Qualitative Comparison of Open-Source SDN Controllers

Andrei Bondkovskii [*] [#], John Keeney [#], Sven van der Meer [#], Stefan Weber [*]
[*] School of Computer Science and Statistics, Trinity College Dublin, Dublin, Ireland
bondkowski@gmail.com Stefan.Weber@scss.tcd.ie
[#] Network Management Lab, Ericsson, Athlone, Ireland.
John.Keeney@ericsson.com Sven.van.der.Meer@ericsson.com

***Abstract*—**Telecommunication companies with expensive networks may become the biggest beneficiaries of SDN; however, in contrast to traditional routers, the development of SDN controllers is driven by open-source projects with involvement of the industry. Two prevalent projects in SDN development are the OpenDaylight and the ONOS controllers. These SDN controllers are advanced in their development - having gone through a number of releases - and have been described as being useful for a large number of use-cases. In this work, we compare and evaluate these controllers, in particular their northbound interfaces, by configuring them for a representative use-case, port-mirroring.

I. INTRODUCTION

The adoption of software-defined networking (SDN) promises significant benefits and cost reductions to operators of data networks. The ultimate goal of SDN is to provide networks as abstractions to applications that use the network and remove the need to physically configure network connections. In this work we examine the northbound interfaces for two of the major open-source SDN controllers, particularly because these interfaces are not standardized parts of SDN controllers and are currently poorly described. Northbound interfaces (fig 1) are mechanisms that network owners use to create abstractions of network services for application developers. We focus on the usability and maturity of these northbound interfaces motivated by a representative telecommunications network management use-case: duplicating control-plane and user-plane traffic using traffic-mirroring to direct the traffic to a network probe performing Deep-Packet-Inspection (DPI).

II. BACKGROUND

While SDN is gaining industry traction the concept of network programmability is not new. One of the first significant attempts dates from the mid 1990s with ATM [1], when IP was not considered as the protocol of choice for future networks. Open Signaling [2] and DCAN [3] promoted the idea of separating the control plane from the data plane in networking devices, while Active Networking [4] proposed that routers should recognize the application using the network and treat their traffic in an application-appropriate manner. In the mid 2000s, the 4D Project [5], predecessor of NOX [6] the first SDN controller, proposed delegating control of network devices to a single device. At the same time the NETCONF [7] protocol was standardized as a candidate replacement for SNMP, and later became one of driving forces for SDN.

The simplest definition of SDN describes a network architecture where a separate controller manages networking devices. Alternatively, SDN is a framework where networks are treated as abstractions and are controlled programmatically, with minimal direct manipulation of individual network components [8]. Different points of view reflect the diversity of expectations of SDN however the centralized control of the network remains the primary feature of SDN. The OpenFlow protocol [8], first presented in 2008, is a communications protocol to remotely manipulate the forwarding function of routers and switches, thus supporting a separation of network controllers from network elements. OpenFlow is considered by some as a synonym for SDN and is now a key constituent of most industrial SDN initiatives by acting as a southbound protocol used by SDN controllers (fig 1).

The CORONET [11] developers proposed to use VLAN mechanisms installed into switches to simplify packet forwarding and minimize the size of flow tables. Microflow [12] goes even further and proposes to deal with small flows in the data plane without a controller, which means that forwarding devices would share the control plane with a controller. This approach aimed at data centers but violates one of the main principles of SDN - control and data plane separation. Another principle of SDN - centralized control - is also sometimes sacrificed to achieve scalability. Kandoo [13] propose to divide control tasks between a "root" controller for task requiring network-wide state and "leaf" controllers for applications that that require only local information. Distributed controllers are also supported by the Open Network Foundation (ONF)

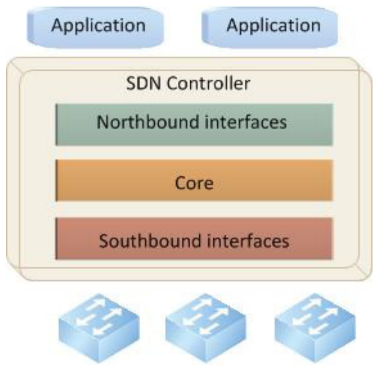

Fig 1: Common SDN Controller Architecture

An SDN controller or network operating system (NOS) is a software based component that implements the control functions of the network and is usually represented as a collection of applications providing different functions of the controller [15]. The architecture of a controller can be broadly broken into 3 parts as shown in fig 1: the northbound interface, the core and the southbound interface. Northbound interfaces are a collection of API's that supports higher level applications with controller services such as synchronized topology views of the network, connectivity oriented services, QoS etc. Southbound interfaces provides control functions to the underlying infrastructure level devices as switches and routers, usually using a control protocol like OpenFlow, NetConf, OVSDB etc. Southbound interfaces collect information from network devices such as status, notifications, alarms, etc., while updating the networks devices with forwarding rules.

