UNIVERSITY OF ALBERTA

Master of Science in Internetworking

Capstone Project (MINT-709)

Deployment of SDN Controller and configuring Cisco devices using NETCONF

Presented by

Jaspal Singh

Supervisor

Shahnawaz Mir

Table of Contents

List of Figures

Project Introduction

Traditional networking concepts demand a lot of manual operations and high operational cost simultaneously. Therefore, the biggest challenge for large organizations and entities is implementing the complete network manually, along with monitoring and troubleshooting operations. Moreover, as the network grows, the organization has to go through the change implementations, and if it is a manual process referring to traditional networking, then the organization has to spend a lot in terms of time and operational costs. Software-defined networking can solve the above-discussed problem and enable the implementation of automation in the network. It leads to the one-touch provisioning of devices and automated monitoring. It also makes the network scalability easily achievable with less cost of operations and fast compared to traditional networking. SDN refers to implementing software and automation integrated with the traditional network, making its features more flexible and scalable. The SDN controller is the central part of the SDN architecture. In this project, the challenge is to develop a MINT SDN controller in the MINT lab.

Furthermore, to overcome this challenge, we will use an open-source SDN controller ODL to integrate the Cisco devices with the SDN controller in the MINT lab. Testing the compatibility of the YANG models of Cisco devices with the current version of the ODL is also a challenge. We will use the NETCONF protocol to communicate between the Cisco device and the ODL controller. There are multiple features and plugins in the ODL to enable SDN capabilities. We will test and implement those plugins in our MINT SDN lab as per the requirements. This will set the base for integrating cisco devices in the MINT SDN lab environment.

Introduction to Software-Defined Networking (SDN)

Software Defined Networking (SDN), as the name represents, belongs to implementing software-related features in the networking field. The SDN model allows the current physical hardware device to integrate with the software-based applications and provide automated control over the network, leading to more flexible network management. The main idea behind SDN architecture is to centralize the control plan of the devices in the network and obtain control and complete network overview at a centralized controller [1]. This approach allows a clean and easy integration of software applications with the end network devices through a centralized controller. Also, it reduces the need for network implementation at the individual device level and converts the complex implementation process to a controlled and automated process, saving time and cost simultaneously.

Most organizations today are moving towards virtualization, and all the new implementation and network changes are mostly related to the virtualization of the devices. As a result, traditional internetworking is changing quickly, and hardware implementations are reducing significantly. The companies are trying to lower their dependency on the actual hardware network devices and have started implementing the network devices on the servers using virtual images. A single server can handle many network devices and allows the administrators to access and move the network devices within the topology from a single hardware. This move toward virtualization has unlocked the scope for SDN, and organizations can implement automation and network programmability with the help of SDN. For example, if a company has a network topology with a large number of switches, routers and firewalls, even if significant devices are on VM. The VM can solve the device installation and moving efforts, but the administrator must still configure those devices manually. It will take considerable time to configure newly added switches, routers, and firewalls one by one, and the possibility of error is also pretty high. At this point, the SDN comes into the picture, and with one click administrator can provision the new devices. VLAN configuration for all the switches is possible with one click, and so on with other network devices.

The SDN can provide the complete overview and control of the network in one place without going into each network device individually. The SDN allows the integration of network devices with programmable applications, and the administrator can control or manage the devices via those applications [2]. The SDN works as the interface between these two. SDN being the center point, the automation of the process is easily achievable and is highly programmable as per the requirement. The SDN serves as the API (Application programming interface) between the applications and network devices. The network administrator can use predefined standard APIs to control and manage network devices via the SDN controller [3].

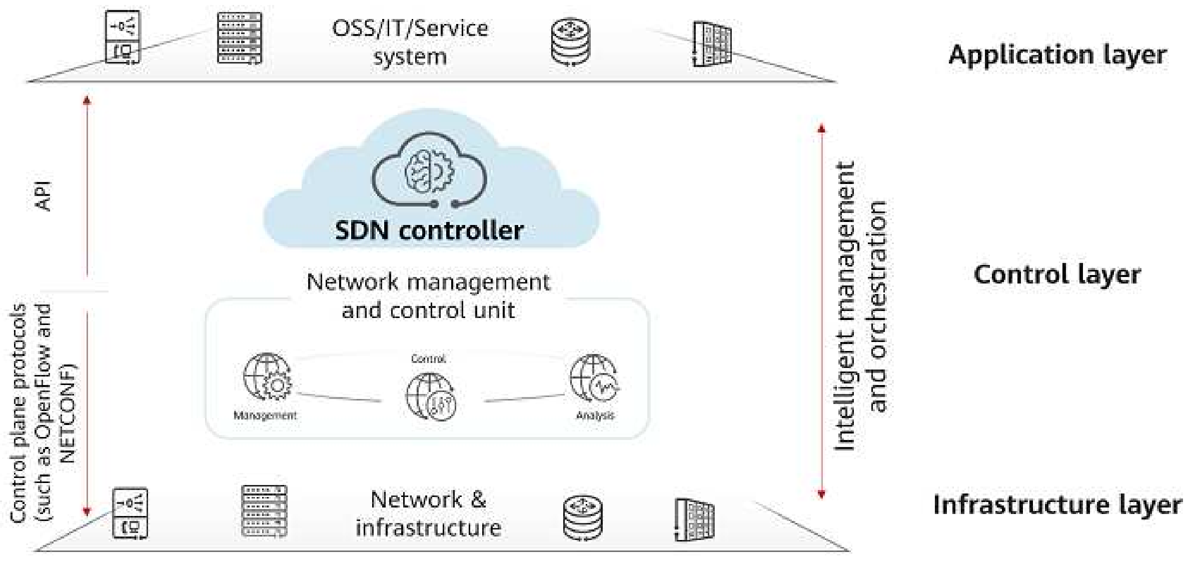


Figure 1.SDN Architecture Layers Overview

https://support.huawei.com/enterprise/en/doc/EDOC1100202532

Key benefits of implementing SDN

- It reduces the requirement for actual hardware devices in the network topology.
- The centralized control plane option enables open-source automation tools to integrate with the network devices.
- Increase the efficiency with the programmable provisioning of new devices.
- Reduces the implementation and configuration errors as compared to the manual process.
- It provides the organization with a more flexible and scalable option with minimum operational cost.
- Organizations can deploy applications and services more efficiently due to centralized control and access to the different API options.
- All the network requirements can be merged on a single centralized control, enabling different operable functions from a single point.
- Reduces the network complexity by introducing automation in the network implementation and operations. [4]

Challenges in configuration and Integration of Cisco devices with ODL

Open Daylight controller is a Linux foundation group product, not a Cisco vendor-specific product. So, the challenge here is integrating the cisco devices with the ODL controller and utilizing different APIs to implement the above-discussed features of SDN. We will not be using any physical hardware device for this setup, so deploying this SDN architecture on VM is a topic of research and trials.

The compatible version and IOS should match for successful communication with the controller and software applications. The primary protocol or plugin we will be focussing on the SBI side is the NETCONF. The challenge is integrating the cisco devices with ODL and configuring these devices via controller using the NETCONF connector plugin feature on the ODL. Deployment of NETCONF compatible devices and valid NETCONF configuration on the Cisco side is a significant part of the problem. All the products or devices in this setup may be on different VMs. This is another challenge to integrating all the devices running on different VMs. As different hardware devices offer different YANG data model capabilities, it is essential to fetch the device capabilities and test the compatibility with the SDN controller.

The end network device should also share the data model schemas along with the YANG capabilities. ODL may fail to mount the data model schema due to standard YANG incompatibility. The ODL works on the YANG models concerning the RFC 6020, and the targeted network devices should follow the RFC 6020 to be compatible with the ODL; otherwise, the ODL will not fetch the capabilities from the device.

