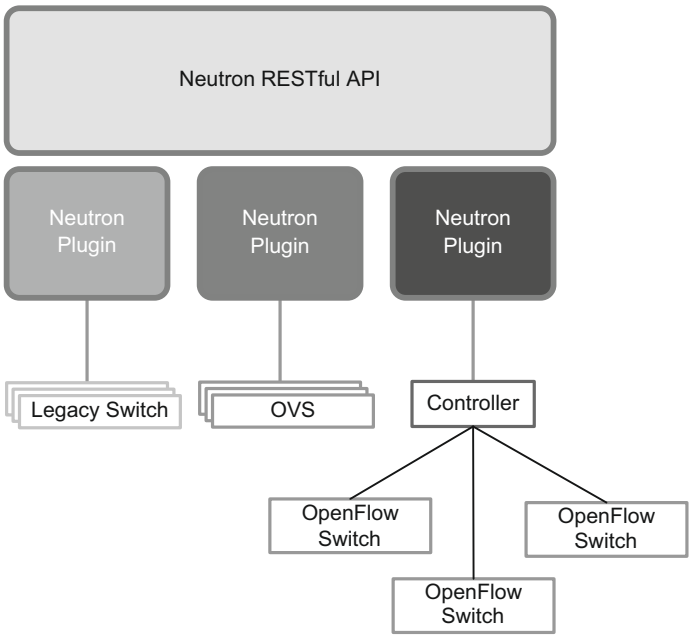


FIGURE 11.4

Openstack plugins.

of the OVS interface, for example, consists of the plugin itself, which supports the standard Neutron northbound APIs, and an agent that resides on the Nova compute nodes in the OpenStack architecture. An OVS instance runs locally on that compute node and is controlled via that agent.

OpenStack exposes an abstraction of a *virtual pool of networks*. This is closely related to the virtual networks abstraction we discussed in relation to the SDN via overlays solution. So, for example, with OpenStack one could create a network and use that for one specific tenant, which maps quite well to the SDN via overlays solutions concept. OpenStack has plugins for many of the existing Overlay solutions.

11.11 Example: Applying SDN Open Source

The *RouteFlow* project [14] provides an excellent use case for open source SDN code. The RouteFlow project has developed a complete SDN test network using only open source software. The goals of the RouteFlow project are to demonstrate:

- A potential migration path from traditional layer three networks to OpenFlow-based layer three networks
- A pure open source framework supporting different aspects of network virtualization
- IP Routing-as-a-Service
- Legacy routing equipment interoperating with simplified intra- and interdomain routing implementations

Figure 11.5 depicts the network topology the project uses to demonstrate their proposed migration path from networks of legacy routers to OpenFlow. This is particularly important because it demonstrates a practical method of integrating the complex world of routing protocols with the OpenFlow paradigm.

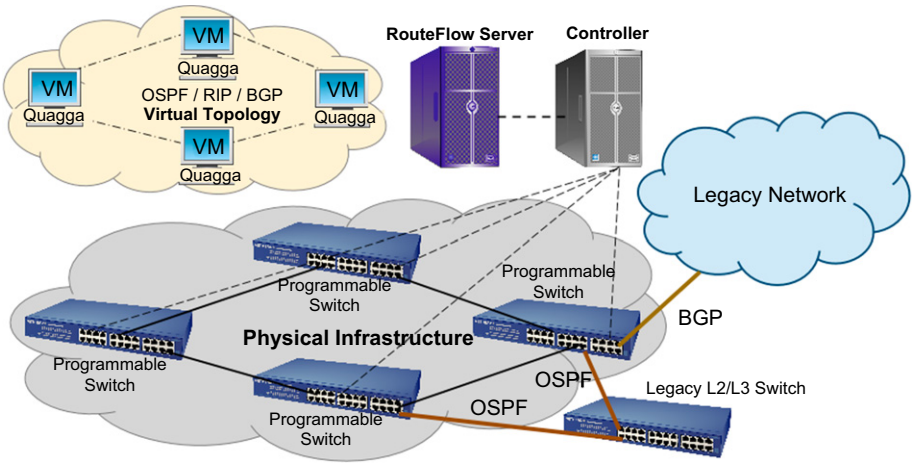


FIGURE 11.5

RouteFlow network topology (Courtesy of CPqD).

