SNMP4SDN

Release master

Feb 19, 2019

Contents

CHAPTER 1

SNMP4SDN Developer Guide

1.1 Overview

We propose a southbound plugin that can control the off-the-shelf commodity Ethernet switches for the purpose of building SDN using Ethernet switches. For Ethernet switches, forwarding table, VLAN table, and ACL are where one can install flow configuration on, and this is done via SNMP and CLI in the proposed plugin. In addition, some settings required for Ethernet switches in SDN, e.g., disabling STP and flooding, are proposed.

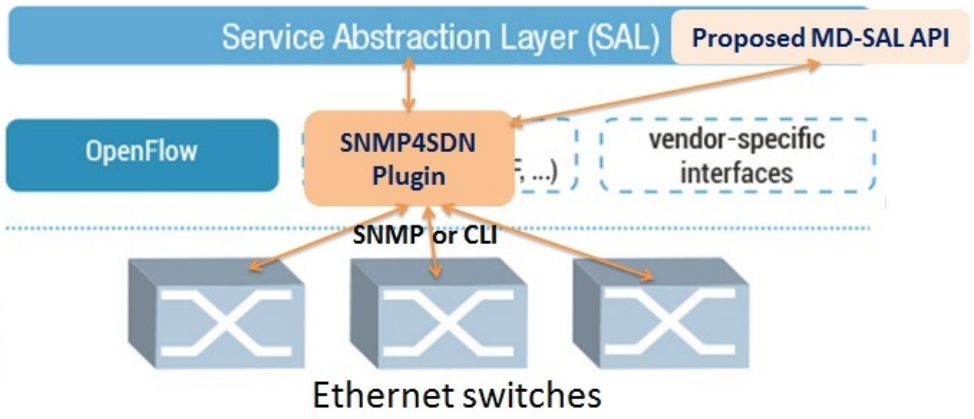

Fig. 1: SNMP4SDN as an OpenDaylight southbound plugin

1.2 Architecture

The modules in the plugin are depicted as the following figure.

- AclService: add/remove ACL profile and rule on the switches.