The RESTCONF on the NBI side can be integrated with different scripting languages to automate network operations. Testing the RESCONF plugin of ODL with the NETCONF connector on the SBI side is also a challenge in our project environment, where ODL will be responsible for flow management at the controller level. ODL is a distributed architecture SDN controller, so it is important to match and install all the required features of ODL.

Control Plan and Data Plan

A control plan is that part of the architecture which decides the routing path and gives the overview of the network topology. It is the source of the information which specifies how the device will forward the packet in the network. The devices in the network create their forwarding tables with the use of a control plan. Also, the control plan is the routing information generator and works at the self-dependent level. However, A data plan drives the information generated by the control plan to forward the packets at the physical level from source to destination. The actual forwarding of packets between physical interfaces results from data plan functions. The data plan relies on the details provided by the control plan and does not generate any routing information by itself. [5]

The SDN approach of decoupling the control and data plan provides several benefits and is the essential step of the SDN architecture. By separating the control and data plan, it is easy to scale both at the individual level and as per the requirements. It will be more straightforward to make the control plan programmable after the decoupling, and both will not be affected if anything happens to the hardware device, as the control plan can be implemented on a different server when separated.

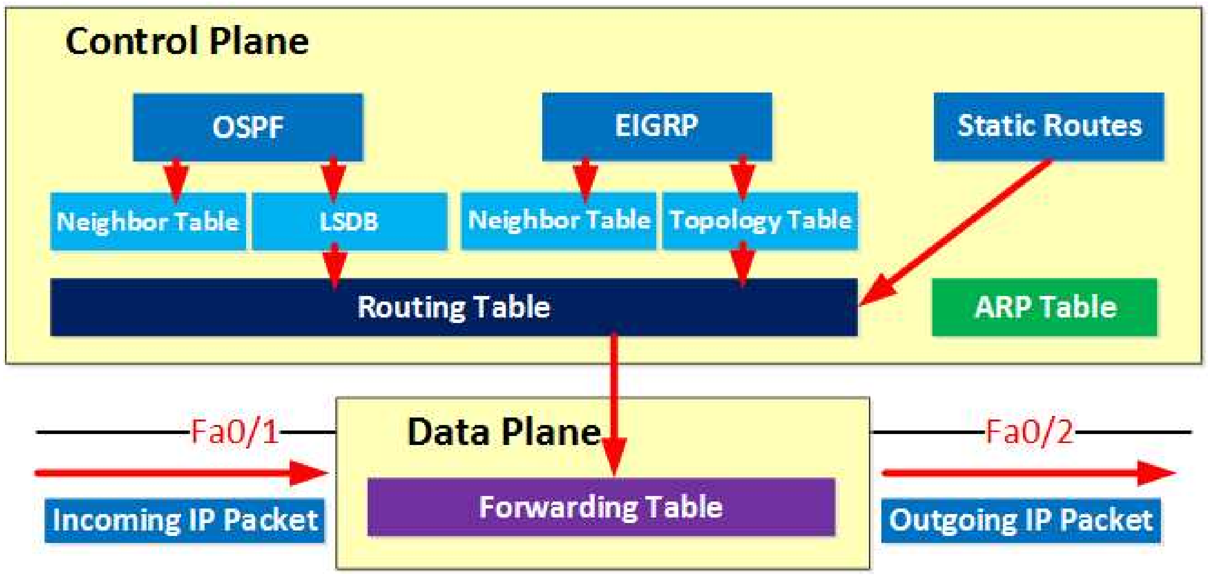


Figure 2. Control Plane and Data Plane

https://networklessons.com/switching/cef-cisco-express-forwarding

Comparison with traditional networking

In the case of traditional network implementation, the administrator has visibility of the device's control and data plan at the individual device level. However, that makes network monitoring and configuration more challenging and time-consuming. Also, any process, in that case, is more error-prone than the automated process. Moreover, in a traditional network, everything relies on the hardware implementation and change related to capacity or upgradation goes through the hardware implementation. Regardless, in the case of SDN, most of the part is pushed to the software implementation, so the capacity and traffic flow expansion are readily available via the centralized control plan user interface without infusing in new hardware. That makes the SDN stand out from traditional networking by providing the most significant advantage of effortless scalability without spending much on operations and deployments.

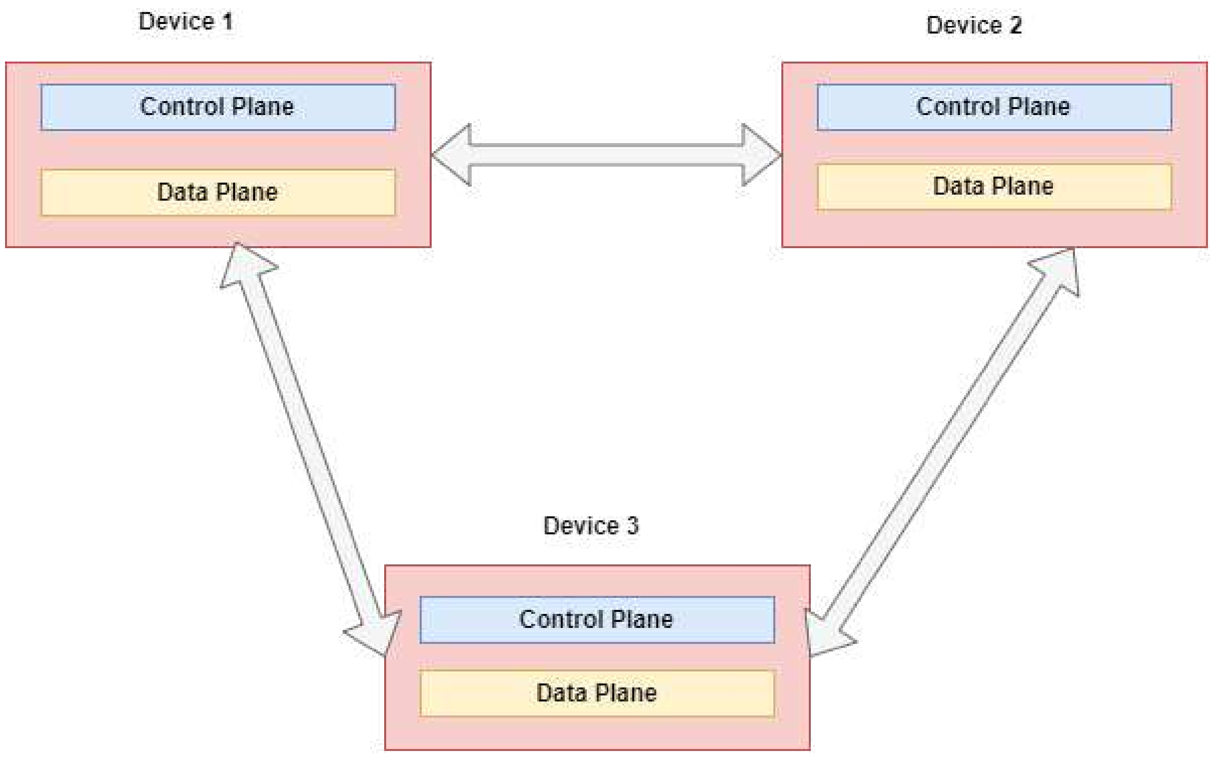


Figure 3. Traditional Network Overview

On the other hand, the direct implementation of APIs in traditional networking is a significant problem as different hardware vendors have different proprietary software and integrating APIs is a big problem. That is why the traditional networking architecture uses only networking protocols instead of APIs due to the custom ASICs of device vendors. SDN architecture uses the open-source protocol with the APIs.

