Qualitative Analysis of Open-Source SDN Controllers

by

Andrei Bondkovskii, Mr.

Dissertation

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

Master of Science in Computer Science

University of Dublin, Trinity College

August 2015

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Andrei Bondkovskii

August 27, 2015

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Andrei Bondkovskii

August 27, 2015

Acknowledgments

This thesis would not have been possible without the support of many people. Firstly I would like to thank my supervisor, Stefan Weber, who patiently spent a lot of time guiding me. I also wish to extend my thanks to my course director, Stephen Barrett, who gave me huge moral support. Many thanks to my mentors from Ericsson LMI John Keeney and Sven van der Meer who inspired me to write this thesis and was helping day by day. And finally, special thanks to my wonderful wife who endured this long way with me, always offering support and love.

Andrei Bondkovskii

University of Dublin, Trinity College

August 2015

Qualitative Analysis of Open-Source SDN Controllers

Andrei Bondkovskii, M.Sc.
University of Dublin, Trinity College, 2015

Supervisor: Stefan Weber

The adaptation of software-defined networks (SDN) technologies promises significant benefits and cost reductions to operators of data networks. 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 controller. 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 our evaluation environment by configuring them for a representative use-case, port mirroring.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter we will shortly describe Software-Defined Networking and changes that it brings to the networks. We will explain the choice of controllers and their interfaces analysed in this work. Finally we will provide overview of work methods and structure of dissertation.

1.1 Towards Software-Defined Networking

Software-Defined Networking is catching more attention day by day. Academics that involved in SDN research and development declare lots of benefits and solutions for accumulated problems of data networks. The adaptation of SDN promises to significantly reduce CAPEX and OPEX for network owners, make networks more flexible and reliable. The ultimate goal of SDN is to provide network as abstraction to applications that use the network and finally get rid of need to manually configure network connections.

The idea of SDN is no way a new. History knows several failures to achieve network programmability. First serious attempts were done in mid 1990s when ATM [13], but not IP was considered as the protocol of future networks. Open Signalling [2] and DCAN [21] were promoting idea of separating control plane of data plane in networking devices, while Active Networking [28, 29] proposed to let router to recognize the application that wants

to use the network and treat it in corresponding way. Next pick of interest to concept of programmable networks was in middle 2000s, when 4D Project [10, 26] predecessor of first SDN controller - NOX [11], proposed to delegate control plane of networking devices to a single device. At the same time NETCONF [9] protocol was standardized that first was considered as replacement for SNMP, but then become one of driving forces for SDN.

There are several different definitions of SDN. The simplest one describes SDN as a network architecture where controller manages networking devices as shown in figure 1.1. From another point of view SDN is a framework where networks treated as abstractions and are controlled programmatically, with minimal direct "touch" of individual network components [8].

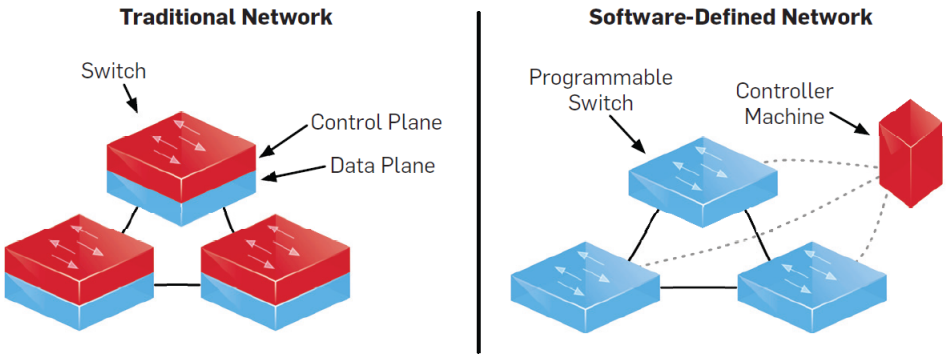

Figure 1.1: Centralized control layer [3]

Such different points of view reflect diversity of expectations of SDN as a new technology. Nevertheless the implementation of centralized control of the network became the

number one task related to SDN. One of the key roles here belongs to OpenFlow protocol, which was first presented in 2008 by consortium that united Stanford, Berkeley, MIT, Princeton and some other universities. OpenFlow currently treated if not as synonym of SDN, then at least as a compulsory part of it. OpenFlow main function is to provide connection and data structures for controllers to manage network devices.