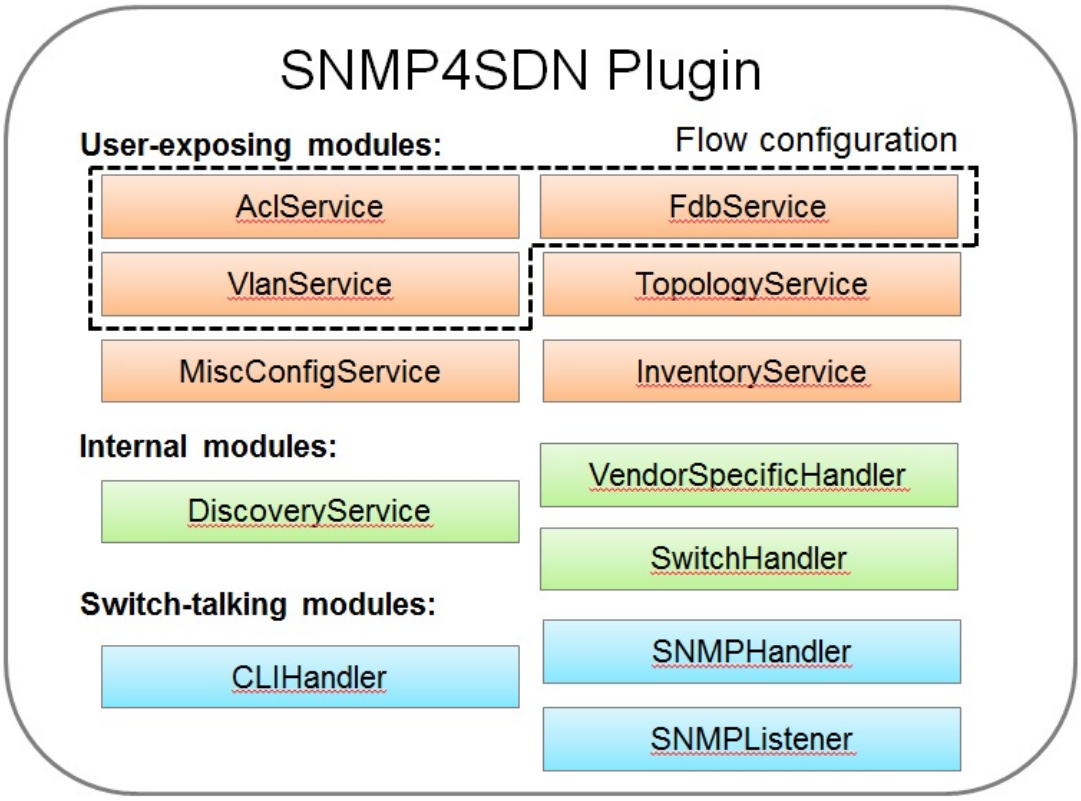

Fig. 2: Modules in the SNMP4SDN Plugin

- FdbService: add/modify/remove FDB table entry on the switches.
- VlanService: add/modify/remove VLAN table entry on the switches.
- TopologyService: query and acquire the subnet topology.
- InventoryService: acquire the switches and their ports.
- DiscoveryService: probe and resolve the underlying switches as well as the port pairs connecting the switches. The probing is realized by SNMP queries. The updates from discovery will also be reflected to the Topology-Service.
- MiscConfigService: do kinds of settings on switches
  - Supported STP and ARP settings such as enable/disable STP, get port's STP state, get ARP table, set ARP entry, and others
- VendorSpecificHandler: to assist the flow configuration services to call the switch-talking modules with correct parameters value and order.
- Switch-talking modules
  - For the services above, when they need to read or configure the underlying switches via SNMP or CLI, these queries are dealt with the modules SNMPHandler and CLIHandler which directly talk with the switches. The SNMPListener is to listen to snmp trap such as link up/down event or switch on/off event.

1.3 Design

In terms of the architecture of the SNMP4SDN Plugin's features, the features include flow configuration, topology discovery, and multi-vendor support. Their architectures please refer to Wiki (Developer Guide - Design).

1.4 Installation and Configuration Guide

- Please refer to the *Getting Started Guide* in https://www.opendaylight.org/downloads, find the SNMP4SDN section.
- For the latest full guide, please refer to Wiki (Installation Guide, User Guide - Configuration).

1.5 Tutorial

- For the latest full guide, please refer to Wiki (User Guide - Tutorial).

1.6 Programmatic Interface(s)

SNMP4SDN Plugin exposes APIs via MD-SAL with YANG model. The methods (RPC call) and data structures for them are listed below.

1.6.1 TopologyService

- RPC call
  - get-edge-list

  - get-node-list
  - get-node-connector-list
  - set-discovery-interval (given interval time in seconds)
  - rediscover
- Data structure
  - node: composed of node-id, node-type
  - node-connector: composed of node-connector-id, node-connector-type, node
  - topo-edge: composed of head-node-connector-id, head-node-connector-type, head-node-id, head-node-type, tail-node-connector-id, tail-node-connector-type, tail-node-id, tail-node-type

1.6.2 VlanService

- RPC call
  - add-vlan (given node ID, VLAN ID, VLAN name)
  - add-vlan-and-set-ports (given node ID, VLAN ID, VLAN name, tagged ports, untagged ports)
  - set-vlan-ports (given node ID, VLAN ID, tagged ports, untagged ports)
  - delete-vlan (given node ID, VLAN ID)
  - get-vlan-table (given node ID)

1.6.3 AclService

- RPC call
  - create-acl-profile (given node ID, acl-profile-index, acl-profile)
  - del-acl-profile (given node ID, acl-profile-index)
  - set-acl-rule (given node ID, acl-index, acl-rule)
  - del-acl-rule (given node ID, acl-index)
  - clear-acl-table (given node ID)
- Data structure
  - acl-profile-index: composed of profile-id, profile name
  - acl-profile: composed of acl-layer, vlan-mask, src-ip-mask, dst-ip-mask
  - acl-layer: IP or ETHERNET
  - acl-index: composed of acl-profile-index, acl-rule-index
  - acl-rule-index: composed of rule-id, rule-name
  - acl-rule: composed of port-list, acl-layer, acl-field, acl-action
  - acl-field: composed of vlan-id, src-ip, dst-ip
  - acl-action: PERMIT or DENY

1.6.4 FdbService

- RPC call
  - set-fdb-entry (given fdb-entry)
  - del-fdb-entry (given node-id, vlan-id, dest-mac-adddr)
  - get-fdb-entry (given node-id, vlan-id, dest-mac-adddr)
  - get-fdb-table (given node-id)
- Data structure
  - fdb-entry: composed of node-id, vlan-id, dest-mac-addr, port, fdb-entry-type
  - fdb-entry-type: OTHER/INVALID/LEARNED/SELF/MGMT