Most of the networking industry leaders have now announced their own SDN controllers (e.g.: Cisco ACI, Juniper Contrail, Big Switch's BNC, Brocade Vyatta Controller, etc). Nonetheless SDN research and development remains concentrated in open-source projects, including: the OpenDaylight [9] controller hosted by the Linux foundation; the Open Network Operating System (ONOS) [10] initiated by Stanford and Berkeley Universities; the Floodlight project currently sponsored by Big Switch Networks; and OpenContrail driven by Juniper and others.

III. SDN Controllers – Northbound Interfaces

To compare the functionality of the northbound interfaces of different SDN controllers we chose the two most popular open-source SDN controllers: OpenDaylight and ONOS. These two controllers have multiple northbound interfaces. For OpenDaylight we chose the Yang interface (a basic controller interface) and the Group Based Policy interface (the primary interface for controller developers). For ONOS we examine the Intent Framework which has a graphical user interface (GUI), a command-line interface (CLI), and a REST API. Using these controller interfaces we attempt a simple use case (traffic mirroring), and analyze their suitability for this task.

The chosen SDN controllers primarily use OpenFlow as the underlying southbound interface. However, unlike some fundamental network functions (e.g. topology discovery, forwarding, filtering) traffic mirroring unusually is not natively implemented in OpenFlow but can be implemented using “group tables” (introduced in OpenFlow 1.3). Group tables are used to implement network functions that need to be aware of group of device ports instead of single port. For this reason traffic mirroring is a non-trivial task for SDN controllers, and is useful for comparing SDN controllers.

A. Open Network Operation System - ONOS

ONOS [10] is an open-source SDN controller driven by ON.Lab (OpenNetworks Laboratory) – a non-profit organization founded in 2012 with the assistance of Stanford University and University of Berkeley to develop the ONOS SDN controller. ONOS is positioned as an SDN controller for telecommunications companies and service providers, with a focus on scalability, high availability and performance. The controller is implemented as a collection of OSGi Java applications (running as Apache Karaf components) that interact through Java APIs and REST APIs. ONOS provides a wide range of instruments for developing new controller-based applications, including templates and simple integration into command line or graphical user interfaces. ONOS provides two main northbound interfaces: the global network topology view and the intent framework. Applications can use the topology API to calculate paths, provision flows and perform other network functions. Since an the ONOS controller can be distributed the primary concern for the topology view is to maintain the consistency of the view. Each distributed controller instance maintains a global view and that is synchronized with another randomly chosen controller instance. ONOS Intents are an abstraction to represent the desire to connect two or more points, but does not specify a particular path between those points. When an intent is installed the controller calculates a best path and configures the underlying network devices to maintain the necessary routes (flows) to achieve the intent. If the network topology changes the controller must dynamically update the underlying flows.

To administer networks ONOS provides a REST API, a Command-line interface (CLI) and a web-based graphical GUI interface. The REST API is primary used by developers but can also be used for day-to-day network administration (with some difficulty). The CLI, provided as an extended version of the Apache Karaf CLI, provides access to the main network functions such as maintaining flows on switches and the provision network elements such as switches and hosts. The CLI also provides an interface to the intent framework in a manner that is suitable for network administrators. The primary function of the GUI is to provide graphical representations of the network topology and support easily installation (but not modification or deletion) of intents.

B. OpenDaylight - ODL

OpenDaylight (ODL)[9] is an open-source SDN controller project maintained as part of the Linux foundation. The ODL project was started in 2013 and is widely supported by industry members and researchers, with the goal to make SDN more transparent and to act as basis for Network Function Virtualization (NFV). The controller is written in Java with the main development decisions voted on by an elected Technical Steering Committee. Similar to ONOS ODL uses Apache Karaf OSGI components. One of the main non-commercial use cases for ODL is providing network services for the OpenStack cloud platform.