1.2 Controllers And Interfaces

The gravity of SDN concept is also increased by the fact that all of the industry leaders announced or released own controllers: Cisco ACI, Juniper Contrail, Big Switch's BNC, Brocade Vyatta Controller, etc. Nonetheless the driving force of SDN research and development is concentrated in open source projects that mushroomed recently. Some of them are OpenDaylight controller hosted by Linux foundation, Open Network Operating System initiated by Stanford and Berkeley, Floodlight project that currently sponsored by Big Switch Networks, OpenContrail driven by Juniper and others. Several of projects obtained significant support by industry and accumulated massive developer communities. All of these controllers shares same general architecture as shown in figure 1.2.

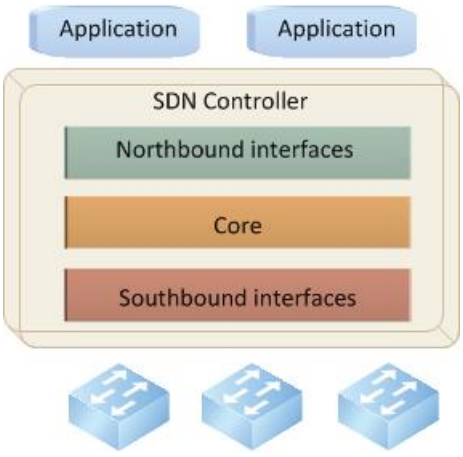

Figure 1.2: Common SDN controller

In this work we will take a close look on northbound interfaces since it is the less

described and not standardized part of SDN controllers. Northbound interface is the way for applications to interact with controller, the place where controllers developers has a chance to create abstractions of network services for application developers. Usability and maturity of northbound interfaces are major criteria for network owners in selecting of the controller. To study these qualities and compare them between controllers and interfaces we took a specific use case where we will test them. This use case is describing probing in telecommunication networks. To stay platform independent and not affect throughput, probing device has to use mirrored traffic instead of being in-line. Thence we chose traffic mirroring as a function to be configured to compare qualities of northbound interfaces.

To keep the work concise we decided to pick two of controllers. First of them stand out of the crowd and became recognized leader, it is OpenDaylight that accumulated the biggest partners network and developer community. Second - ONOS became was chosen as only controller that positioning itself as specialized for telecommunication networks. It is also proved by the fact that the biggest telecommunication vendor - Ericsson supports this controller and use its architecture as base for its own controller - Ericsson Virtual Router. These two controllers has several northbound interfaces each. Again to keep work concise we decided to choose two interfaces for each of them. For OpenDaylight we chose Yang interface that is basic controller interface and Group Based Policy interface that is priority one interface for controller developers. For ONOS we took Intent Framework which has user interfaces integrated in GUI and CLI of controller and REST API that allow to application developers to interact with controller in well known standard way. In this work we will try to configure port mirroring using these controllers and interfaces, whereupon will analyse their suitability for this task.

1.3 Overview

Chapter 2 provides overview of legacy switching process and legacy network functions provided by SDN, detailed look at OpenFlow protocol, its evolution and suitability for

port mirroring configuration. It also explains in detail our use case and introduces OpenDaylight and ONOS controllers as well as their interfaces.

Chapter 3 describes test network topology and provides design of port mirroring in this network using chosen interfaces: ONOS's Intent Framework and REST API as well as OpeDaylight's Yang interface and Group Based Policy interface.

Chapter 4 describes in details test environment and provides step by step implementation of port mirroring using chosen interfaces in the same order as they presented in previous chapter.

Chapter 5 provides detailed evolution of functionality of chosen interfaces related to port mirroring as well as comparison of these interfaces and short analyse on its suitability.

Chapter 6 discuses relativeness of use case and its applicability to interfaces and lessons that can be learned from work done.

Chapter 2

State Of The Art

In this chapter we will discuss the legacy packet forwarding and main network functions that based on switching. Then we will discuss a concept of SDN and how traditional network functions are provided in new environment. Then we will take detailed look on the OpenFlow protocol as the main southbound protocol in software-defined networking and on the evolution of OpenFlow. Then we will get acquainted with chosen open source SDN controllers - ONOS and OpenDaylight and will study their most developed northbound interfaces. Finally we will describe the use case that lies in the basement of this work.