1.6.5 MiscConfigService

- RPC call
  - set-stp-port-state (given node-id, port, is_nable)
  - get-stp-port-state (given node-id, port)
  - get-stp-port-root (given node-id, port)
  - enable-stp (given node-id)
  - disable-stp (given node-id)
  - delete-arp-entry (given node-id, ip-address)
  - set-arp-entry (given node-id, arp-entry)
  - get-arp-entry (given node-id, ip-address)
  - get-arp-table (given node-id)
- Data structure
  - stp-port-state: DISABLE/BLOCKING/LISTENING/LEARNING/FORWARDING/BROKEN
  - arp-entry: composed of ip-address and mac-address

1.6.6 SwitchDbService

- RPC call
  - reload-db (The following 4 RPC implemention is TBD)
  - add-switch-entry
  - delete-switch-entry
  - clear-db
  - update-db
- Data structure
  - switch-info: compose of node-ip, node-mac, community, cli-user-name, cli-password, model

1.7 Help

- SNMP4SDN Wiki
- SNMP4SDN Mailing List (user, developer)
- Latest troubleshooting in Wiki

CHAPTER 2

SNMP4SDN User Guide

2.1 Overview

We propose a southbound plugin that can control the off-the-shelf commodity Ethernet switches for the purpose of building SDN using Ethernet switches. For Ethernet switches, forwarding table, VLAN table, and ACL are where one can install flow configuration on, and this is done via SNMP and CLI in the proposed plugin. In addition, some settings required for Ethernet switches in SDN, e.g., disabling STP and flooding, are proposed.

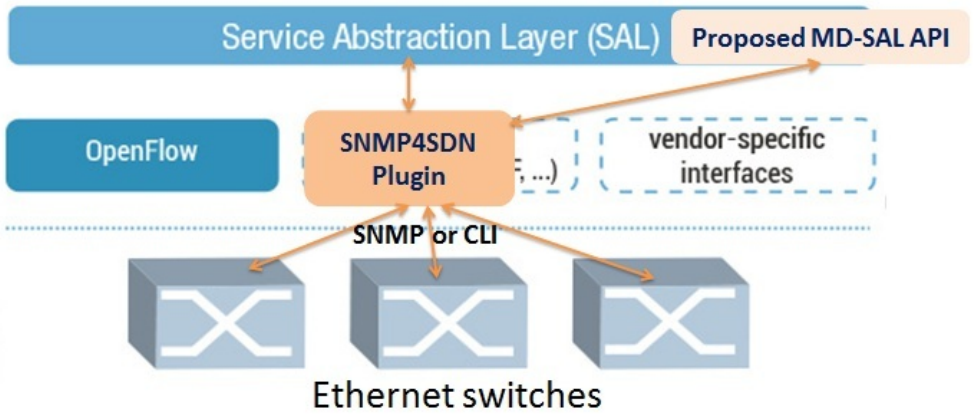

Fig. 1: SNMP4SDN as an OpenDaylight southbound plugin

2.2 Configuration

Just follow the steps:

2.2.1 Prepare the switch list database file

A sample is here, and we suggest to save it as */etc/snmp4sdn_swdb.csv* so that SNMP4SDN Plugin can automatically load this file. Note that the first line is title and should not be removed.

2.2.2 Prepare the vendor-specific configuration file

A sample is here, and we suggest to save it as */etc/snmp4sdn_VendorSpecificSwitchConfig.xml* so that SNMP4SDN Plugin can automatically load this file.

2.3 Install SNMP4SDN Plugin

If using SNMP4SDN Plugin provided in OpenDaylight release, just do the following from the Karaf CLI:

```
feature:repo-add mvn:org.opendaylight.snmp4sdn/features-snmp4sdn/0.8.1-SNAPSHOT/xml/
↪features
```

```
feature:install odl-snmp4sdn-all
```

2.4 Troubleshooting

2.4.1 Installation Troubleshooting

Feature installation failure

When trying to install a feature, if the following failure occurs:

```
Error executing command: Could not start bundle ...
Reason: Missing Constraint: Require-Capability: osgi.ee; filter="(&(osgi.
↪ee=JavaSE)(version=1.7))"
```

A workaround: exit Karaf, and edit the file <karaf_directory>/etc/config.properties, remove the line *${services-${karaf.framework}}* and the ", " in the line above.