The ODL controller provides two northbound interfaces for applications: The OSGi interface for applications that are in the same address space as the controller and a web-based REST interface. Unlike ONOS, ODL does not provide any user-friendly interface that would allow configuration or data collection by untrained users. Most of interfaces of the controller are represented in web-based interface DLUX, which is a collection of interfaces that provide access to the controller's functions as visualized REST and YANG interfaces. The ODL topology module provides a simple schema or map of interconnections of devices (usually switches) connected to the controller and hosts known to the switches. The ODL YANG UI module consist of a collection of all accessible REST API's in the controller along with information about the data structures used to communicate with the devices. Using this non-trivial YANG UI module a user can configure a network or query detailed information.

ODL also supports Group Based Policies, a concept inherited from Cisco. Group based policies are similar to ONOS intents but are more flexible and comprehensive. Group based policies cannot be manipulated by the CLI interface but can be accessed by the REST API and has a well-defined DLUX graphical interface. Group based policies require advanced skills and deep understanding of the ODL network model, however they provide flexibility to define intents or connections at the transport layer as well as at the network and data-link layers.

IV. Use Case: Port/Traffic Mirroring to support Traffic Analysis in Mobile Core Networks

A key requirement for any Operating Support System (OSS) monitoring and managing a mobile telecommunications network is to understand the quality of the data connections experienced by network subscribers. Each of the nodes in a managed network provide a plethora of monitoring data that must be gathered, correlated and aggregated by the OSS system. However, examining the control-plane traffic and user-plane traffic itself using network probing is another source of useful information about traffic characteristics and network behavior.

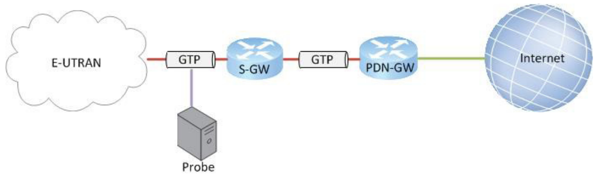

Fig 2: Traffic probing in a mobile telecoms network

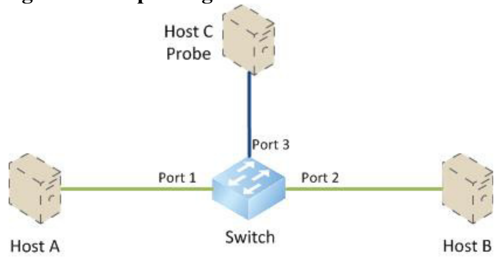

Fig 3: Port Mirroring for DPI network probing

In an LTE mobile telecommunications network (fig 2) user plane traffic flows to/from the Radio Access Network (EUTRAN) from/to the internet where all traffic is encapsulated in GTP (GPRS Tunneling Protocol) tunnels, and traffic must be extracted and reassembled before it can be examined. Therefore to perform traffic analysis requires massive-volume deep-packet-inspection (DPI) network probes to analyze user-plane and control-plane traffic. To avoid jeopardizing network throughput and performance this analysis should not be performed in-line but should instead operate on a copy of the user-plane and control-plane traffic. Unlike existing approaches requiring expensive hardware network taps, a simpler way to achieve this is to mirror the traffic being carried over a number of core-network connections using port-mirroring at a small number of strategic network switches/nodes (fig 3), for example at Serving-Gateways (S-GW) or PDN-Gateways (PDN-GW) (fig 2). As these network switches and nodes become virtualized, and networks between them become virtualized and much more dynamic, it is necessary to consider SDN techniques to achieve this traffic mirroring. In the following sections we examine the feasibility of executing such port-mirroring using the current off-the-shelf versions of the ONOS and ODL SDN controllers.

A. *Port mirroring using the ONOS SDN controller*

As mentioned, there are two main mechanisms employed by the ONOS SDN controller, the ONOS Intent framework, and the ONOS Flow framework. The ONOS Intent framework is an application that allows users to express forwarding policies in forms of intents, using abstract representations of switch ports as anchors. There are two interfaces provided: a CLI combined with Karaf's command line interface and limited functionality provided into ONOS's graphic user interface. For the purpose of this work we use CLI to install intents. The ONOS Flow framework support direct REST manipulation of individual switches. Both methods can provide port mirroring for this use case, but the main difference is that the Intent approach will itself evaluate how to configure the appropriate switches, whereas the Flow approach requires explicit instructions to install flows on particular switches.