For the purpose of this work we will define switching as packet forwarding between hosts in a single broadcast domain or point-to-point link and forwarding decision are based on information from protocols from layer 2 of the OSI model e.g. Ethernet's MAC addresses, 802.1Q's VLAN IDs. Routing is packet forwarding between broadcast domains or inside a domain or p-2-p link that bases forwarding decisions on information from a protocol from layer 3 and higher of the OSI model. Thus packet forwarding inside one broadcast domain that uses IP address or, for example, TCP ports will be called routing. For another example, packet forwarding between two hosts in separate broadcast domains for which the MAC address of the destination is used will also be called routing.

2.1 Legacy Switching

Packet switching in Local Area Networks (LANs) is the process of forwarding of frames between hosts in single local area network, where only data-link layer addresses are examined [27]. The main difference between switches and its predecessors, hubs, is that switches are not only propagating packets, but also make a decisions on propagation of each packet. Ethernet switches have two main functions: Media Access Control (MAC) address learning and retransmission of Ethernet frames from one ports to other. One of the main logical components of the switch is MAC address table. This table consist of two columns: MAC address of the destination and switch port which should be used to reach that address. Every time the frame with uknown source MAC-address hits the switch, a new table entity is created with source MAC address of the frame and ingress port of the frame. The entity could be also added manually by switch administrator which is basically very comprehensive method of network administration. In most basic functionality implementation switch makes a decision on a packet by examining destination MAC address in its header and comparing it to information in its MAC address table. If switch has table entity that contains destination MAC address of incoming packet it will process it to the port in such entity. When switch has no such entity, it sends packet through all active ports except the one it received packet from. Because most of connectivities in Ethernet networks are bidirectional, there is a big chance that switch will learn MAC address of destination when it owner send reply and it will allow switch to use unicast delivery for next packets addressed to that destination. Unicast forwarding significantly increases effectiveness of the network comparing to broadcast, but some functions, including MAC learning, are still require broadcasting. A group of hosts that could be reached with particular broadcast delivery forms a broadcast domain. The efficiency of data exchange in broadcast domain is inversely proportional to its size. In modern IP networks broadcast domains rarely unites more than 254 nodes. Routers, layer 3 switches and other network layer devices are used to separate boundaries of broadcast domains and reduce its size.

Virtual Local Area Networks (VLANs) could be used for separation of broadcast domains inside layer 2 switch. Division into different VLANs achieved by addition of information of vlan number to Ethernet frame header and also by separation of MAC tables of the switch. Switch never shares information about MAC address between tables providing by that logical separation of virtual networks. Another important technology that used in switches is port aggregation that could be used for increasing throughput and high availability of connection. To provide aggregated port switch usually creates logical port which use virtual MAC address for frames exchange. That MAC represents each physical port included into aggregated logical port. L3/L4 (layer 3 and 4 of OSI model) switching is a process of forwarding of packets where decisions based on information in headers of L3 and L4 protocols like IP, ICMP, TCP, UDP, SCTP etc. L3 switching routing is a transferring packets between LANs or broadcast domains. L4 switching is basically used not for interconnections but to provide additional services to the network as load balancing, access control, Quality-of-Service (QoS) etc.

2.2 On Functionality In Software-Defined Networking

Started from enterprise networks, SDN then captured all other areas of networking. Many efforts are directed on implementations of SDN in data centres, Wide Area Networks(WAN), wireless networks, telecommunication networks and even home networks. Targeting telecommunication networks, SDN researchers and developers first of all trying to increase scalability and availability of SDN. ONOS developers came with idea of separating controller functions between different software instances which could be located on different physical servers. They detached functions that provide core controller functionality, as forwarding or topology building, from those which provide connectivity with forwarding devices. With third release ONOS controller could be separated to 32

software instances and this number expected to grow in next releases. Many other ways to achieve scalability are proposed. CORONET [16] developers proposed to use VLAN mechanisms installed into switches to simplify packet forwarding and minimize the size of flow tables. Microflow [19] goes even further and propose to deal with small flows in data plane without controller, which means that forwarding devices would share control plane with controller. This approach is pointed on data canters and it also violates one of the main principles of SDN - control and data plane separation. Another principle of SDN - centralized control - is also some times sacrificed in order to achieve scalability. Kandoo [12] propose to divide single controller to "root" controller that is in charge for tasks that demanded network wide state and "leave" controllers that providing functionality for applications that could be run locally. The idea of distributed controller was also supported by Open Network Foundation (ONF), thus OpenFlow which is without any doubt is current most popular SDN southbound protocol received support of multiple controllers in specification 1.5.