2.4.2 Runtime Troubleshooting

Problem starting SNMP Trap Interface

It is possible to get the following exception during controller startup. (The error would not be printed in Karaf console, one may see it in <karaf_directory>/data/log/karaf.log)

```
2014-01-31 15:00:44.688 CET [fileinstall-./plugins] WARN  o.o.snmp4sdn.internal.
↪SNMPListener - Problem starting SNMP Trap Interface: {}
 java.net.BindException: Permission denied
        at java.net.PlainDatagramSocketImpl.bind0(Native Method) ~[na:1.7.0_51]
        at java.net.AbstractPlainDatagramSocketImpl.
↪bind(AbstractPlainDatagramSocketImpl.java:95) ~[na:1.7.0_51]
        at java.net.DatagramSocket.bind(DatagramSocket.java:376) ~[na:1.7.0_51]
```

(continues on next page)

(continued from previous page)

```
        at java.net.DatagramSocket.<init>(DatagramSocket.java:231) ~[na:1.7.0_51]
        at java.net.DatagramSocket.<init>(DatagramSocket.java:284) ~[na:1.7.0_51]
        at java.net.DatagramSocket.<init>(DatagramSocket.java:256) ~[na:1.7.0_51]
        at org.snmpj.SNMPTrapReceiverInterface.<init>(SNMPTrapReceiverInterface.
↪java:126) ~[org.snmpj-1.4.3.jar:na]
        at org.snmpj.SNMPTrapReceiverInterface.<init>(SNMPTrapReceiverInterface.
↪java:99) ~[org.snmpj-1.4.3.jar:na]
        at org.opendaylight.snmp4sdn.internal.SNMPListener.<init>(SNMPListener.
↪java:75) ~[bundlefile:na]
        at org.opendaylight.snmp4sdn.core.internal.Controller.start(Controller.
↪java:174) [bundlefile:na]
...
```

This indicates that the controller is being run as a user which does not have sufficient OS privileges to bind the SNMPTRAP port (162/UDP)

2.4.3 Switch list file missing

The SNMP4SDN Plugin needs a switch list file, which is necessary for topology discovery and should be provided by the administrator (so please prepare one for the first time using SNMP4SDN Plugin, here is the sample). The default file path is /etc/snmp4sdn_swdb.csv. SNMP4SDN Plugin would automatically load this file and start topology discovery. If this file is not ready there, the following message like this will pop up:

```
2016-02-02 04:21:52,476 | INFO| Event Dispatcher | CmethUtil                          |␣
↪466 - org.opendaylight.snmp4sdn - 0.3.0.SNAPSHOT | CmethUtil.readDB() err: {}
java.io.FileNotFoundException: /etc/snmp4sdn_swdb.csv (No such file or directory)
    at java.io.FileInputStream.open0(Native Method)[:1.8.0_65]
    at java.io.FileInputStream.open(FileInputStream.java:195)[:1.8.0_65]
    at java.io.FileInputStream.<init>(FileInputStream.java:138)[:1.8.0_65]
    at java.io.FileInputStream.<init>(FileInputStream.java:93)[:1.8.0_65]
    at java.io.FileReader.<init>(FileReader.java:58)[:1.8.0_65]
    at org.opendaylight.snmp4sdn.internal.util.CmethUtil.readDB(CmethUtil.java:66)
    at org.opendaylight.snmp4sdn.internal.util.CmethUtil.<init>(CmethUtil.java:43)
...
```

2.5 Configuration

Just follow the steps:

2.5.1 1. Prepare the switch list database file

A sample is here, and we suggest to save it as */etc/snmp4sdn_swdb.csv* so that SNMP4SDN Plugin can automatically load this file.

Note: The first line is title and should not be removed.

2.5.2 2. Prepare the vendor-specific configuration file

A sample is here, and we suggest to save it as */etc/snmp4sdn_VendorSpecificSwitchConfig.xml* so that SNMP4SDN Plugin can automatically load this file.

2.5.3 3. Install SNMP4SDN Plugin

If using SNMP4SDN Plugin provided in OpenDaylight release, just do the following:

Launch Karaf in Linux console:

```
cd <Boron_controller_directory>/bin
(for example, cd distribution-karaf-x.x.x-Boron/bin)
```

```
./karaf
```

Then in Karaf console, execute:

```
feature:install odl-snmp4sdn-all
```

2.5.4 4. Load switch list

For initialization, we need to feed SNMP4SDN Plugin the switch list. Actually SNMP4SDN Plugin automatically try to load the switch list at /etc/snmp4sdn_swdb.csv if there is. If so, this step could be skipped. In Karaf console, execute:

```
snmp4sdn:ReadDB <switch_list_path>
(For example, snmp4sdn:ReadDB /etc/snmp4sdn_swdb.csv)
(in Windows OS, For example, snmp4sdn:ReadDB D://snmp4sdn_swdb.csv)
```

A sample is here, and we suggest to save it as */etc/snmp4sdn_swdb.csv* so that SNMP4SDN Plugin can automatically load this file.