As the control plan is decoupled from hardware to software, it is straightforward to integrate the different vendor devices with APIs through the SDN controller. SDN controller works as a software interface between the device and the application user interface, enabling the network to be programmable and automated compared to traditional networking architecture. These capabilities of SDN make it the best option to implement network infrastructure in today's growing virtualization network environment. [3]

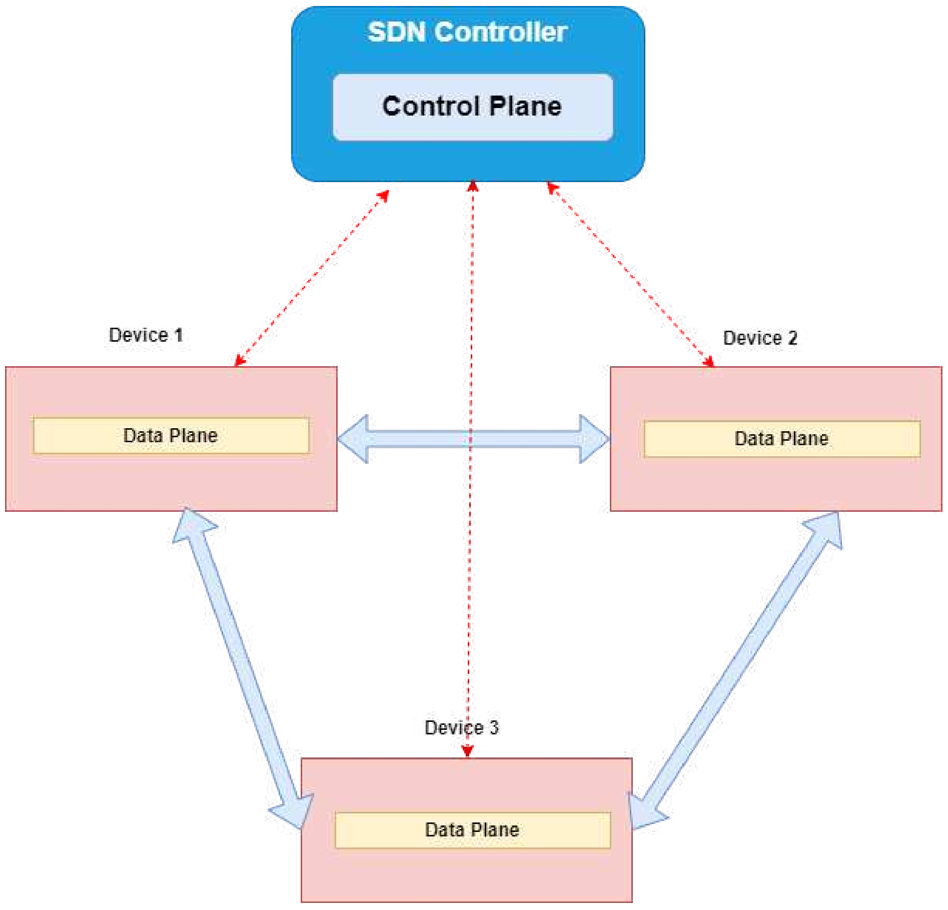


Figure 4. Software Defined Network Overview

SDN Architecture

The SDN architecture consists of three parts: the controller, the application and the network devices. The controller is the middle and essential part that connects the applications with the network devices and efficiently uses the programmability features. As mentioned earlier, the significant difference between traditional architecture and SDN is that the SDN separates the control and data plane of the network device and takes command of the control plan of the device to make a centralized control, and the device only does the work of data plan to forward the traffic.

In SDN, the control plan decisions are operatable from the controller instead of network devices and provide complete network control from a single point. A user can provide inputs from the application as per the requirements, and the controller makes the control plan decisions as per the input and forwards the decision to the network devices from a single point to forward the packet at the physical level. This way, SDN architecture uses to control and data plans more effectively.

The controller is responsible for the traffic flow between the plans and enables the integration of devices with the applications in the management plan. The controller is also accountable for device management and flow information of the complete network. Multiple SDN controllers are available in the market with different functionality and architecture, but the end goal is the same for all. They offer different plugins and APIs, and users can implement them per their requirements.

The SDN controller is the central part of the SDN architecture. It needs to communicate with the application part and with the network devices. Therefore, the controller uses the communication interface to talk with the applications and network devices. SBI (Southbound Interface) is the interface used by the controller to talk with the network devices, and NBI (Northbound Interface) is used to talk with the applications. The specific protocols handle this communication between the interfaces and controller, but adding API (application programming interface*)* enables network programmability. API help to communicate between different programs and applications, allowing the controller, in this case, to control the network device data forwarding plane with programmability access. [6]

As mentioned earlier, the controller will have all the control plan information of devices in the network topology, and the NBI will be used to share that information with applications at the user end. Applications can use APIs to control the network device by gathering information via NBI and sending programs to the controller via NBI. The flow starts from the SDN application working at the application layer, the SDN controller handling the control plane, and finally, the network devices working at the data plane level.

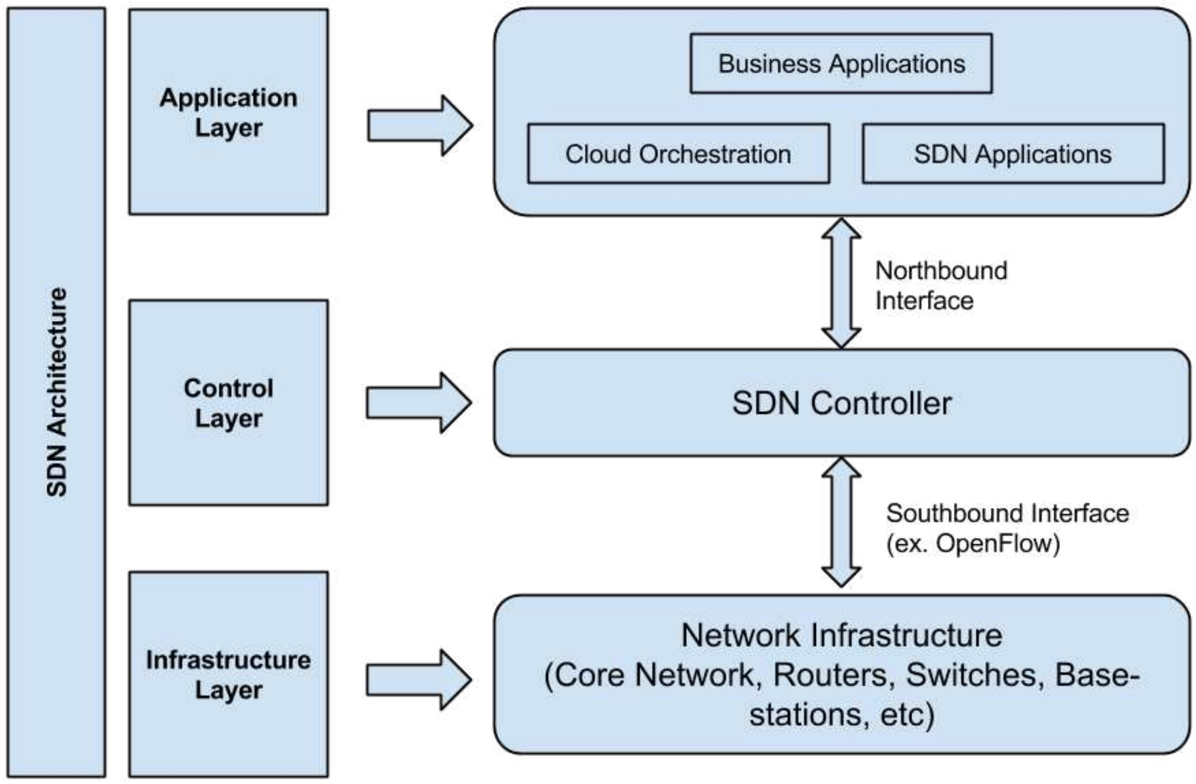


Figure 5. SDN Architecture Flow Diagram

https://medium.com/@fjzdjd1204/openflow-opens-new-doors-for-network-ad1d489d521b

This architecture facilitates more scalability in the network due to the modular design and controllability at each layer. The same modular design also provides more security control at different levels. However, the centralized controller makes a disadvantage of single point of failure risk in the topology. SDN architecture provides the advantage of deploying the network faster with low-cost implementation. Moreover, this implementation makes cloud network operations easily executable and cost-effective.

