OpenDaylight: Towards a Model-Driven SDN Controller Architecture

Jan Medved
Cisco Systems
San Jose, CA, USA

Anton Tkacik
Cisco Systems
San Jose, CA, USA

Robert Varga
Cisco Systems
San Jose, CA, USA

Ken Gray
Cisco Systems
San Jose, CA, USA

***Abstract*— This paper describes a novel Software-Defined Networking (SDN) Controller architecture that is built on Model-Driven Software Engineering (MDSE) principles. It supports both the "classic" OpenFlow-based approach to SDN and emerging model-driven network management/programmability technologies, such as NETCONF/YANG. The architecture was first implemented in the OpenDaylight Project Hydrogen release, and it is being further evolved in subsequent OpenDaylight releases.**

Keywords—OpenDaylight; SDN; NETCONF; YANG; I2RS; OpenFlow; policy; Model-Driven Software Engineering (MDSE); Object Modeling Group (OMG); late binding; code-generation

I. INTRODUCTION

The OpenDaylight controller software architecture lies at the intersection of three emerging technology trends in networking and software engineering: Software Defined Networking (SDN), model driven software engineering (MDSE) and model driven network management. SDN is driven (in part) by demands for increased network programmability, stemming from demands for increased network agility required by the movement toward increased virtualization, abstraction and programmatic control. An SDN controller brings together two different worlds – networking and software engineering. Therefore, it must function not only as a platform in solution delivery, but also as a "foundry" in application development. The OpenDaylight controller software architecture defines application creation and interaction patterns (APIs and RPC governance), underlying application services (e.g. message routing, formatting and data storage) and has ultimately lead to new development tools for an environment enabled for open source collaboration.

The rest of the document is organized as follows. Section II provides a brief background of the key software and networking trends that define OpenDaylight's model-driven architecture. Section III describes the context and requirements for the OpenDaylight controller architecture/design. Section IV describes the architecture (of the OpenDaylight controller). Section V provides a description of sample controller plugins and applications. Finally, Section VI lists future work and conclusions.

II. BACKGROUND

A. Software-Defined Networking

Software-defined networking emerged from work done at UC Berkeley and Stanford University around 2008. The original SDN network architecture proposed decoupling of network control from packet forwarding and a direct programmability of the network control function [1]. Network intelligence is (logically) centralized in software-based SDN controllers, which maintain a global view of the network. As a result, the network appears to the applications and policy engines as a single, logical switch. This architecture has been further evolved in the Open Networking Foundation (ONF), an organization that also drives the evolution of the OpenFlow protocol standard [2]. OpenFlow allows the network control function to remotely program the packet forwarding function.

Over the years, multiple controllers were developed for this architecture: Beacon [5], Floodlight [6], NOX [7], POX [8], Ryu [9] and more recently ONOS [10]. Each controller implemented OpenFlow as the sole southbound protocol towards the packet forwarding function, and provided access to its control plane functions through northbound REST APIs. These controller platforms supported mostly network control plane applications, with the "killer app" being network virtualization [34].

While we believe that decoupling of control and forwarding functions is a valid and useful SDN use case, we also believe that SDN is defined more broadly than by the OpenFlow protocol abstractions. OpenFlow only supports retrieval and programming of data related to the network forwarding function. However, the network is a source of a vast amount of other useful data that can be utilized by SDN applications. Moreover, network data can be retrieved and networks can be programmed through a variety of protocols and APIs at different levels of information abstraction, such as BGP, NETCONF, Netflow/IP-Fix, SNMP, or vendor-specific CLIs. Basically, SDN defines a generic feedback/control/policy loop between applications and the network: applications get data/information from the network, process the data and make

policy/control decisions that are driven back into the network, as shown in Figure 1. [3]

SDN use cases have evolved from primarily Data Center specific orchestration applications to a variety of diverse SDN applications that harvest network information and use network APIs to program the network. Examples are, in addition to virtualization [31] and [32], applications such as multi-layer path optimization, network analytics, policy configuration [30], service chaining or security [29].

B. Model-Driven Software Engineering

OpenDaylight leverages Model Driven Software Engineering (MDSE) defined by the Object Management Group (OMG) [26]. MDSE describes a framework based on consistent relationships between (different) models, standardized mappings and patterns that enable model generation and, by extension, code/API generation from models. This generalization can overlay any specific modeling language. Although OMG focus their MDA solution on UML, YANG has emerged as the data modeling language for the networking domain.