Note: The first line is title and should not be removed.

2.5.5 5. Show switch list

```
snmp4sdn:PrintDB
```

2.6 Tutorial

2.6.1 Topology Service

Execute topology discovery

The SNMP4SDN Plugin automatically executes topology discovery on startup. One may use the following commands to invoke topology discovery manually. Note that you may need to wait for seconds for itto complete.

Note: Currently, one needs to manually execute *snmp4sdn:TopoDiscover* first (just once), then later the automatic topology discovery can be successful. If switches change (switch added or removed), *snmp4sdn:TopoDiscover* is also required. A future version will fix it to eliminate these requirements.

```
snmp4sdn:TopoDiscover
```

If one like to discover all inventory (i.e. switches and their ports) but not edges, just execute "TopoDiscoverSwitches":

```
snmp4sdn:TopoDiscoverSwitches
```

If one like to only discover all edges but not inventory, just execute "TopoDiscoverEdges":

```
snmp4sdn:TopoDiscoverEdges
```

You can also trigger topology discovery via the REST API by using `curl` from the Linux console (or any other REST client):

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/
↪topology:rediscover
```

You can change the periodic topology discovery interval via a REST API:

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/topology:set-
↪discovery-interval -d "{"input":{"interval-second":'<interval_time>'}}"
For example, set the interval as 15 seconds:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/topology:set-
↪discovery-interval -d "{"input":{"interval-second":'15'}}"
```

Show the topology

SNMP4SDN Plugin supports to show topology via REST API:

- Get topology

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/
↪topology:get-edge-list
```

- Get switch list

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/
↪topology:get-node-list
```

- Get switches' ports list

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/
↪topology:get-node-connector-list
```

- The three commands above are just for user to get the latest topology discovery result, it does not trigger SNMP4SDN Plugin to do topology discovery.

- To trigger SNMP4SDN Plugin to do topology discover, as described in aforementioned *Execute topology discovery*.

2.6.2 Flow configuration

FDB configuration

SNMP4SDN supports to add entry on FDB table via REST API:

- Get FDB table

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/fdb:get-fdb-table -d "{input:{"node-id":<switch-mac-address-in-
↪number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/fdb:get-fdb-
↪table -d "{input:{"node-id":158969157063648}}"
```

- Get FDB table entry

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/fdb:get-fdb-entry -d "{input:{"node-id":<switch-mac-address-in-
↪number>, "vlan-id":<vlan-id-in-number>, "dest-mac-addr":<destination-mac-
↪address-in-number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/fdb:get-fdb-
↪entry -d "{input:{"node-id":158969157063648, "vlan-id":1, "dest-mac-addr
↪":158969157063648}}"
```

- Set FDB table entry

  (Notice invalid value: (1) non unicast mac (2) port not in the VLAN)

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/fdb:set-fdb-entry -d "{input:{"node-id":<switch-mac-address-in-
↪number>, "vlan-id":<vlan-id-in-number>, "dest-mac-addr":<destination-mac-
↪address-in-number>, "port":<port-in-number>, "type":'<type>'}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/fdb:set-fdb-
↪entry -d "{input:{"node-id":158969157063648, "vlan-id":1, "dest-mac-addr
↪":187649984473770, "port":23, "type":'MGMT'}}"
```

- Delete FDB table entry

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/fdb:del-fdb-entry -d "{input:{"node-id":<switch-mac-address-in-
↪number>, "vlan-id":<vlan-id-in-number>, "dest-mac-addr":<destination-mac-
↪address-in-number>}}"
```

(continues on next page)

(continued from previous page)

```
For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/fdb:del-fdb-
↪entry -d "{input:{"node-id":158969157063648, "vlan-id":1, "dest-mac-addr
↪":187649984473770}}"
```

VLAN configuration

SNMP4SDN supports to add entry on VLAN table via REST API:

- Get VLAN table

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/vlan:get-vlan-table -d "{input:{node-id:<switch-mac-address-in-
↪number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/vlan:get-
↪vlan-table -d "{input:{node-id:158969157063648}}"
```