Southbound Interface

Southbound API provides an open interface to allow communication between the decoupled control and data plan. The southbound API provides the path to push the configuration to the data plan. As discussed earlier, in traditional networking architecture, it is challenging to push the configuration to hardware devices as all the vendors have different hardware architectures and use different programming languages. However, in the case of SDN, the southbound API act as an open interface and makes it a standard interface for a controller to data plan communication for all different hardware devices [7].

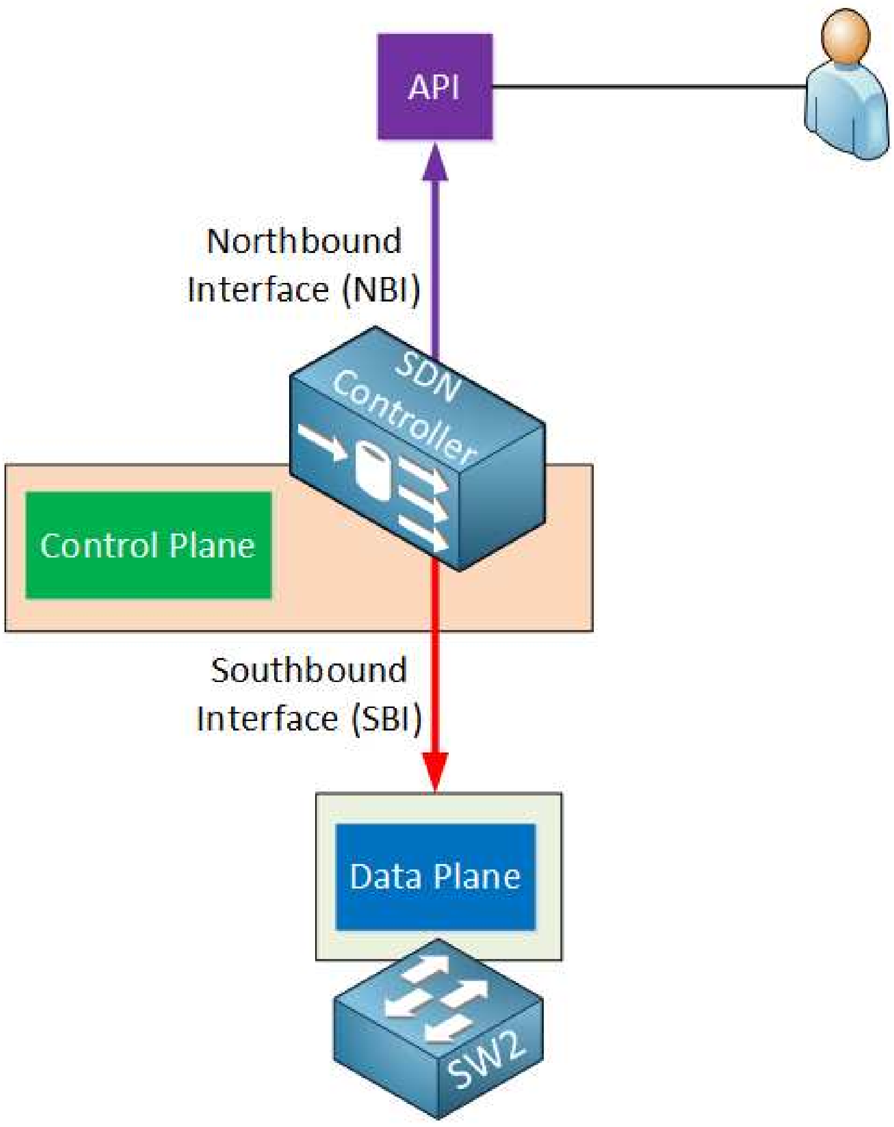


Figure 6. SDN Southbound Interface

https://networklessons.com/cisco/ccna-routing-switching-icnd2-200-105/introduction-to-sdn-software-defined-networking

Northbound Interface

The management plan or the application user interface communication with the controller is as important as communication between the control and data plan. All the network topology overview and decision-manipulating information are communicated via the Northbound API. Programmable actions from the application are transferred to the control plan via the NBI [7]. SDN uses open Northbound API to make sure there is a standard API between the controller and applications.

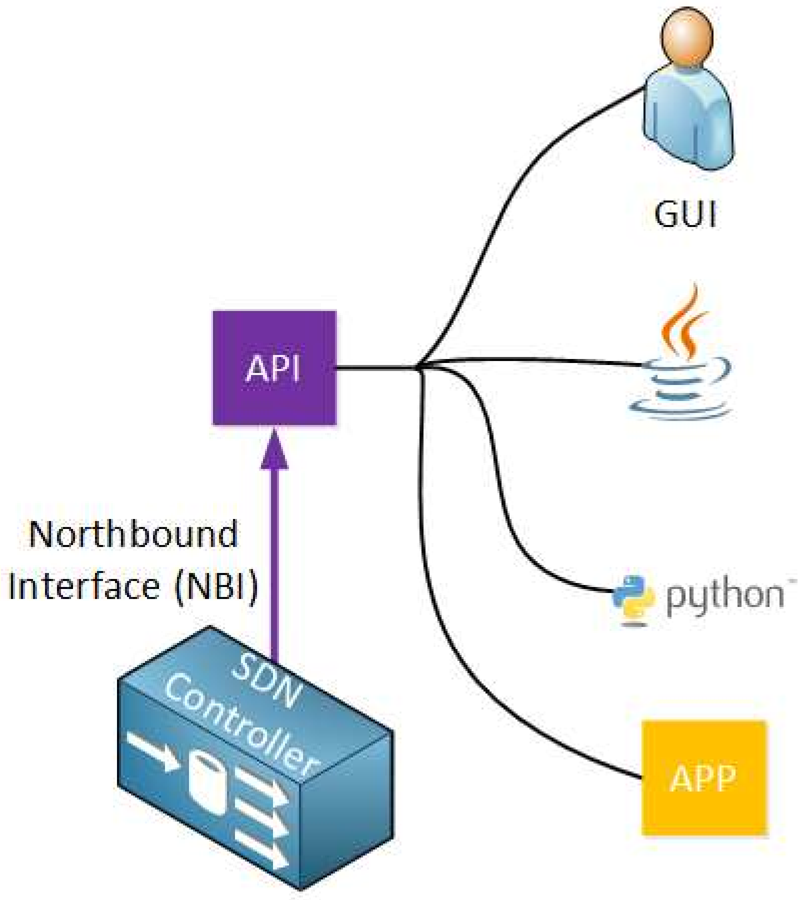


Figure 7. *SDN Northbound Interface*

https://networklessons.com/cisco/ccna-routing-switching-icnd2-200-105/introduction-to-sdn-software-defined-networking

The SDN controller mostly uses the REST API to deliver the information between the applications and network devices. The REST API uses the HTTP message structure to send and receive information [8]. As we know, HTTP is a protocol used by web browsers to send and retrieve data from servers. In the same way, REST API uses HTTP messages to send and receive data from applications or devices in the network [8]. GET messages are used to retrieve the data, and PUSH/PUT messages are used to send the data [8]. In the case of SDN, we can define the targeted nodes and objects in the SDN controller and easily retrieve or send the data as per the requirement. The most used data format for this data is XML and JSON.