The models created are portable (cross platform) and can be leveraged to describe business policies, platform operation or component specific capability (vertically) as well as (in the case of SDN) manifest at the service interfaces provided by a network control entity (horizontally and vertically within an application service framework).

MDSE enables run-time mediation and inter-model mapping, for example, if the models are of two different domain types, such as J2EE and UML.

C. Model-Driven Network Management/Programmability

The model-driven approach is being increasingly used in the networking domain to describe the functionality of network devices [4], services [15], policies [18], [19], and network APIs [16], [17]. The protocols of choice are NETCONF and RESTCONF; the modeling language of choice is YANG.

NETCONF [13] is an IETF network management protocol that defines configuration and operational conceptual data stores and a set of Create, Retrieve, Update, Delete (CRUD) operations that can be used to access these data stores. In addition to the data stores CRUD operations, NETCONF also supports simple Remote Procedure Call (RPC) and Notification operations. NETCONF operations are realized on top of a simple Call (RPC) layer. NETCONF uses an XML-based data encoding for the configuration and operational data, as well as for its protocol messages.

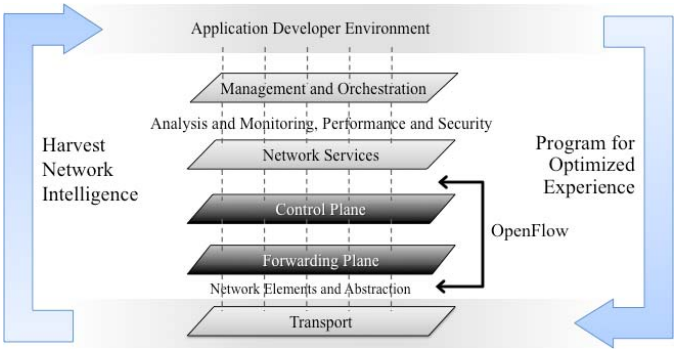

Fig. 1. SDN function definition

RESTCONF [24] is a REST-like protocol that provides a programmatic interface over HTTP for accessing data defined in YANG, using the data stores defined in NETCONF. Configuration data and state data are exposed as resources that can be retrieved with the HTTP GET method. Resources representing configuration data can be modified with the HTTP DELETE, PATCH, POST, and PUT methods. Data is encoded in either XML or JSON.

YANG [11] was originally developed to model configuration and state data in network devices, but it can also be used to describe other network constructs, such as services [15], policies, protocols, or subscribers. YANG is tree-structured rather than object-oriented; data is structured into a tree and it can contain complex types, such as lists and unions. In addition to data definitions, YANG supports constructs to model Remote Procedure Calls (RPCs) and Notifications, which make it suitable for use as an Interface Description Language (IDL) in a model-driven system.

III. REQUIREMENTS

The SDN controller should be both a **platform** for deploying SDN applications and provide (or be associated with) an **SDN application development environment**.

An SDN controller *platform* should be built to meet the following key requirements:

- **Flexibility**: The controller must be able to accommodate a variety of diverse applications; at the same time, controller applications should use a common framework and programming model and provide consistent APIs to their clients. This is important for troubleshooting, system integration, and for combining applications into higher-level orchestrated workflows.
- **Scale the development process**: There should be no common controller infrastructure subsystem where a plugin would have to add code. The architecture must allow for plugins to be developed independently of each other and of the controller infrastructure, and it must support relatively short system integration times. Currently, there are 18 active OpenDaylight projects. Fifteen more projects are in the proposal stage. OpenDaylight projects are largely autonomous, and are being developed by independent teams, with little coordination between them.
- **Run-time Extensibility**: The controller must be able to load new protocol and service/application plugins at run-time. The controller's infrastructure should adapt itself to data schemas (models) that are either ingested from dynamically loaded plugins or discovered from devices. Run-time extensibility allows the controller to adapt to network changes (new devices and/or new features) and avoids the lengthy release cycle typical for legacy EMS/NMS systems, where each new feature in a

network device results in a manual change of the device model in the NMS/EMS.

- **Performance & scale**: A controller must be able to perform well for a variety of different loads/applications in a diverse set of environments; however, performance should not be achieved at the expense of modularity. The controller architecture should allow for horizontal scaling in clustered/cloud environments.