- Add VLAN

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/vlan:add-vlan -d "{"input":{"node-id":<switch-mac-address-in-number>,
↪ "vlan-id":<vlan-id-in-number>, "vlan-name":'<vlan-name>'}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/vlan:add-
↪vlan -d "{"input":{"node-id":158969157063648, "vlan-id":123, "vlan-name":'v123'}
↪}"
```

- Delete VLAN

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/vlan:delete-vlan -d "{"input":{"node-id":<switch-mac-address-in-
↪number>, "vlan-id":<vlan-id-in-number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/vlan:delete-
↪vlan -d "{"input":{"node-id":158969157063648, "vlan-id":123}}"
```

- Add VLAN and set ports

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/vlan:add-vlan-and-set-ports -d "{"input":{"node-id":<switch-mac-
↪address-in-number>, "vlan-id":<vlan-id-in-number>, "vlan-name":'<vlan-name>',
↪"tagged-port-list":'<tagged-ports-separated-by-comma>', "untagged-port-list":'
↪<untagged-ports-separated-by-comma>'}}"
```

(continues on next page)

(continued from previous page)

```
For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/vlan:add-
↪vlan-and-set-ports -d "{"input":{"node-id":158969157063648, "vlan-id":123,
↪"vlan-name":'v123', "tagged-port-list":'1,2,3', "untagged-port-list":'4,5,6'}}"
```

- Set VLAN ports

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/vlan:set-vlan-ports -d "{"input":{"node-id":<switch-mac-address-in-
↪number>, "vlan-id":<vlan-id-in-number>, "tagged-port-list":'<tagged-ports-
↪separated-by-comma>', "untagged-port-list":'<untagged-ports-separated-by-comma>
↪'}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/vlan:set-
↪vlan-ports -d "{"input":{"node-id":"158969157063648", "vlan-id":"123", "tagged-
↪port-list":'4,5', "untagged-port-list":'2,3'}}"
```

ACL configuration

SNMP4SDN supports to add flow on ACL table via REST API. However, it is so far only implemented for the D-Link DGS-3120 switch.

ACL configuration via CLI is vendor-specific, and SNMP4SDN will support configuration with vendor-specific CLI in future release.

To do ACL configuration using the REST APIs, use commands like the following:

- Clear ACL table

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/acl:clear-acl-table -d "{"input":{"nodeId":<switch-mac-address-in-
↪number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/acl:clear-
↪acl-table -d "{"input":{"nodeId":158969157063648}}"
```

- Create ACL profile (IP layer)

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/acl:create-acl-profile -d "{input:{"nodeId":<switch-mac-address-in-
↪number>,"profile-id":<profile_id_in_number>,"profile-name":'<profile_name>',
↪"acl-layer":'IP',"vlan-mask":<vlan_mask_in_number>,"src-ip-mask":'<src_ip_mask>
↪',"dst-ip-mask":"<destination_ip_mask>"}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/acl:create-
↪acl-profile -d "{input:{"nodeId":158969157063648,"profile-id":1,"profile-name":
↪'profile_1',"acl-layer":'IP',"vlan-mask":1,"src-ip-mask":'255.255.0.0',"dst-ip-
↪mask":'255.255.255.255'}}"
```

(continues on next page)

(continued from previous page)

- Create ACL profile (MAC layer)

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/acl:create-acl-profile -d "{input:{"nodeId":<switch-mac-address-in-
↪number>,"profile-id":<profile_id_in_number>,"profile-name":'<profile_name>',
↪"acl-layer":'ETHERNET',"vlan-mask":<vlan_mask_in_number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/acl:create-
↪acl-profile -d "{input:{"nodeId":158969157063648,"profile-id":2,"profile-name":
↪'profile_2',"acl-layer":'ETHERNET',"vlan-mask":4095}}"
```

- Delete ACL profile

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/acl:del-acl-
↪profile -d "{input:{"nodeId":<switch-mac-address-in-number>,"profile-id":
↪<profile_id_in_number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/acl:del-acl-
↪profile -d "{input:{"nodeId":158969157063648,"profile-id":1}}"
```

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/acl:del-acl-profile -d "{input:{"nodeId":<switch-mac-address-in-
↪number>,"profile-name":"<profile_name>"}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/acl:del-acl-
↪profile -d "{input:{"nodeId":158969157063648,"profile-name":'profile_2'}}"
```

- Set ACL rule

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/acl:set-acl-rule -d "{input:{"nodeId":<switch-mac-address-in-number>,
↪"profile-id":<profile_id_in_number>,"profile-name":'<profile_name>',"rule-id":
↪<rule_id_in_number>,"port-list":[<port_number>,<port_number>,...],"acl-layer":'
↪<acl_layer>',"vlan-id":<vlan_id_in_number>,"src-ip":"<src_ip_address>","dst-ip":
↪'<dst_ip_address>',"acl-action":'<acl_action>'}}"
(<acl_layer>: IP or ETHERNET)
(<acl_action>: PERMIT as permit, DENY as deny)

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/acl:set-acl-
↪rule -d "{input:{"nodeId":158969157063648,"profile-id":1,"profile-name":
↪'profile_1',"rule-id":1,"port-list":[1,2,3],"acl-layer":'IP',"vlan-id":2,"src-ip
↪":'1.1.1.1',"dst-ip":'2.2.2.2',"acl-action":'PERMIT'}}"
```