Using SDN in data centres developers also targeting to achieve new level of power usage efficiency. Heller et al. in [14] shows that while most power saving efforts concentrated on servers and cooling, data network's efficiency has a big potential for optimisation. With network-wide power manager that able to find network subset which satisfies conditions while consumes minimum of power it is possible to save 25-62% of energy under various traffic conditions. Keeping in mind that network itself consumes from 10% to 20% of data center electricity, SDN-enabled networks can achieve 2,5-12,5% savings.

OpenFlow handling data as flows in similar approach as traditional optical networks do. This fact makes SDN easy adoptable for optical networks while keeping benefits from manageability improvement. Implementing SDN in WAN Google achieved almost 100% utilization of their optical links [15].

2.2.1 Traditional Network Functions Implemented With SDN

Turning back to enterprise networks we will next describe implementation of "traditional" or basic network functions with SDN. By basic network functions we understand functions of forwarding devices that do not need information from levels 5-7 of the forwarding data to make forwarding decisions. Thus we do not include in this list deep packet inspection or intrusion detection system which usually examines the content of the packet. Our goal in this section is not to define all network functions, but to show how some of them are different while been implemented in SDN.

1. Forwarding of packet is the first and most basic network function. Traditionally, as it was mentioned before in this work, we distinguish forwarding itself and routing. Forwarding uses layer 2 information of the packet's header in order to make decision which way to send the packet and usually operates in single broadcast domain, while routing is using layer 3 information and serving to send packets between broadcast domains or networks. In SDN this definitions are not so strict any more as the process of making decision and the process of forwarding itself are decoupled now. Thus we can use information of layer 3 or 4 to switch packets inside one broadcast domain or in opposite we can use MAC addresses to decide to send packet in another network probably changing packet header in advance.

2. Broadcast is another basic function which was implemented naturally with appearance first multi-port devices. Broadcast in SDN environment is also might be used in much more comprehensive and flexible way. As controller has network wide view, it can for example interleaves broadcast with multicast for same data flow on different devices.

3. Topology discovery is fundamental network function that provides basement to all other functions. It is almost impossible to imagine modern network where system administrator set up links and its addresses manually. Network wide topology view

is one of the biggest benefits of SDN, that provides consistent approach to implementation of network functions in different network devices. Usage of common view reduces possibility of errors in traffic forwarding that may lead to loops and congestions.

4. Filtering of packets allows to implement many traditional network functions: layer 3 and 4 firewall, access control, quality-of-service, etc. SDN brings great deal of flexibility in packet filtering with network wide view. Since SDN controller manage the whole network, filtering of packets might be implemented closer to source, say in access layer of network, which eliminates the need of transport them to distribution level - traditional place of such functions implementation.

5. High availability, or fast re-routing of packets to another physical channel in case of link failure, is also improved with adoption of SDN. Having full view of network topology, SDN can provide high-availability naturally at the policy level.

6. Load balancing traditionally implements at 2-4 layers of OSI model and uses wildcard mask to divide addresses (MAC, IP, TCP/UDP port) into pools that proceed in different way. Advanced load balancers can also consider actual meters and provide dynamically changing balancing. As SDN controller aggregates comprehensive meter gathering with ability of setting up flexible rules, it naturally provides base for implementation of complex load balance applications without need of additional hardware.

7. Traffic mirroring is using to support network functions that could not be put in line or undesirable in line because of significant lower throughput. It could be deep packet inspection, intrusion detection system (especially with anomaly detection), probes, etc. Another major reason to use traffic mirroring is for traffic analysis and troubleshooting. traditionally traffic mirroring is corresponds to port mirroring as it is the simplest way to mirror traffic. Another option is mirroring of VLAN traffic.

With SDN and OpenFlow in particular, traffic mirroring adopts flexibility of rules and could be fine tuned. For example SDN switch can mirror traffic based on IP addresses or even TCP port.

In next section we will take a detailed look at OpenFlow protocol and how port mirroring can be done with OpenFlow.

2.3 OpenFlow Protocol

OpenFlow is the first standard communications interface defined between the control and forwarding layers of an SDN architecture [22]. Appearing in 2008 OpenFlow became a de-facto standard southbound interface of SDN controllers. OpenFlow as technology traditionally divided to three main components: switch and SDN controller that supports OpenFlow and protocol that provides information exchange between them.