To support development of SDN applications, an SDN controller should also provide (or be associated with) an *application development environment* that should meet the following key requirements:

- Use a domain-specific modeling language to describe internal and external system behavior; this fosters co-operation between developers and network domain experts and facilitates system integration.
- Code generation from models should be used to enforce standard API contracts and to generate boilerplate code performing repetitive and error-prone tasks, such as parameter range checking.
- A domain-specific modeling language and code generation tools should enable rapid evolution APIs and protocols (Agility).
- Code generation should produce functionally equivalent APIs for different language bindings.
- Modeling tools for the controller should be aligned with modeling tools for devices. Then, a common tool chain can be used for both, and device models can be re-used in the controller, creating a zero-touch path between the device and a controller application/plugin that uses its models.
- Domain-specific language/technologies/tools used in the controller must be usable for modeling of generic network constructs, such as services, service chains, subscriber managements and policies.
- The tool chain should support code generation for model-to-model adaptations for services and devices.

IV. Controller Architecture

A. History and Overview

OpenDaylight was originally inspired by Beacon, which introduced the use of Open Service Gateway Interface (OSGi) that is key for modularity and run-time loading of components (plugins) into the controller. The initial "bootstrap" contribution to ODL – the XNC Controller [27] - introduced a major innovation: the Service Adaptation Layer (SAL) that separated southbound (SB) protocol plugins and northbound (NB) service/application plugins. The architecture comprised three layers: SB protocol plugins, the SAL, and the NB Application/Service Functions., as shown in Fig 2.

The SB (protocol) plugins interface with network devices. The SAL adapted the SB plugin functions to higher-level Application/Service functions, which provide the controller's

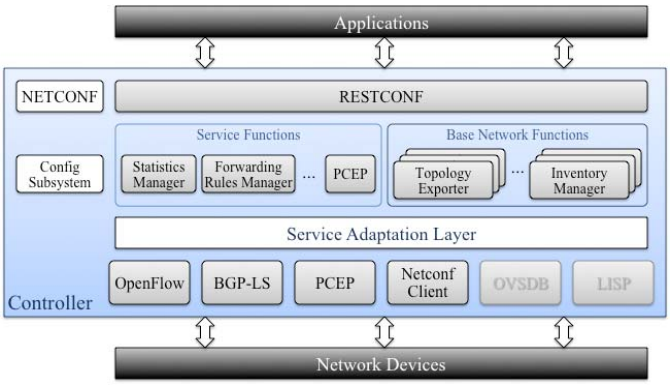

Fig. 2. OpenDaylight controller architecture – network view

NB APIs to applications. Examples of Application/Service functions are the Topology Exporter, the Inventory Manager and the Openflow Statistics Manager. The layered architecture allows the controller to support multiple southbound protocols (via SB protocol plugins) and to provide a uniform set of services and APIs to applications through a common set of NB APIs.

In the original SAL (referred to as the "API-Driven SAL", AD-SAL), developers had to define the SAL APIs used by the plugins and code the adaptation functionality between the northbound and southbound plugins. It quickly became apparent that hand-coding the SAL APIs and adaptations to support new plugin functionality would not scale in an initiative of ODP's magnitude.

B. Evolution to the Model-Driven Service Adaptation Layer

To address the requirements for the controller listed in Section III, a new model-driven architecture was proposed and implemented for the Service Adaptation Layer (referred to as the "Model-Driven SAL", MD-SAL). The architecture is built around concepts, protocols and modeling language described in detail in Section II.

Controller plugins can be either data/service Providers or data/service Consumers. A Provider provides data/services through its APIs. A Consumer consumes services/data provided by one or more Providers. For example, the OpenFlow protocol plugin provides services to add, modify or delete flows from switches connected to the plugin. One of the consumers of the OpenFlow plugin's services is the Forwarding Rules Manager that provides higher-level flow programming services to controller clients.

The OpenDaylight development environment includes tooling that generates this code (codecs and Java APIs). The tooling preserves YANG data type hierarchies, retains data tree hierarchy (providing normal Java compile-time type safety) and data addressing hierarchies. A plugin's APIs are resolved when the plugin is loaded into the controller. The SAL does not contain any plugin-specific code or APIs and is therefore a generic plumbing that can adapt itself to any plugins or services/applications loaded into the controller

From the infrastructure's point of view, there is no difference between a protocol plugin and an application/service plugin. All plugin lifecycles are the same, each plugin is an OSGi

bundle that contains models defining the plugin's APIs. Since all plugins are the same to the controller infrastructure, the architecture can be drawn as shown in Figure 3. The figure also shows how in a cluster of multiple controller instances an MD-SAL instance within a JVM container will be connected to the cluster's message bus and data store.