- Delete ACL rule

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/acl:del-acl-rule -d "{input:{"nodeId":<switch-mac-address-in-number>,
↪"profile-id":<profile_id_in_number>,"profile-name":'<profile_name>',"rule-id":
↪<rule_id_in_number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/acl:del-acl-
↪rule -d "{input:{"nodeId":158969157063648,"profile-id":1,"profile-name":
↪'profile_1',"rule-id":1}}"
```

2.6.3 Special configuration

SNMP4SDN supports setting the following special configurations via REST API:

- Set STP port state

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/config:set-stp-port-state -d "{input:{"node-id":<switch-mac-address-
↪in-number>, "port":<port_number>, enable:<true_or_false>}}"
(true: enable, false: disable)

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/config:set-
↪stp-port-state -d "{input:{"node-id":158969157063648, "port":2, enable:false}}"
```

- Get STP port state

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/config:get-stp-port-state -d "{input:{"node-id":<switch-mac-address-
↪in-number>, "port":<port_number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/config:get-
↪stp-port-state -d "{input:{"node-id":158969157063648, "port":2}}"
```

- Get STP port root

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/config:get-stp-port-root -d "{input:{"node-id":<switch-mac-address-
↪in-number>, "port":<port_number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/config:get-
↪stp-port-root -d "{input:{"node-id":158969157063648, "port":2}}"
```

- Enable STP

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/config:enable-stp -d "{input:{"node-id":<switch-mac-address-in-
↪number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/
↪config:enable-stp -d "{input:{"node-id":158969157063648}}"
```

- Disable STP

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/config:disable-stp -d "{input:{"node-id":<switch-mac-address-in-
↪number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/
↪config:disable-stp -d "{input:{"node-id":158969157063648}}"
```

- Get ARP table

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/config:get-arp-table -d "{input:{"node-id":<switch-mac-address-in-
↪number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/config:get-
↪arp-table -d "{input:{"node-id":158969157063648}}"
```

- Set ARP entry

  (Notice to give IP address with subnet prefix)

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/config:set-arp-entry -d "{input:{"node-id":<switch-mac-address-in-
↪number>, "ip-address":'<ip_address>', "mac-address":<mac_address_in_number>}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/config:set-
↪arp-entry -d "{input:{"node-id":158969157063648, "ip-address":'10.217.9.9',
↪"mac-address":1}}"
```

- Get ARP entry

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/config:get-arp-entry -d "{input:{"node-id":<switch-mac-address-in-
↪number>, "ip-address":'<ip_address>'}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/config:get-
↪arp-entry -d "{input:{"node-id":158969157063648, "ip-address":'10.217.9.9'}}"
```

(continues on next page)

(continued from previous page)

- Delete ARP entry

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://<controller_ip_address>:8181/restconf/
↪operations/config:delete-arp-entry -d "{input:{"node-id":<switch-mac-address-in-
↪number>, "ip-address":'<ip_address>'}}"

For example:
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST http://localhost:8181/restconf/operations/
↪config:delete-arp-entry -d "{input:{"node-id":158969157063648, "ip-address":'10.
↪217.9.9'}}"
```

2.6.4 Using Postman to invoke REST API

Besides using the curl tool to invoke REST API, like the examples aforementioned, one can also use GUI tool like Postman for better data display.

- Install Postman: Install Postman in the Chrome browser
- In the chrome browser bar enter

```
chrome://apps/
```

- Click on Postman.