Today's Internet is utterly dependent on this family of routing protocols, so, without a clean mechanism to integrate the information they carry into an OpenFlow network, the use of OpenFlow will remain constrained to isolated data center deployments. In Figure 11.5 we see the test network connected to the Internet cloud with BGP routing information injected into the test network via that connection. The test network itself includes one legacy layer two/layer three switch, which communicates routing information to the OpenFlow part of the test network via OSPF. In the test network, we see a cloud of four OpenFlow-enabled switches under the control of an OpenFlow controller. This OpenFlow controller is connected to the RouteFlow server. The RouteFlow server, in turn, collects routing table information from the virtual switches in the adjacent cloud. Figure 11.6 shows these same components in a system architecture view. It is significant that all of the components presented in Figure 11.6 are derived from open source projects, though not all are SDN-specific.

The major components described in the figure are listed below, along with the source of the software used to build the component:

- Network (OpenFlow) controller: NOX, POX, Floodlight, Ryu
- RouteFlow server: Developed in RouteFlow Project
- Virtual topology RouteFlow client developed in RouteFlow Project
- Virtual topology routing engine: Quagga routing engine, XORP
- Virtual topology switches: OVS

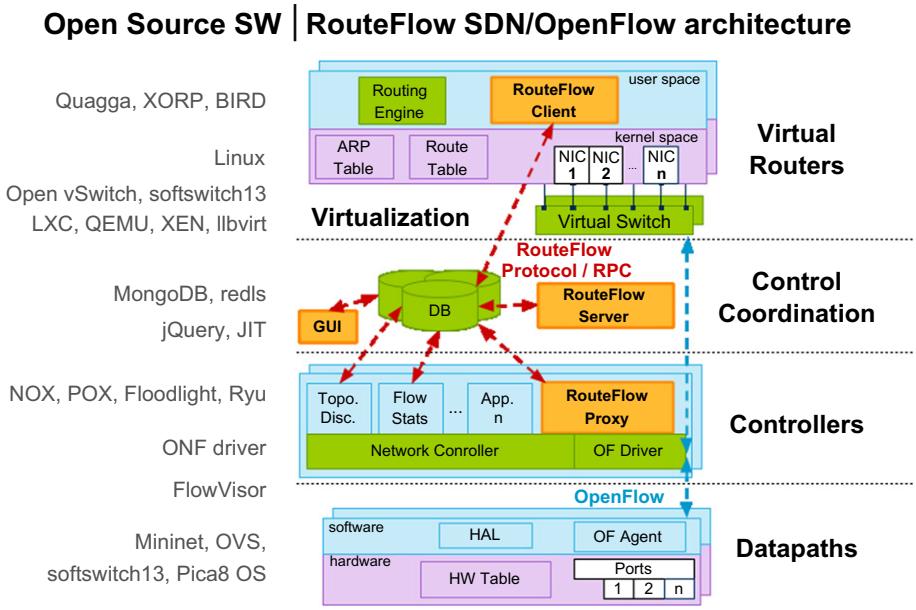


FIGURE 11.6

RouteFlow architecture (Courtesy of CPqD).

- OpenFlow-enabled hardware switches: For example, NetFPGA
- VMs in virtual topology: Linux virtual machines

The virtual routers shown in Figure 11.6 correspond to the VMs in Figure 11.5. Each virtual router comprises an OVS instance that forwards routing updates to the routing engine in the VM. This routing engine processes the OSPF or BGP updates in the legacy fashion and propagates the route and ARP tables in that VM accordingly. The key to the RouteFlow solution is the RouteFlow client that runs on the same VM instance that is in each virtual router. As shown in Figure 11.6, the RouteFlow client gathers the routing information in its local VM routing tables and communicates this to the RouteFlow server via the RouteFlow protocol. The RouteFlow server thus maintains global knowledge of the individual distributed routing tables in each of the virtual routers and can use this information to map flows to the real topology of OpenFlow-enabled switches. This information is communicated from the RouteFlow server to the RouteFlow proxy, which is an OpenFlow controller application running on the same machine as the OpenFlow controller. At this point, the routing information has been translated to OpenFlow tuples, which can be programmed directly into the OpenFlow switches using the OpenFlow protocol. The process results in the OpenFlow-enabled hardware switches forwarding packets just as they would if they followed local routing tables populated by local instances of OSPF and BGP running on each switch, as is the case with legacy layer three switches. RouteFlow is an example of the sort of proactive application that we described in Section 10.2. The flows are programmed proactively as the result of external route information. We contrast this to a hypothetical reactive version of RouteFlow, wherein an unmatched packet would be forwarded to the controller, triggering the determination of the correct path for that packet and the subsequent programming of a flow in the switch.