1) *ONOS Intent Faramework – CLI*

To implement port mirroring on a network switch (e.g. switch *of:0000000000000001*) using the ONOS Intent framework one must simply create two intents that will match all incoming traffic on the appropriate incoming ports (e.g. all traffic from/to port 1 to/from port 2) and broadcast all traffic to another port (e.g. port 3). This can be achieved using the following Karaf CLI commands:

```
> add-single-to-multi-intent of:0000000000000001/1
          of:0000000000000001/2 of:0000000000000001/3
> add-single-to-multi-intent of:0000000000000001/2
          of:0000000000000001/1 of:0000000000000001/3
```

These commands create the two intents, which are automatically translated to required OpenFlow commands to be installed in the switch. To verify configuration one can execute the *intents* CLI command or using the graphical user interface.

2) *ONOS Intent Framework – REST*

To implement similar port-mirroring intents using the ONOS REST API a pair of JSON REST request similar to the following is required:

```
{  "type": "SinglePointToMultiPointIntent",
   "appId": "DefaultApplicationId{id=35, name=org.onosproject.cli}",
   "details": "SinglePointToMultiPointIntent{
       appId=DefaultApplicationId {id=35, name=org.onosproject.cli},
       priority=100,
       ingress=ConnectPoint{ elementId=of:0000000000000001,
              portNumber=1},
       egress=[ConnectPoint{elementId=of:0000000000000001,
                   portNumber=2},
                ConnectPoint{elementId=of:0000000000000001,
                   portNumber=3}],
       ...   " }    ...
}
```

This request is sent to the REST endpoint: *http://w.x.y.z:8181/onos/v1/intents* where the IP address of the controller's host is represented as *w.x.y.z* and /onos/v1 is common address for all REST interfaces for ONOS. There is no REST mechanism to change an intent, instead it must be removed and re-added. Installed intents can be inspected using a similar REST request.

3) ONOS Flow framework – REST

To install the necessary flows to perform port mirroring on a particular switch a pair of REST GET requests similar to the following can be sent to *http://w.x.y.z:8181/onos/v1/flows* :

```
{ "priority":5, "isPermanent":true,
  "deviceId":"of:0000000000000001",
  "treatment": {"instructions": [
          {"type":"OUTPUT","port":2},{"type":"OUTPUT","port":3}]},
  "selector": {"criteria": [
          {"type": "IN_PORT","port": 1}]}
} …
```

B. Port Mirroring using the OpenDaylight SDN Controller

As mentioned, OpenDaylight provides two mechanisms to achieve port mirroring for our use case: using the REST/Yang interface, or using the Group-based Policy interface. However, it is important to note, in the current version of OpenDaylight some OSGi bundles cannot be installed at the same time, for example the Group Based Policy bundle is not compatible with OpenDaylight OpenFlow bundle.

1) OpenDaylight REST / Yang interface

OpenDaylight interfaces are also expressed using Yang data structure definition, and a graphical Yang user interface is provided based on the ODL DLUX graphical user interface. Using the Yang UI it is possible to directly manipulate the data structures provided by the *opendaylight-inventory* (ODLI), which is in turn translated to REST commands, which are further translated by the ODL controller and sent via one of its southbound interfaces (OpenFlow, NETCONF, OVSDB, etc.) to the underlying network elements. The graphical Yang interface for the controller is available at the address *http://w.x.y.z:8181/index.html/yangui/index.*

To implement port mirroring using the Yang interface requires the creation of two flows where each flow defines a "match" for all input from a port and sends it to two output ports using a pair of "output-actions". The action is available at *config/opendaylight-inventory:nodes/node/XXX/flow-node-inventory:table/0* where *XXX* refers to the node name, (openflow:1 in the following example), and where this flow is defined for table 0. The Yang interface then creates a REST request to create both flows similar to the request below:

```
{ "table": [{ "id": "0",
      "flow": [ { "id": "probeflow1",
           "match": {"in-port": "openflow:1:1"},
           "instructions": { "instruction": [{ "order": "0",
                  "apply-actions": {
                    "action": [{ "order": 1,
                        "output-action": { "max-length": 65535,
                          "output-node-connector": "3"
                        }},{"order": "0"
                        "output-action": {"max-length": 65535,
                          "output-node-connector": "2"
                        },
                      }]}}]
          }, …
          "table_id": "0"},
      "flow": [ { "id": "probeflow2", … } …   ] }]
}
```