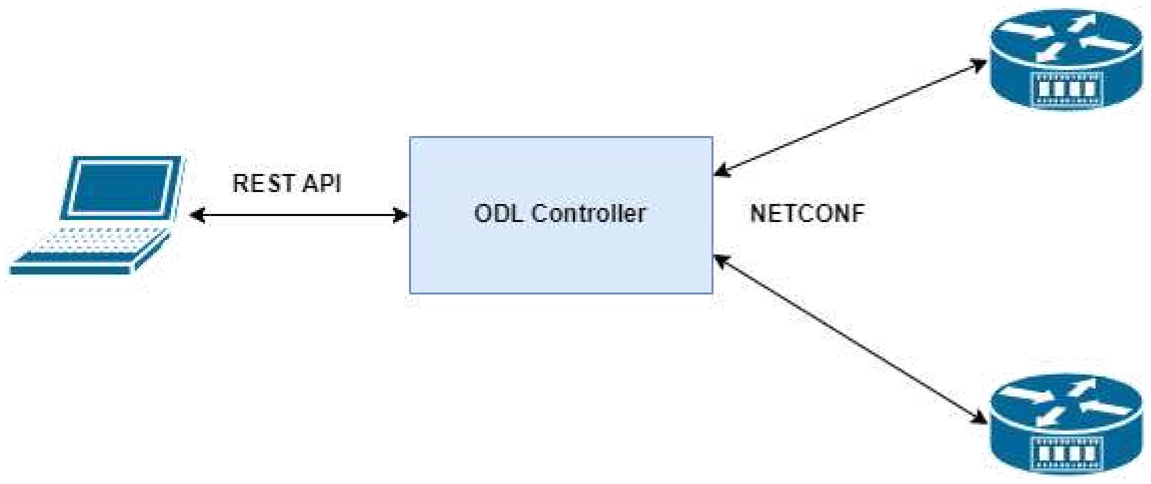


Figure 8. ODL and REST API Overview

Multiple protocols are available for network programmability, but the most popular and used protocols are NETCONF and RESTCONF.

NETCONF

NETCONF (Network configuration) is a more flexible way of modifying the device's configuration rather than using CLI for configuration. NETCONF uses XML data format to recover information or to configure the device [9]. In addition, we can integrate the NETCONF protocol with the SDN controller, adding flexible and automation features for device configuration and monitoring. NETCONF uses SSH over TCP port 830 to exchange information and can provide API support to controllers and applications for SDN implementation. The network devices store the data in the YANG data model structure, which defines the standard way of storing configurational data in the devices so that protocols like NETCONF can use it to retrieve and update the data in the device. Simply, the network devices store the configuration data in the standard YANG model data structure, and the controller can send RPC (Remote procedure call) to change or retrieve the configuration in XML format [10].

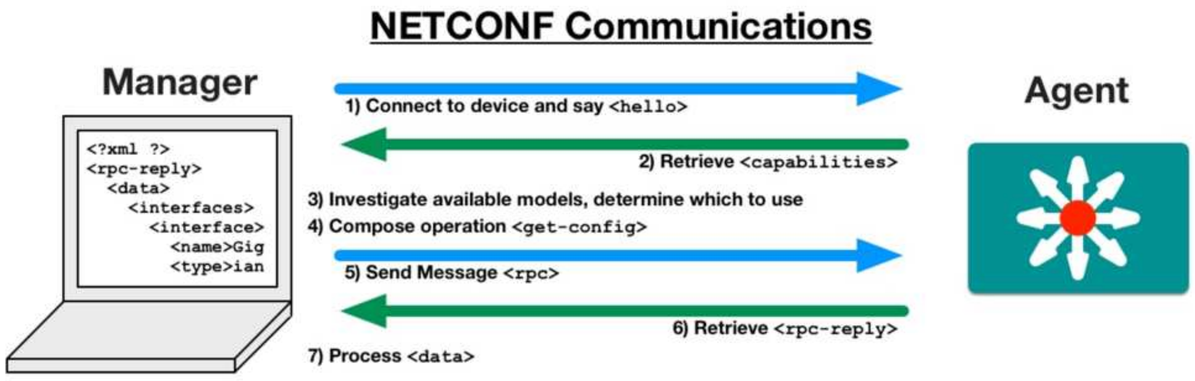


Figure 9. NETCONF Session Diagram

https://medium.com/@k.okasha/network-automation-and-the-rise-of-netconf-e96cc33fe28

In the same way, the network device replies with the requested information in XML data format. There are multiple predefined NETCONF operations for specific tasks. For example, the get-config operation retrieves the configuration, and copy-config is to edit or pushes the configuration to the device. This communication uses the XML data format and TCP port 830 (default) over SSH protocol [11]. Currently, the standard data modelling language supported by multiple vendors is the YANG, and SDN uses this language model as a standard communication model language to perform different functions on the network devices [11]. SNMP protocol is also in the same category as NETCONF, but NETCONF provides more functionalities.

NETCONF features :

1. The distinction between configuration and state data

2. Multiple configuration data stores (candidate, running, startup)

3. Configuration testing and validation support (validation, confirmed commit, rollback on an error)

4. Selective data retrieval with filtering

5. Streaming and playback of event notifications (filter the notifications with date)

6. Extensible procedure call mechanism (additional operations). [9]

RESTCONF

RESTCONF protocol is the same as NETCONF in terms of the main functionality and use. The most significant difference is that RESCONF uses the HTTP protocol to send and receive communication messages and supports XML and JSON, both data structure languages [13]. Therefore, RESTCONF is not the NETCONF substitute. Moreover, it adds more features to NETCONF and enables the users to use the HTTP RESTful interface to send and retrieve information from the device NETCONF configuration datastore [13]. It performs the same operations as the NETCONF. For example, the GET operation retrieves the configuration, and POST/PUT is to edit or pushes the configuration to the device. In addition, both NETCONF and RESCONF deliver the same functionality because they use the same YANG data model structure.

YANG

The Network Configuration Protocol (NETCONF), NETCONF remote procedure calls, and NETCONF notifications are used to manage the configuration and statistics data of the network device, and YANG is a data modelling language that converts the device configuration and statistics data into a data model, which the NETCONF or other available API protocols can access. [14]

YANG represents the data models in terms of the modules and submodules. The hierarchical structure of the data is represented as a tree model, and each node is defined with a name. The nodes have child nodes further, and the child node could contain multiple elements to represent data [14]. Module header, revision, and definition statements are the three different statements that make up a module. The definition statements are the module's body where the data model is specified, while the revision statements provide information about the module's history. Finally, the module header statements give an overview and provide information about the module.

When the network device is integrated with the SDN controller, the controller initially fetches all the YANG data models from the device and stores them in its directory. The network device shares the data models with the NETCONF controller as its capabilities, and it has the below-mentioned format:

Also, the data model capabilities depend on the device type and hardware models. The device shares several data models, but we are taking the Cisco-IOS-XE-native data module as an example for our project.

"name": "Cisco-IOS-XE-native",

"revision": "2020-07-04",

"namespace": "http://cisco.com/ns/yang/Cisco-IOS-XE-native"

Here the name represents the module name, and the revision defines the current version of the mentioned module in the device. Each module defines an XML namespace as a globally unique URI. The SDN controller or the NETCONF plugin uses this namespace to encode the corresponding data in XML format.

For data modelling, YANG specifies four different node types:

Leaf Nodes: The leaf node represents simple data like a string or an integer. For example, the hostname element here is the leaf node. [14]

Example:

```
"native": {
    "hostname": "mint",
    "ethernet": {
      "Cisco-IOS-XE-ethernet:cfm": {
        "alarm": {
          "reset": 10000,
          "delay": 2500
```

Leaf-List Nodes: A series of leaf nodes with precisely one value of a specific type per leaf is known as a leaf-list. [14]

Container Nodes: An internal data node that exists once per instance in the Yang model data tree. A container is nothing more than a collection of child nodes. For example, in the Cisco-IOS-XE-native module, native is the container node. [14]

Example:

```
"native": {
    "hostname": "mint",
    "ethernet": {
      "Cisco-IOS-XE-ethernet:cfm": {
        "alarm": {
          "reset": 10000,
          "delay": 2500
```

List Nodes: It represents the collection of child nodes. The values of each entry's key leafs serve as a unique identification for that entry, which functions like a structure or record instance. A list can include any number of child nodes of any type (including leafs, lists, containers, etc.), and it can have multiple vital leafs [14]. For example, the number of interfaces under the interface node.