C. OpenDaylight YANG Models

The Hydrogen production code contains approximately 110 YANG models; 3 models were defined in IETF RFCs, 8 in IETF drafts, the rest have been defined for specifically for the controller and its applications. Approximately 10 models describe IETF protocol PDUs, 27 models describe various aspects of the OpenFlow protocol and 35 models define the controller's internal wiring and configuration. Approximately 15 additional models were used for prototyping and proof-of-concept work.

D. Model-Driven Protocol and Application Plugins

Several MD-SAL-based protocol plugins have been implemented in the Hydrogen release. The **OpenFlow plugin** implements OF1.0 and 1.3 [28]. The **BGP-LS/PCEP plugin** implements both a BGP listener that supports IPv4 and IPv6 AFI/SAFIs [20], the link-state AFI/SAFI [21] and a PCEP Protocol speaker that support the stateful Path Computation Element Protocol (PCEP) [22]. The **NETCONF Connector** (not strictly a plugin but a part of MD-SAL) implements the client-side of the NETCONF protocol [23].

In addition to protocol plugins, several MD-SAL based Application/Service functions have been implemented in the Hydrogen release. They fall into two categories: Base Network Functions and Service Functions. The Base Network Function category contains two foundational services: **Topology** and **Inventory**. The Topology Explorer service provides one or more network topologies discovered by the controller. The Inventory Manager lists the nodes in the network known to the controller; these can include both logical and physical nodes, routers, switches, hosts, appliances, etc..Two types of Service Functions have been implemented: OpenFlow and PCEP. OpenFlow services contain the Forwarding Rules Manager and the Statistics Manager plugins. The PCEP Service function provides transaction-oriented programming of RSVP-TE tunnels.

E. Model-Driven SAL (MD-SAL) Detailed Description

1) Common Concepts

The common concepts used in the MD-SAL design are as follows. An **RPC** is a one-to-one call triggered by a Consumer, which may be processed by a Provider either local or remote. A **Notification** is an event, which a Consumer may be interested in to receive, and which is triggered / originated in a Provider. The **Data Store** is a conceptual data tree, which is described by YANG schemas. A **Path** is a unique locator of a leaf or sub-tree in the conceptual data tree. Finally, a **Mount** is a logically-nested MD-SAL instance, which may be using a separate set of YANG models; it supports its own RPCs and Notifications and it allows for reusing device models and a context in network-wide contexts without having to redefine the device models in the controller ("Mount", as its name suggests, is basically a logical mount of a remote conceptual data store).

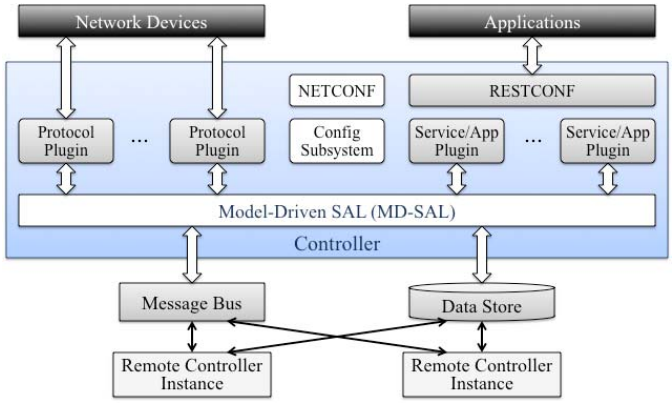

Fig. 3. OpenDaylight controller architecture – software engineering view

2) MD-SAL Functionality and Data Representations

The MD-SAL provides a variety of functions required for adaptation between Providers and Consumers. First, it routes RPC calls between Consumers and Providers (**RPC Call Router).** Second, it provides a subscription-based mechanism for delivery of Notifications from Publishers to Subscribers (**Notification Broker**). Third, it routes data reads from Consumers to a particular data store and coordinates data changes between Providers (**Data Broker**). Finally, it creates and manages Mounts (**Mount Manager**).