Example: Get VLAN table using Postman

As the screenshot shown below, one needs to fill in required fields.

```
URL:
http://<controller_ip_address>:8181/restconf/operations/vlan:get-vlan-table

Accept header:
application/json

Content-type:
application/json

Body:
{input:{"node-id":<node_id>}}
for example:
{input:{"node-id":158969157063648}}
```

2.7 Multi-vendor support

So far the supported vendor-specific configurations:

- Add VLAN and set ports
- (More functions are TBD)

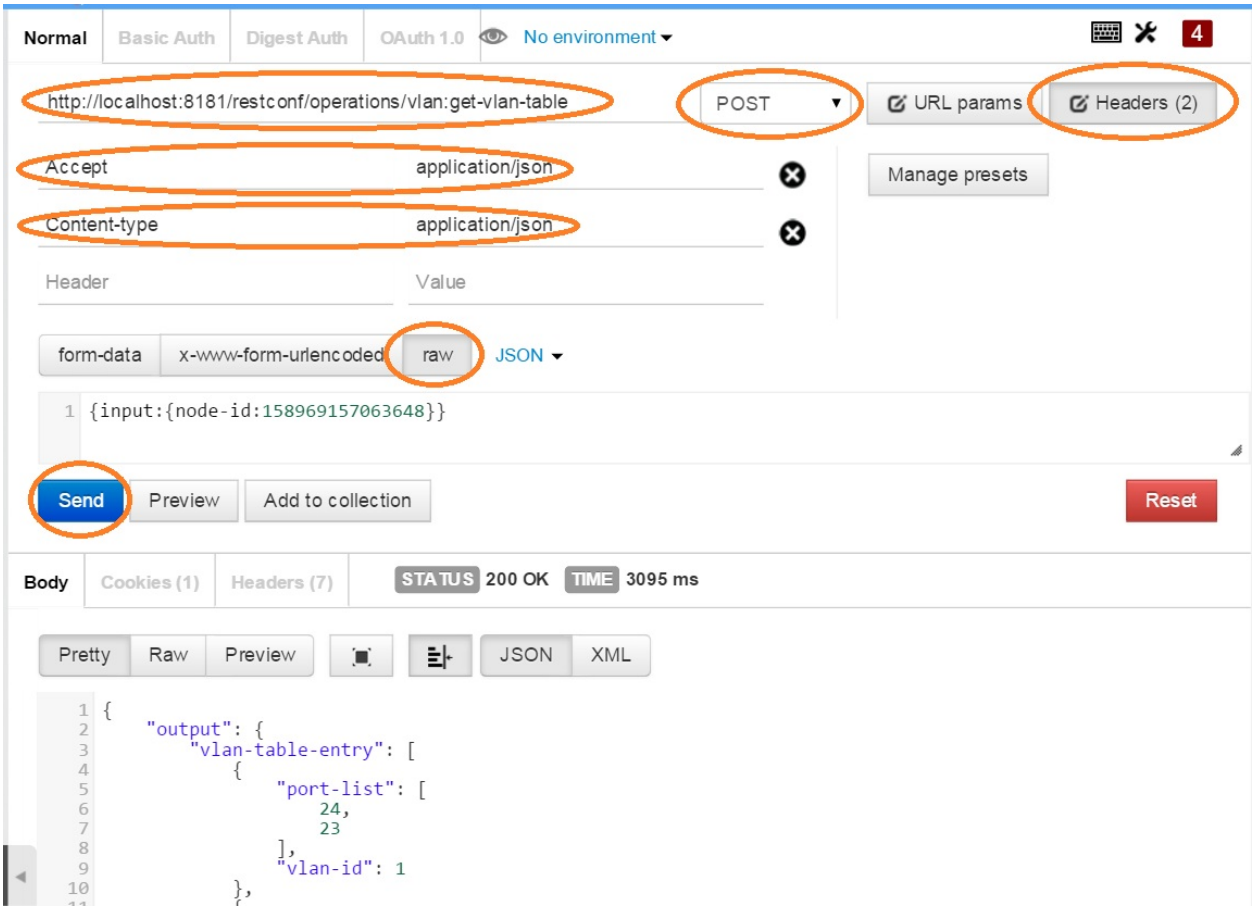

Fig. 2: Example: Get VLAN table using Postman

The SNMP4SDN Plugin would examine whether the configuration is described in the vendor-specific configuration file. If yes, the configuration description would be adopted, otherwise just use the default configuration. For example, adding VLAN and setting the ports is supported via SNMP standard MIB. However we found some special cases, for example, certain Accton switch requires to add VLAN first and then allows to set the ports. So one may describe this in the vendor-specific configuration file.

A vendor-specific configuration file sample is here, and we suggest to save it as */etc/snmp4sdn_VendorSpecificSwitchConfig.xml* so that SNMP4SDN Plugin can automatically load it.

2.8 Help

- SNMP4SDN Wiki
- SNMP4SDN Mailing Lists: (user, developer)
- Latest troubleshooting in Wiki