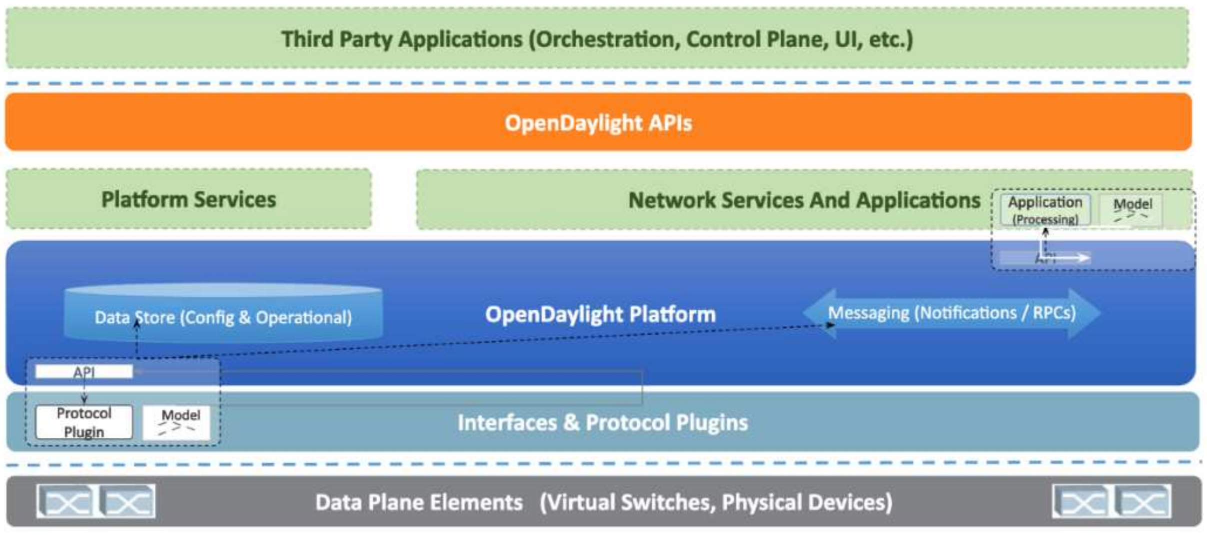
```
 "interface": {
    "GigabitEthernet": [
      {
        "name": "1",
        "mop": {
```

```
      "enabled": false,
      "sysid": false
    },
    "Cisco-IOS-XE-ethernet:negotiation": {
      "auto": true
    },
    "logging": {
      "event": {
        "link-status": [
          null
        ]
      }
    },
    "ip": {
      "address": {
        "primary": {
          "address": "10.3.100.100",
          "mask": "255.255.255.0"
        }
      }
    }
  },
  {
    "name": "2",
    "mop": {
      "enabled": false,
      "sysid": false
    },
    "Cisco-IOS-XE-ethernet:negotiation": {
      "auto": true
    },
    "logging": {
         "event": {
            "link-status": [
               null
            ]
         }
      },
      "ip": {
         "address": {
            "primary": {
               "address": "10.1.1.1",
               "mask": "255.255.255.0"
            }
         }
      },
      "description": "Cisco"
   },
```

Introduction to Open Daylight (ODL)

Open Daylight (ODL) is an open-source controller that can deploy the SDN model [15]. Furthermore, the controller can be modified and operated according to the network requirement using multiple plugins and applications [15]. The modular architecture of ODL provides scalability and automation advantages discussed earlier. The open-source ODL provides access to the controller with network programmability features and supports multiple vendors. It is easily deployable on any operating system supporting java.

The ODL is based on the MD-SAL (Model-Driven Service Abstraction Layer) architecture. This architecture provides the tool to exchange information between the YANG models used by the end network devices and applications [15]. YANG is a data modelling language that delivers a hierarchal data structure for different management protocols like NETCONF or RESTCONF to perform functions like network configuration, monitoring and fetching configuration. The network device vendors integrate the YANG models into the devices, and users can utilize the protocols like NETCONF or RESTCONF to use the YANG capabilities of the device to control and manage the network devices. The ODL uses multiple plugins and protocols which support the NBI and SBI as per the requirements and provides SDN-controlled functionality with modular architecture. The user can deploy the plugins per the network requirements, and each plugin works individually.



Third Party Applications (Orchestration, Control Plane, UI, etc.)
OpenDaylight APIs
Platform Services
Network Services And Applications
Application (Processing)
Model
API
Data Store (Config & Operational)
OpenDaylight Platform
Messaging (Notifications / RPCs)
API
Protocol Plugin
Model
Interfaces & Protocol Plugins
Data Plane Elements (Virtual Switches, Physical Devices)

Figure 10. ODL Architecture

https://www.opendaylight.org/what-we-do/current-release/sodium

The ODL controller uses its Northbound interface to flow the information between the application and the controller. The ODL's NBI also supports API calls, which applications can use to manage and control network devices. In addition, the ODL controller provides the option to install the plugins with multiple protocols, and the SBI of ODL uses those plugins to communicate with the end network devices. In this project, we will use one of the ODL Southbound plugins, which uses the NetConf protocol to communicate with the end network devices.

The following technologies are used by the Open Daylight Controller:

- **OSGI**

The OSGI framework, which enables the installation of bundles and packages' JAR files and binds bundles for information exchange, functions as the back-end of OpenDaylight. [16]

- **Karaf**

It serves the function of packaging and installing applications smoothly; this function is performed by the OSGI-based Karaf application container. [17]

- **YANG**

Applications Data, remote procedure calls, and notifications are modelled using the data modelling language YANG.

The below-mentioned components manage the Network Service functions to communicate with the network devices in the infrastructure topology in ODL:

Topology Manager: It stores the details and information of the end network devices and creates a root node in the operational data tree whenever any network device connects with the southbound connector. After the initial node installation, it continuously looks for updates in the form of notifications from the device or other components to keep the topology component up to date. [18]

Statistics Manager: This component stores the device's operational stats information. It collects and stores the device's current stats in an operational data tree. [18]

Inventory Manager: It keeps the latest connectivity information about any connected node and device to the ODL. It continuously tracks the connected devices and ensures the ODL inventory is correct and updated. [18]

The Working of MD-SAL

MD-SAL integrates the controller's different components and provides a standard process to modify the NBI, SBI and data structures used in different elements of ODL. MD-SAL uses the concept of the tree structure to represent messaging process within the controller to integrate with the yang models of the end network devices [17]. For example, the MD-SAL state data is in the form of a data tree with two types of data tree structures. The operational data tree contains the current noted state information of the network, and applications can observe the network's current state using the operational data tree. On the other hand, the config data tree contains the data inserted by the controller for required changes in the network.

MD-SAL uses the standard data modelling language YANG as the data model defining language in the ODL. Therefore, the MD-SAL defines the controller's modules and data structures in the ODL using the YANG models.

As mentioned earlier, the ODL uses different plugins on the NBI and SBI to integrate with the applications and the end network devices. In this project, we will use the NETCONF as the southbound plugin to create a communication channel between the network devices and the ODL controller. The diagram below shows when the end network devices initiate the NETCONF connection with the ODL. First, it connects with the SBI plugin, and further, that plugin updates the ODL controller modules. These updates are in the form of node update notifications.

The Inventory Manager installs the new node to the operational data structures. Next, Statistics Manager updates its information from the new information stored in the inventory database. Finally, using the latest inputs, the topology manager updates its connected topology data tree structures.

ODL's flexible architecture enables this controller to integrate with the new SDN technologies and applications. ODL provides the interfaces that enable the required control and management of the implemented network. It differs from other SDN options due to its MD-SAL and microservice architecture and support for a broad range of networking protocols.