2) OpenDaylight Group-based Policy Interface

OpenDaylight Group Based Policies (GBP) allows users to express forwarding rules in form of policies between endpoint groups, thus expressing network configurations in a declarative versus imperative way [9] matching application connectivity requirements to the underlying details of the network infrastructure. Group-based policies can be manipulated using either the REST API or using the DLUX graphical interface. To define a GBP policy it is necessary to define endpoints, and grouping rules/selectors for endpoints to form endpoint groups. Endpoint groups are then referenced (selected) in contracts, either as providers of some capability or as consumers of some required capability. In the contract itself subject rules define how two endpoints are allowed to communicate, where these rules match against traffic and perform any necessary actions on that traffic. Currently there are only two types of action: 1) allow and 2) redirect to a separate service function chain. The allow action does not provide support for flooding or copying of traffic, thus cannot be used for purpose of port mirroring. Service function chains do not yet provide a port mirroring service either, but such service will likely appear in the near future. Additionally the rule classifiers are also limited (classifiers are the conditions in the rules that define which traffic is applicable for the rules). Currently classifiers support three options: 1) Ethernet type, 2) IP protocol, 3) L4 protocol, so there is no support for switch port selection, however a more flexible traffic type classifier would likely suffice to select all traffic required for our GTP traffic selection use case. Therefore, while port -mirroring is not currently supported with OpenDaylight Group-based Policies, the approach is flexible and useful for other use cases.

V. COMPARISON

To verify the correct operation of each of the mechanisms described above, each was implemented and tested (except for the ODL Group-based Policy interface, which at time of writing does not support a port mirroring function, however a hypothetical implementation can still be imagined). Each implementation was tested in an identical test environment and manually verified using a packet analyzer to ensure all traffic from/to switch port 1 to/from switch port 2 was mirrored to switch port 3 and correctly redirected to the probe appliance.

In this section we evaluate different aspects of the four candidate approaches described above: ONOS's Intent Framework, ONOS's REST API, ODL's Yang interface and ODL's Group-based Policy interface. For the evaluation we defined four basic functions required to support port mirroring as a service. 1) *Discovery*: support to examine the network topology, identify the particular device and traffic ports (ingress, egress, mirror), and to examine the current configuration of device(s) involved. 2) *Setup*: ease and clarity to define and install the port-mirroring function. 3) *Change*: the extent to which is possible to change of port-mirroring function on the fly, e.g. add new switch ports without stopping the service. 4) *Removal*: the ease of removal of the function.

A. *Discovery*

Discovery consists of two main functions: topology view and presentation of the current configuration. In each of the systems their topology view component is implemented separately from the controller component itself, however the topology view often includes aspects of the controller interface. For example the ONOS Intent Framework GUI is integrated into the topology view, so it is possible to identify or change intent configurations directly with the topology view. The ONOS Intent Framework's CLI is also coupled with the topology CLI. The ONOS REST API also provides a topology view, although it is not as expressive, since results are represented only as JSON strings. For topology discovery in ONOS it is recommended to use web-based ONOS GUI rather than the REST API.

ODL's Yang and Group-based Policy interfaces are integrated into the graphical DLUX interface with a graphical topology representation; however the topology viewer is not as informative as in ONOS. It is also possible to query topology using the Yang interface, but again results are presented only as JSON strings.

Configuration representation in the ONOS Intent Framework is simple with both interfaces, GUI and CLI. The difference is that in the GUI the path that packets are forwarded through can be visualized in the current topology, i.e. by selecting a device all related intents' connections are highlighted, however with this approach it is difficult to identify the effect of individual intents when multiple intents are installed. The ONOS CLI presents a representation of intents so users can easily understand an intent's functionality and also define ingress and egress ports. The ONOS REST API provides very similar representation of intents as JSON objects, which also contain some control information, but remain readable.