The implementation of the above SAL functions requires the use of two data representations and two sets of SAL Plugin APIs. The **Binding-Independent** data format/APIs is a Data Object Model (DOM) representation of YANG trees. This format is suitable for generic components, such as the data store, the NETCONF Connector, RESTCONF, which can derive behavior from a YANG model itself. The **Binding-Aware** data format/APIs is a specific YANG to Java language binding, which specifies how Java Data Transfer Objects (DTOs) and APIs are generated from YANG model. The API definition for these DTOs, interfaces for invoking / implementing RPCs, interfaces containing Notification callbacks are generated at compile time. Codecs to translate between the Java DTOs and DOM representation are generated on demand at run time. Note that the functionality and performance requirements for both data representations are the same.

3) MD-SAL Design

We separated data handling functionality into two distinct brokers: a binding-independent **DOM Broker** that interprets YANG models at runtime and is the core component of the MD-SAL runtime, and a **Binding-Aware Broker** that exposes Java APIs for plugins using binding-aware representation of data (Java DTOs. These brokers, along with their supporting components are shown in Figure 4.

The DOM Broker uses YANG data APIs to describe data and Instance Identifiers specific to YANG to describe paths to data in the system. Data structures in the Binding-Aware Broker that are visible to applications are generated from YANG models in YANG tools. The DOM Broker relies on

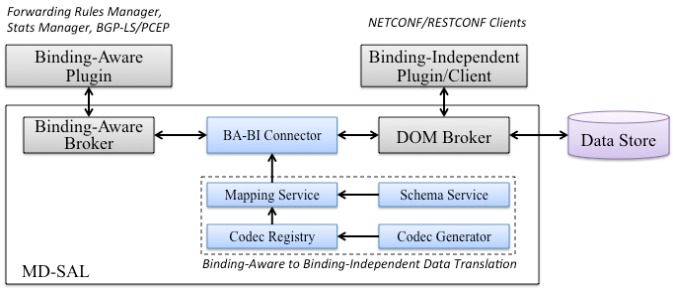

Fig. 4. MD-SAL detailed design

presence of YANG schemas, which are interpreted at runtime for functionality-specific purposes, such as RPC routing, data store organization, and validation of paths.

The Binding-Aware Broker relies on Java APIs, which are generated from YANG models, and on common properties of Java DTOs, which are enforced by code generation. Therefore data transfer optimizations (zero-copy) are possible when a data Consumer and a data Provider are both Binding-Aware.

The Binding-Aware Broker connects to the DOM Broker through the **BA-BI Connector,** so that Binding-Aware Consumer/Provider applications/plugins can communicate with their respective binding-independent counterparts. The BA-BI Connector, together with the Mapping Service, the Schema Service, the **Codec Registry** and the **Codec Generator** implement **dynamic late binding:** the codecs that translate YANG data representations between a binding-independent (DOM) format and DTOs, which are specific to Java bindings, are auto-generated on demand.

The physical **Data Store** is pluggable – MD-SAL provides an SPI through which different data store implementations can be plugged in.

The **Mount** concept and the support for APIs generated from models allow for applications talking to NETCONF devices to be compiled directly against device models – there is no need for controller-level models that represent devices. Device models are loaded into the controller from a NETCONF device when the controller connects to the device, and apps can work directly with them.

V. EXAMPLE APPLICATIONS AND PLUGINS

A. BGP-LS Plugin, RIB and Topology Provider

OpenDaylight includes a processing pipeline (shown in Figure 5), which extracts network topology information from a BGP-LS feed. The pipeline is implemented as a three-stage process with four YANG models: a message-level model for RFC4760-enabled BGP, a model of the BGP local RIB (LocRIB), a message-level and LocRIB models of draft-ietf-idr-ls-distribution [21], and finally the network topology model specified in [25].

The **BGP Parser & RIB** module translates incoming BGP messages into Java Data Transfer Objects (DTOs) according the YANG BGP message-level model. The DTOs are passed to the **Routing Information Base (RIB)** module, which performs BGP Best Path Selection and maintains the LocRIB in the MD-SAL datastore – thus making it widely available to internal and external applications. Finally, the Local RIB data is consumed via change notifications by the **BGP-LS Topology Exporter**, which creates a network topology view and stores it in the MD-SAL datastore, from where it is accessible to external clients via RESTCONF or to internal plugins via generated Java DTOs.

The BGP-LS/PCEP plugin implementation was adapted to the model-driven OpenDaylight architecture from an existing non-model-driven system. Compared to the previous implementation, the model-driven approach saved 42% of the code, and provided more complete APIs and better type checking on the APIs.