We have presented this test network showing OpenFlow-enabled hardware switches. RouteFlow additionally supports software switches such as Mininet and OVS for use with network virtualization applications. That is, whereas we described switches depicted in Figure 11.5 as OpenFlow-enabled hardware switches, the RouteFlow project has also demonstrated that these switches may be a soft switch that runs in the ToR alongside a hypervisor.

11.12 Conclusion

SDN's roots are squarely aligned with the open source movement. Indeed, what we call Open SDN in this book remains closely linked with the open source community. In addition to reviewing the better-known SDN-related open software projects, we have provided insight into the many variants of open source licenses in common use. Different open source licenses may be more or less suitable for a given organization. We have defined three broad classes of users that have very different requirements and goals when they use open source software, and we have explained why certain open source licenses might be inappropriate for certain classes of users. Depending on the class of user with which our readers associate themselves, we hope that it is now clear where these readers might find open source applicable for their SDN project. Whatever the commercial future of open source software is within the expanding SDN ecosystem, it is clear that the cauldron of creativity that spawned SDN itself could never have existed without open source software and the sharing of new ideas that accompanies it.

The open source movement may be a powerful one, but as the potential financial ramifications of SDN have become more apparent, other very strong forces have arisen as well. Some of these may be

in concert with the open source origins of SDN; others may be in discord. In the next chapter we look at how those other forces are shaping the future of SDN.

References

[1] Duffy J. Big switch's big pivot. Network World; September 11, 2013. Retrieved from <www.networkworld.com/community/blog/big-switchs-big-pivot>.

[2] Miniman S. SDN, OpenFlow and OpenStack quantum. Wikibon; July 17, 2013. Retrieved from <wikibon.org/wiki/v/SDN,_OpenFlow_and_OpenStack_Quantum>.

[3] Comprehensive list of open source SDN projects. SDNCentral. Retrieved from <www.sdncentral.com/comprehensive-list-of-open-source-sdn-projects>.

[4] Various licenses and comments about them. GNU Operating System. Retrieved from <www.gnu.org/licenses/license-list.html>.

[5] Jacobs D. Floodlight primer: an OpenFlow controller. TechTarget. Retrieved from <searchsdn.techtarget.com/tip/Floodlight-primer-An-OpenFlow-controller>.

[6] Muntaner G. Evaluation of OpenFlow controllers; October 15, 2012. Retrieved from <www.valleytalk.org/wp-content/uploads/2013/02/Evaluation_Of_OF_Controllers.pdf>.

[7] Wilson R. The eclipse public license. OSS Watch; November 12, 2012. Retrieved from <www.oss-watch.ac.uk/resources/epl>.

[8] Erikson D. Beacon, OpenFlow @ Stanford; February 2013. Retrieved from <openflow.stanford.edu/display/Beacon/Home>.

[9] Floodlight documentation. Project floodlight. Retrieved from <docs.projectfloodlight.org/display/floodlightcontroller>.

[10] Lawson S. Network heavy hitters to pool SDN efforts in OpenDaylight project. Network World; April 8, 2013. Retrieved from <www.networkworld.com/news/2013/040813-network-heavy-hitters-to-pool-268479.html>.

[11] Duffy J. Skepticism follows Cisco-IBM led OpenDaylight SDN consortium. Network world; April 9, 2013. Retrieved from <www.networkworld.com/news/2013/040913-opendaylight-268534.html>.

[12] Casado M. List of OpenFlow software projects (that I know of). Retrieved from <yuba.stanford.edu/~casado/of-sw.html>.

[13] Stewart P. Defining Cisco's onePK. The Packet University; March 19, 2013. Retrieved from <www.packetu.com/2013/03/19/defining-ciscos-onepk>.

[14] Welcome to the RouteFlow project. Routeflow. Retrieved from <sites.google.com/site/routeflow>.

[15] Now available: our SDN security suite. OpenFlowSec.org. Retrieved from <www.openflowsec.org>.

[16] Sorensen S. Top open source SDN projects to keep your eyes on. O'Reilly Community; August 1, 2012. Retrieved from <broadcast.oreilly.com/2012/08/top-open-source-sdn-projects-t.html>.