The ODL Yang interface provides a good representation of the configuration supported with Yang data structures. Instead of raw JSON strings or XML, the user can observe the information already parsed and presented graphically as the populated tables, pull down menus, radio buttons, comments, etc., as defined in the Yang interface. This automatic parsing significantly improves the quality of representation. The major drawback is a lack of documentation for the Yang interface, for example it is difficult to define the URL to send the GET REST request. Troubleshooting is also difficult as error messages are often inconsistent. The ODL Group-based Policy interface has a graphical representation of the current configuration, which shows all configured policies in a topology view. This representation is natively understandable once the user is familiar with the abstractions provided by interface, e.g. endpoints, subjects, contracts, etc..

B. *Setup*

The ONOS Intent Framework has two mechanisms to install intents. The first is to select two host icons in GUI and then select *setup intent* from the mouse context menu. This is suitable only for host-to-host or switch-to-switch intents and so is not applicable for port mirroring. The second mechanism uses the CLI, which is aided by context-sensitive text-completion for the options and parameters being configured.

There is a significant lack of documentation for the ONOS REST API, requiring manual specification of the JSON string to be sent. Some strategies to assist in forming the JSON string include: querying for similar intents and then changing the returned string, trial and error by examining error messages indicating incorrect requests, directly examine the source code for hints about data-structures, etc.. Although the ONOS REST API is the most expressive and reliable mechanism to install intents, the lack of documentation is a significant inhibitor.

Similar to the ONOS REST API, the ODL Yang interface uses REST to perform flow configuration operations, however, the Yang interface visualizes data structures used to generate the REST JSON string with inputs specified using named text areas, radio buttons, pull down menus, etc.. The Yang interface is poorly documented, and most available tutorials and resources quickly become out of date as the interface evolves. For example, there is no indication of which inputs are compulsory or optional. On other hand, once familiar with the interface users can easily use it for a wide range of configurations.

As described in the previous section the ODL Group-based Policy mechanism is not currently suitable for port mirroring, however other operations are easy to perform in the fully visualized interface. Although the Group-based Policy model is complicated, once it is understood then it becomes easy to use when creating new rules. As the other interfaces, this interface is new and still under active development, and again similar to other interfaces, this suffers from a significant lack of documentation.

C. *Change*

The ONOS Intent Framework GUI does not support changing existing intents. Instead it is necessary to delete existing intents and create new ones. For our use case this is not a major impediment. To configure port mirroring, we used the ONOS REST API to use both intents and flows. As with the Intent Framework interface, changing intents is not possible. Using the ONOS REST API it is possible to change flows on the switch by sending new configurations with the same table-/flow-ID. As with the difficulty in setting up flows or intents data structures are poorly documented.

With the ODL Yang interface, one creates flows to install them to switches. These flows can also be easily changed by sending new configuration with same table-/flow-ID. As with initial flow creation, this operation becomes easier as the user becomes familiar with the interface and data structures. Although ODL Group-based policies cannot currently be used for port mirroring, we expect that once available, such port-mirroring configuration changes will be as easy to use as other configuration changes supported by this interface.

D. *Removal*

With the graphical ONOS Intent Framework interface it is not possible to remove intents, but it is easy to locate and delete intents using the CLI interface. To remove intent using the ONOS CLI requires the *remove-intent* command along with

Table 1 Summary of Functionality Comparison of Northbound Interfaces

	ONOS Intent Framework	*ONOS REST API*	*ODL Yang Interface*	*ODL GBP Interface*
Discovery	Easy but incomplete with GUI. CLI is more complete but more complicated	Moderate	Moderate, not documented	Simple, fully visualized
Setup	Straightforward with CLI, limited with GUI	Difficult, data structures not documented	Difficult, not documented	Not implemented
Change	Not implemented	Not implemented with intents, difficult with flows	Difficult / easy after previous step, not documented	Not implemented
Removal	Easy with CLI, not implemented in GUI	Moderate, data structures not documented	Moderate/easy after previous step, not documented	Not implemented

the intent ID. Once an intent is removed it is not deleted, but rather its status is changed to *withdrawn* and remains in intent database. Removal of intents using the ONOS REST API is easier then creation (perform a *GET* followed by a *DELETE* for the required intent), but still far from obvious due to the lack of documentation. Removal of flows using the ONOS REST API is largely similar to the process of removing intents, so is easy once the installation process has been understood.