B. Generic Model-to-Model Adaptations

The Model-Driven SAL allows easy creation of model-to-model transformation chains as shown in Figure 6. Consider an example where a customer wants to write his application against a generic Flow Service API. Device from Vendor 1 supports flow creation through Access List Programming. Device from Vendor 2 supports flow creation through the OpenFlow protocol. Each vendor can create an OpenDaylight adaptation plugin that adapts their respective device's programming model to the generic Flow Service model required by the customer. Vendors can create their respective plugins independently of each other and the final solution may be integrated by the customer or a 3rd party integrator.

Model to model translations and evolutions of the existing messaging system position the OpenDaylight platform as a potential Next Generation management system.

C. API Evolution/Generic Model Adoption

The work done in "standardized" model efforts attempts to isolate the common model in overlaps between different models of the same service and augment them with vendor/platform specific extensions (described in MDSE architectural goals). Through these engagements, the model mapping will become simplified and localized (non-common components/extensions to the model will be mapped in the plugins that support them).

The I2RS WG in the IETF represents one such effort, seeking to create common models for the router RIB (programming ephemeral device state) and the network

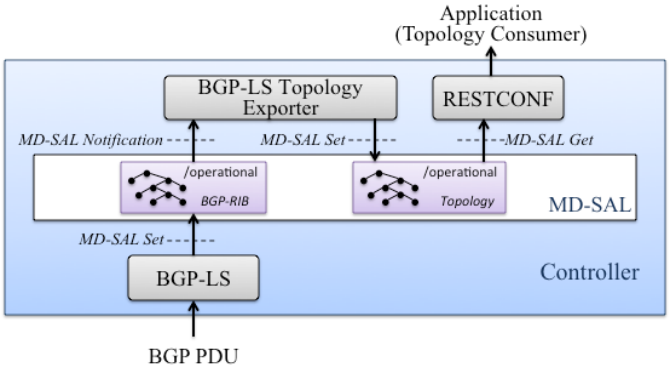

Fig. 5. BGP-LS processing chain

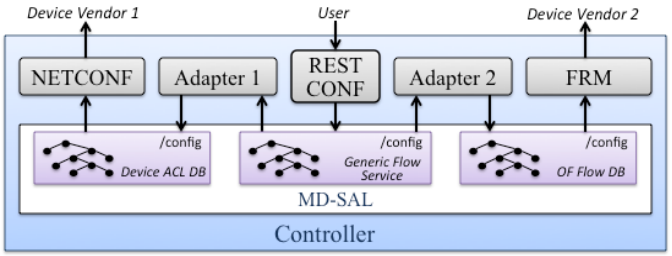

Fig. 6. Model-to-Model Translation

topology (a network information service). The topology model illustrates an object-oriented approach to creating models, where the base model only contains concepts common to all topologies, such as nodes, links and node connectors. The base model is then augmented (extended) with topology specific attributes, for example for L3 IP and Openflow topologies.

VI. Conclusions and Future Work

The OpenDaylight SDN controller and development environment expand and extend the basic premise of SDN. It enables support of multiple southbound protocol plugins and a diverse set of services and applications. In the process, it brings network applications closer to the network and allows application developers and network researchers to focus on SDN APIs rather than protocols used to communicate with network devices. It proved that YANG, in addition to being a modeling language, is a viable IDL to describe SDN network and network device APIs, and that NETCONF/RESTCONF is a viable SDN protocol with capabilities that match and sometimes exceed those of traditional forwarding-function-centric SDN protocols.

Future work on OpenDaylight will include infrastructure work for improved clustering and scaling, and code generation tools for generation of more extensive boiler plate code, skeletons and helper functions for applications. Language bindings for other programing languages, such as Python, are planned. Finally, more protocol plugins and new applications are under development [32].

References

[1] N. McKeown, G. Parulkar, T. Anderson, L. Peterson, H. Balakrishnan, J. Rexford, S. Shenker, J. Turner, "OpenFlow: Enabling innovation in campus networks," March 14, 2008

[2] "Software-Defined Networking: The New Norm for Networks," ONF White Paper, April 13, 2012

[3] D. Ward, "Is it just software defined networks (SDN)?" April 16, 2012, https://blogs.cisco.com/news/is-it-just-sdn/

[4] A. Clemm, "Navigating device management and control interfaces in the age of SDN", February 28, 2014, http://blogs.cisco.com/getyourbuildon/navigating-device-management-and-control-interfaces-in-the-age-of-sdn