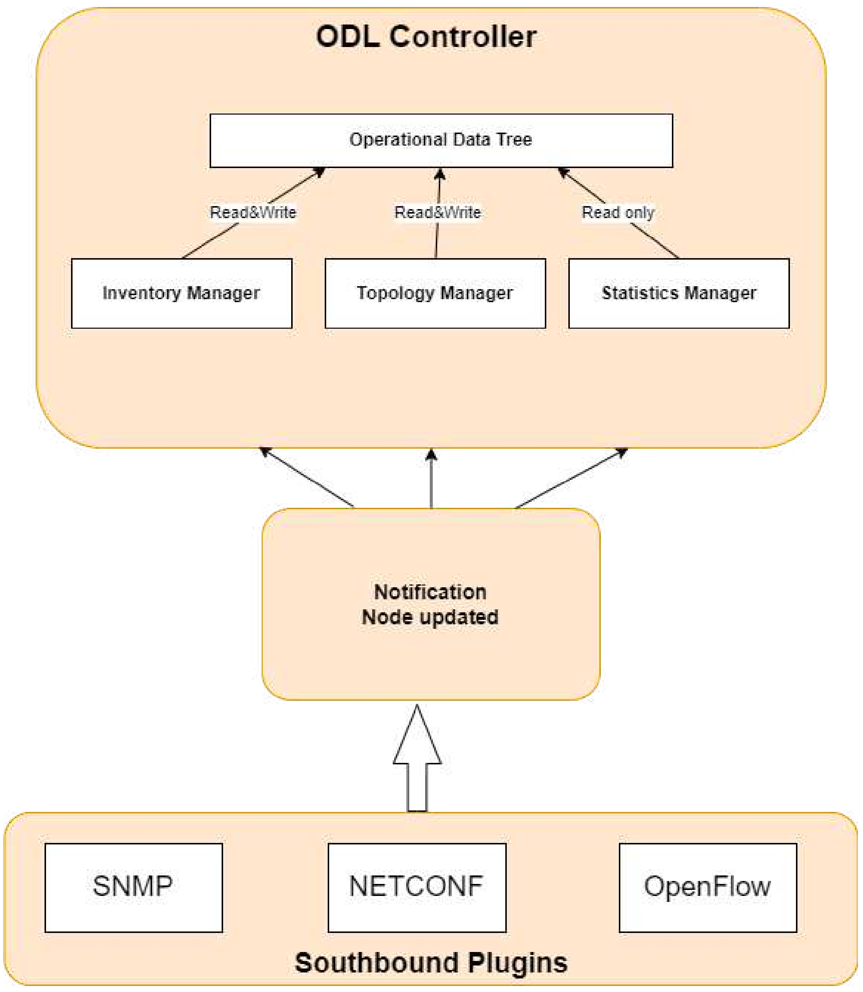


Figure 11. ODL Controller Traffic Flow Components

We will be using the ODL sulfur release in this project. As mentioned earlier, the ODL is built on the java language and uses Maven as a build tool. Therefore, the essential requirement to run an SDN project on the ODL is to have Java JDK 11 and Apache Maven 3.8.3 or later.

Open Daylight controller implementation

In this project, we will use the Ubuntu server 22.04.1 to implement the ODL sufur version. The requirements for the ODL installation are listed below:

1. Java JDK 11.

2. Setting JAVA_HOME.

3. ODL Sulfur zip file.

4. ODL installation.

Before starting the ODL installation, we must update the Ubuntu operating system. We can use the "apt-get update" command to update the available packages and the "apt-get -y upgrade" to install the new ones. The unzip should also be present in Ubuntu OS to decompress the ODL installation zip file.

The latest JAVA application must be on the Ubuntu server, and we can use the JAVA 11 JDK for the same purpose. The command to install the JAVA 11 JDK is "apt-get -y install openjdk-11-jre".

After installing the JAVA 11 JDK, we need to ensure that the application that requires the JAVA to run can find the JDK location to run the JAVA-related process. We can set the JAVA_HOME variable to point toward the JDK installation location for that purpose. So, it becomes mandatory to edit the bash resource file to set the JAVA_HOME value to the JAVA JDK location.

As we have already discussed, the ODL is an open-source controller. Therefore, we can download the installation file from the ODL website. Then, after downloading the zip file to the Ubuntu server, we can unzip the file and run the ODL with the "./bin/karaf" command from unzip location directory.

The ODL is a modular controller, and we have to install the features in the ODL as per the requirement. For example, as per our project requirement, we need to install the below-mentioned features:

odl-netconf-topology

odl-restconf

odl-netconf-connector-all

In the previous versions of the ODL, there was an option to install the UI for ODL called DLUX. However, that feature is not present in the Sulphur release and is only available for versions launched before 2019.

Integration of cisco devices with ODL

To integrate the Cisco router with ODL, we must configure the NETCONF on the router. Then, the ODL will access the router via SSH. In this project, we will implement the CSR1000v router, which runs the IOS-XE version.

Once we have the layer three reachability to the ODL controller, we can start configuring a user profile on the router with privilege level 15. The ODL will access the router through that profile. Therefore, it is essential to have SSH enabled on the router. Once the user profile is defined, we must enable the router's NETCONF. By default, the NETCONF protocol will use port 830 as per the RFC standard, but it can be changed if required. To connect the ODL with the router, the basic requirements are the router's IP address, username, password and NETCONF port number.

After completing the configuration on the router, we need to create a NETCONF connector on the ODL. Next, the ODL integrates the devices with NETCONF capabilities using the southbound NETCONF plugin. Using this NETCONF connector, the ODL can access the configurational and operational data stores, RPCs and notifications as MD-SAL mount points. In addition, this SBI plugin enables the applications and users connected with ODL to interact with the end devices.

To configure the NETCONF connector on the ODL, we can use RESTCONF to send the configuration data to the ODL. For the Northbound, this connector configuration mounts the NETCONF server in the config-subsystem of ODL so that we can also use RESTCONF to access the config-subsystem of ODL. We will use the RESTCONF on the NBI side as it is easier to use, but with the configured NETCONF connector, the RESTCONF data will be going via NETCONF only.

As mentioned earlier, the ODL karaf feature old-net conf-connector-all is a must to implement the net conf connector on the ODL. We can use the RESTCONF POST operation to send all the required details of the Cisco device to the ODL. We can define the node name of the connector in the POST message, and once the ODL receives this request, it will configure the NETCONF connector with the requested node name and end device access information.

The XML body should include the following details:

1. **Node-id** - The node-id will be the name and identification of the NETCONF connector in the ODL's config-subsystem.

2. **Host Address** - It defines the device's IP, which is the IP address of the Cisco router in our case.

3. **Port** - It is the port used by the NETCONF device. It should be the same as the NETCONF port defined in the end device configuration.

4. **Username** - The username defined on the NETCONF devices. This will be used to create the session with the end device.

5. **Password** - The password defined on the NETCONF device for the above username. This username and password should match the configuration on the NETCONF device.

6. **TCP-only** - This attribute selects the session formation over TCP or SSL. If set to true, it will use the TCP; if False, it will use the SSL.

The information mentioned above in the configuration part is mandatory to set up the net conf connector, but some additional configuration options are also present, which we can use as per requirement. Below are the additional configuration options for the netconf-connector.

- **Keepalive-delay**: It defines the timeout interval for keep-alive RPCs. The ODL Netconf-connector sends the keep-alive to the network device to ensure idle session connectivity. By default, value is 120 seconds.
- **Sleep-factor**: This is considered the back-off factor to improve the delay between connection attempts. The default value is 1.5.
- **Max-connection-attempts**: It represents the maximum number of connection attempts. The default value is 0, which means an infinite times attempts.
- **Between-attempts-timeout-millis**: This configuration defines the initial timeout value between the attempts in milliseconds. The default value is 2000 milliseconds.
- **Schema-cache-directory**: This configuration defines the destination schema directory for the YANG data files fetched from the NETCONF network device. By default, the directory location is $ODL_ROOT/cache/schema. We can change it as per the requirement using this configuration element.
- **Reconnect-on-changed-schema**: This option can be set to true or false. True means the ODL netconf-connector would reconnect or disconnect automatically if there is any NETCONF network device schemas change. False is vice versa.
- **Connection-timeout-millis**: This represents the actual timeout value in milliseconds for connection establishment. The default value is 20000 milliseconds.
- **Default-request-timeout-millis**: This element defines the timeout value for the request operations. The ongoing request operation will be blocked once the time is reached. The default value is 60000 milliseconds.