Using the ODL Yang interface to remove a flow is symmetrical to discovery, except uses the *DELETE* operation after using the *GET* method to find the appropriate flow. Similar to discovery, this interface is difficult to use without experience. The ODL Group-based Policy interface provides easy to use methods to delete any part of a configuration depending on the operation required. For example, to delete a forwarding rule, a specific contract can be excluded from the contract list for an endpoint or endpoint group.

VI. Conclusions

Numerous approaches to manipulate and managed network as Software-Defined Networks (SDNs) have been previously presented, but recently a number of tools have presented themselves as ready for use. However, these tools, which are currently in very active development, largely remain poorly documented and their standardization remains uncertain, especially their northbound interfaces. In this work we have assessed the assorted northbound interfaces of two of the most popular SDN controllers (ONOS and OpenDaylight), which showed significant diversity in these northbound interfaces as summarized in Table 1. We based this evaluation on our experiences implementing traffic-mirroring – a basic mandatory networking function required to support real-time deep packet inspection in telecommunications network. We have defined and used simple evaluation criteria, based on how easy it was to: 1) examine the network (*discovery*), 2) add a new network function (*setup*), 3) change an existing function (*change*), and 4) remove a function (*removal*). While our evaluation was restricted to just two SDN controllers (ONOS and ODL) and a single network function (port-/traffic-mirroring), we feel that these criteria will be useful to evaluate other SDN controllers and other supported network functions. It is clear from our evaluation that huge advances have been made, however, as expected there remains gaps in the northbound interfaces of these systems, particularly with respect to ease of use, documentation and completeness of functionality.

References

[1] Juha Heinanen. Rfc1483: "Multiprotocol encapsulation over atm adaptation layer 5". https://tools.ietf.org/html/rfc1483. 1993.

[2] Campbell, A.T., Katzela, I., Miki, K., Vicente, J. "Open signaling for atm, internet and mobile networks" ACM SIGCOMM Comput. Commun. Rev., 29(1), 1999

[3] University of Cambridge. "Dcan, devolved control of atm networks". http://www.cl.cam.ac.uk/research/srg/netos/old-projects/dcan [Online; accessed 2015]

[4] Tennenhouse, D.L., Wetherall, D.J. "Towards an active network architecture". DARPA Active NEtworks Conference and Exposition, 2002.

[5] Greenberg, A., Hjalmtysson, G., Maltz, D.A., Myers, A., Rexford, J., Xie, G., Yan, H., Zhan, J., Zhang, H. "A clean slate 4d approach to network control and management. ACM SIGCOMM Comput. Commun. Rev., 35(5), 2005.

[6] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., Shenker, S., "Nox: towards an operating system for networks". ACM SIGCOMM Comput. Commun. Rev., 38(3), 2008.

[7] Enns, R. "Rfc4741: Netconf configuration protocol" http://www.ietf.org/rfc/rfc4741. 2006.

[8] Marschke, D., Doyle, J., Moyer, P. "Software-Defined Networking: Anatomy of Openflow". Lulu publishing, 2015.

[9] Linux Foundation. "The OpenDaylight Platform" https://www.opendaylight.org [Online; accessed 2015]

[10] ON.Labs "ONOS Open Network Operating System", http://onosproject.org [Online; accessed 2015]

[11] Kim, H., Santos, R.R., Turner, Y., Schlansker, M., Tourrilhes, J., Feamster, N. "Coronet: Fault tolerance for software defined networks". IEEE Intl. Conf. on Network Protocols (ICNP 2012), 2012.

[12] Narayanan, R., Kotha, S., Lin, G., Khan, A., Rizvi, S., Javed, W., Khan, H., Khayam, S.S. "Macro flowows and micro flowows: Enabling rapid network innovation through a split sdn data plane", Eur. Wksp on Software Defined Networking (EWSDN 2012), 2012.

[13] Yeganeh, S.H., Ganjali, Y. "Kandoo: a framework for efficient and scalable offoading of control applications", Wksp. on Hot topics in software defined networks (HotSDN 2012), 2012.

[14] Tanenbaum, A.S., Wetherall, D.J. "Computer Networks" 5th ed. Pearson publishing, 2011.

[15] Nunes, B., Mendonca, M., Nguyen, X.-M., Obraczka, K., Turletti, T., et al. "A survey of software-defined networking: Past, present, and future of programmable networks". IEEE Commun. Surveys Tuts., 16(3), 2014.