[5] "Beacon," https://openflow.stanford.edu/display/Beacon/Home

[6] "Floodlight," http://www.projectfloodlight.org/floodlight/

[7] "NOX," http://www.noxrepo.org/nox/about-nox/

[8] "POX," http://www.noxrepo.org/pox/about-pox/

[9] "Ryu," http://osrg.github.com/ryu/

[10] "ONOS," http://tools.onlab.us/onos.html

[11] M. Bjorklund et al, "YANG - A data modeling language for the network configuration protocol (NETCONF), RFC 6020," October 2010, http://datatracker.ietf.org/doc/rfc6020/

[12] M. Scott & M. Bjorklund, "YANG Module for NETCONF Monitoring, RFC 6022," October 2010, http://datatracker.ietf.org/doc/rfc6022/

[13] R. Enns, M. Bjorklund, J. Schoenwaelder, A. Bierman, "Network Configuration Protocol (NETCONF)", http://tools.ietf.org/html/rfc6241

[14] A. Bierman et al, "RESTCONF Protocol, draft-bierman-netconf-restconf-04," February 13, 2014, http://datatracker.ietf.org/doc/draft-bierman-netconf-restconf/

[15] S. Wallin and C. Wikstrom, "Automating network and service configuration using NETCONF and YANG," Usenix LISA '11, December 4-9 2011 Boston, MA

[16] S. Raza, D. Lenrow, P. Menezes, E. Dorn, T. Tsou, F. Schneider, "Open Networking Foundation North bound interface working group (NBI-WG) charter, v1.1,", October 10, 2013

[17] "Interface to the Routing System (i2rs) IETF working group," http://datatracker.ietf.org/wg/i2rs/

[18] "Application centric infrastructure object-oriented data model: gain advanced network control and programmability," http://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/aci-fabric-controller/white-paper-c11-729586.pdf

[19] "The Cisco Application Policy Infrastructure Controller," http://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/unified-fabric/white-paper-c11-730021.pdf

[20] T. Bates, R. Chandra, D. Katz, Y. Rekhter, "Multiprotocol Extensions for BGP-4", https://tools.ietf.org/html/rfc4760

[21] H. Gredler, J. Medved, S. Previdi, A. Farrel, S. Ray, "North-Bound Distribution of Link-State and TE Information using BGP", http://tools.ietf.org/html/draft-ietf-idr-ls-distribution

[22] E. Crabbe, J. Medved, I. Minei, R. Varga, "PCEP Extensions for Stateful PCE", http://tools.ietf.org/html/draft-ietf-pce-stateful-pce

[23] R. Enns, M. Bjorklund, J. Schoenwaelder, A. Bierman, "Network Configuration Protocol (NETCONF)", http://tools.ietf.org/html/rfc6241

[24] A. Bierman, M. Bjorklund, K. Watsen, and R. Fernando, "RESTCONF Protocol, draft-bierman-netconf-restconf-04," February, 13, 2014, https://datatracker.ietf.org/doc/draft-bierman-netconf-restconf/

[25] A. Clemm, H. Ananthakrishnan, J. Medved, A. Tkacik, R. Varga, N. Bahadur, "A YANG Data Model for Network Topologies", http://tools.ietf.org/html/draft-clemm-i2rs-YANG-network-topo

[26] "OSGi", http://www.osgi.org/

[27] "Cisco Extensible Network Controller", http://www.cisco.com/c/en/us/products/cloud-systems-management/extensible-network-controller-xnc/index.html

[28] "Openflow Protocol Specification," https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf

[29] "Defense4All OpenDaylight Project," https://wiki.opendaylight.org/view/Defense4All:Main

[30] "Group Based Policy OpenDaylight Project," https://wiki.opendaylight.org/view/Group_Policy:Main

[31] "Virtual Tenant Network (VTN) OpenDaylight Project," https://wiki.opendaylight.org/view/OpenDaylight_Virtual_Tenant_Network_(VTN):Main

[32] "OpenDOVE OpenDaylight Project," https://wiki.opendaylight.org/view/Open_DOVE:Main"

[33] OpenDaylight New Project Proposals," https://wiki.opendaylight.org/view/Project_Proposals:Main#Proposals_Submitted_For_Review

[34] "VMWare NSX – the platform for network virtualization," http://www.vmware.com/files/pdf/products/nsx/VMware-NSX-Datasheet.pdf