After pushing the above details in an XML file to the ODL via RESTCONF, ODL will create a new NETCONF connector, which will connect with the NETCONF device mentioned in the XML body using the defined username and password. Once the new net conf connector is created successfully, the metadata will be created in the operational data store of the ODL network-topology subtree. The network device

The NETCONF network device will share its YANG capabilities by sending hello messages to the ODL. Then, the ODL will fetch all the YANG data provided by the device, which will be used to determine the schema context of the remote NETCONF device. Finally, we can send the GET request to the ODL using RESTCONF to fetch the current status and capabilities of the connected NETCONF device.

Configuration of Cisco devices using NETCONF

After integrating the Cisco router with the ODL via the NETCONF connector, we can fetch the device configuration using the RESTCONF GET operation by the application on the NBI side and the PUT operation to push the configuration to the Cisco device.

The ODL stores the data fetched from the device in operation and config data stores, and we can retrieve the required data from these data stores.

We will use the same API to fetch the configuration and even to push the configuration to the device. The only difference will be the RECONF GET and PUT operation. Also, we will edit the XML body per the new configuration requirement during the PUT operation to push the required configuration to the device. Finally, we must define the node name in the API call to target the required NETCONF remote device.

We will use the below-mentioned API call to initiate the GET and PUT operation with the ODL controller:

http://localhost:8181/restconf/config/network-topology:network-topology/topology/topology-netconf/node/node-id/yang-ext:mount/Cisco-IOS-XE-native:native/

The initial part of the API defines the ODL controller host details, and the ODL is accessible at localhost:8181. The next part defines that the restconf is used, and we are targetting the config data store with /config/ element. Further, we must define the modules (/network-topology:network-topology/topology/topology-netconf/node/), which we require to access in ODL to reach the node configured. The (node-id)

element here defines the node-id or name of the remote NETCONF device. The last part (/yang-ext:mount/Cisco-IOS-XE-native:native/) refers to the YANG mount point and the targetted YANG capabilities data store of the remote NETCONF device.

Once we use this API with GET operation, the output will be the complete native configuration of the Cisco device in the XML format. We will use the same output to edit the configuration or to add the new configuration. After editing the XML body as per the requirement, we can use the same API with the new XML body and PUT operation to push the new configuration to the device.

The ODL will then use the mounted netconf-connector and access the Cisco device via SSH. The new data in the XML body of the API call will change the NETCONF YANG data stored in the device with the new configuration, and this is how we can configure the Cisco device using the ODL controller.

To fetch the operational data or the current statistics of the connected NETCONF network device. We will use the below-mentioned API. In this, we are targeting the operational data store of the ODL instead of the config data store.

http://localhost:8181/restconf/operational/network-topology:network-topology/topology/topology-netconf/node/node-id/yang-ext:mount/Cisco-IOS-XE-native:native/

Conclusion

Finding the automation compatibility of devices already in the traditional network infrastructure is challenging. However, shifting to the SDN architecture instead of staying on the traditional network is more advantageous from the long-term perspective. SDN is more flexible and easily scalable, causing the operational and expansion costs extremely low compared to the traditional networking architecture. In addition, centralized control of the complete network makes it significantly easier to integrate with the new applications and software coming into the network industry for different purposes, increasing productivity and security.

This project aims to deploy an open-source SDN controller and integrate the same with the Cisco devices to test the compatibility and workflow with the ODL controller. Furthermore, this project clears the requirements and steps to integrate the Cisco devices with the ODL (Open Daylight) controller and configuration of devices via NETCONF. The main issue faced during the integration was finding the exact RPC of the compatible YANG model in the device to change or push the required configuration. There was a DLUX feature in the earlier versions of the ODL, which provides access to the YANG UI of the mounted node. It is easier to get the RPC of the supported YANG models with the DLUX feature, but this feature was removed after the release of the ODL oxygen version, and we are using the latest sulfur version in this project. We have targeted the native configuration yang model of the cisco device in this project and found the correct RPC for the same model by testing manually. The native YANG model covers the overall configuration of the device, and we need to define the complete configuration whenever we need to make any changes to the configuration. Therefore, there should be a way to find the compatible RPC for the specific YANG model to narrow down the requirement to push the complete device configuration every time we need to make any changes in the configuration. Finally, the testing and implementation in this project draft the use of NETCONF and ODL plugins to integrate Cisco devices with the open-source controller and eventually outline the advantages of moving to SDN architecture from the traditional network.

References

[1] "Software-Defined Networking," [Online]. Available: https://www.cisco.com/c/en_ca/solutions/software-defined-networking/overview.html.

[2] J. H. Muhammad Raza, "What is Software Defined Networking? SDN Explained," [Online]. Available: https://www.bmc.com/blogs/software-defined-networking/.

[3] "What is Software-Defined Networking (SDN)?," [Online]. Available: https://www.vmware.com/topics/glossary/content/software-defined-networking.html.

[4] "What is Software-Defined Networking?," [Online]. Available: https://www.cdw.com/content/cdw/en/articles/datacenter/what-is-software-defined-networking.

[5] Satyabrata_Jena, "Difference between Control Plane and Data Plane," [Online]. Available: https://www.geeksforgeeks.org/difference-between-control-plane-and-data-plane/.

[6] C. Craven, "What Is Software Defined Networking (SDN)? Definition," [Online]. Available: https://www.sdxcentral.com/networking/sdn/definitions/what-the-definition-of-software-defined-networking-sdn/.

[7] W. Werks, "southbound-vs-northbound-sdn-what-are-differences," [Online]. Available: https://www.webwerks.in/blogs/southbound-vs-northbound-sdn-what-are-differences.

[8] "How REST APIs work," [Online]. Available: https://www.ibm.com/topics/rest-apis.

[9] M. Wasserman, "Using the NETCONF Configuration Protocol over Secure SHell (SSH)," [Online]. Available: https://www.rfc-editor.org/rfc/rfc4742.

[10] M. Kashin, "Getting Started With NETCONF and YANG on Cisco IOS XE," [Online]. Available: https://networkop.co.uk/blog/2017/01/25/netconf-intro/.

[11] k. okasha, "Network Automation and the Rise of NETCONF," [Online]. Available: https://medium.com/@k.okasha/network-automation-and-the-rise-of-netconf-e96cc33fe28.

[12] C. Rizos, "Why use NETCONF/YANG when you can use SNMP and CLI?," [Online]. Available: https://www.snmpcenter.com/why-use-netconf/.

[13] A. Bierman, "RESTCONF Protocol," [Online]. Available: https://www.rfc-editor.org/rfc/rfc8040.

[14] E. M. Bjorklund, "YANG - A Data Modeling Language for NETCONF," [Online]. Available: https://www.rfc-editor.org/rfc/rfc6020.

[15] "Platform Overview," [Online]. Available: https://www.opendaylight.org/about/platform-overview.

[16] "OpenDaylight Controller Overview," [Online]. Available: https://nexus.opendaylight.org/content/sites/site/org.opendaylight.docs/master/userguide/manuals/userguide/bk-user-guide/content/_opendaylight_controller_overview.html.

[17] "OpenDaylight concepts and tools," [Online]. Available: https://docs.opendaylight.org/en/stable-sulfur/getting-started-guide/concepts_and_tools.html.

[18] P. Karpov, "What's in OpenDaylight?," [Online]. Available: https://www.mirantis.com/blog/whats-opendaylight/.