SDN controller or network operating system (NOS) is sowtware based component that accommodates all control functions of the network that usually represented as a collection of applications providing different functions of controller [20]. There are three main components in architecture of the controller usually allocated: northbound interface, core and southbound interface. Northbound interface is a collection of API's that supports upper level applications with services of the controller such as synchronized topology of the network, connectivity oriented services, QoS etc. Southbound interface provides control functions to the underlying infrastructure level devices as switches and routers through OpenFlow protocol as well as some others as NetConf, OVSDB etc. Southbound interface collects all the information from network devices as status, notifications, alarms. Another fundamental function is updating switches and other devices with forwarding rules. The main functions of core element are to translate high-level policies received through northbound interface into simple forwarding rules that might be inserted into switches to support policies expressed by user and also synchronizing all the network devices statuses received through southbound interface into global network view that

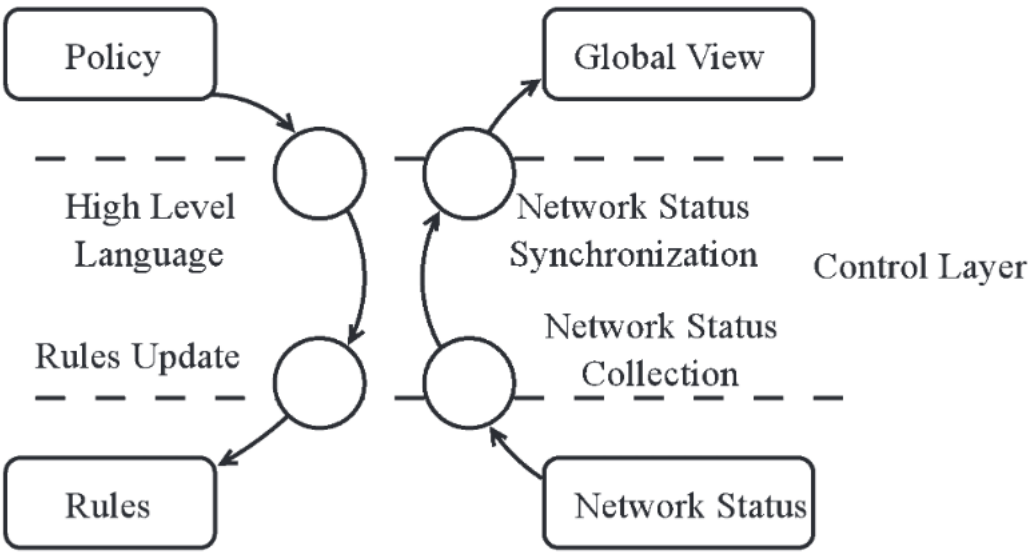

Figure 2.1: Abstract model of SDN controller

might be provided to the users. Other crucial function of the controller's core is checking policies and extracted forwarding rules for conflicts and resolving them keeping network free of configuration errors as loops, undesirable access and others. Wengfeng Xia et al. [30] defines universal logical design of SDN controller shown in figure 2.1 .

High-level language is used to translate user policies to forwarding rules, then forwarding rules could be installed into switches with Rules Update component. Network Status collection component interacts with Network Status Synchronization component in order to obtain global view or network-wide status.

OpenFlow enabled switch is a hardware or software switch that meets requirements of one of versions of OpenFlow switch specification. Most of switches supports specification versions 1.0.x or 1.3.x. Specifications describes logical structures and algorithms of switch as well as structure of all message that switch exchanges with controller. The main components of OpenFlow described in version 1.0 switch specification [5] are flow table and secure channel to the controller. All packets processed by the switch are compared against the flow table. Flow table consist of three components: header field, counters and

actions.

Table 2.1: OpenFlow. Flow table v1.0

Header Fields	Counters	Actions

Header field contains fields that match against packet, counters collects information on tables, ports, flows and queues while actions field consist of actions applied to a packet that matched header field. The traffic between controller and switches is extremely sensitive. Switches must use secure connection (secure interface) to connect to controller and keep traffic secure.

OpenFlow protocol describes format and roles of messages between controller and switches. There are three main types of messages each of which has many sub-types. Main types are controller-to-switch, asynchronous and symmetric. Controller-to-switch messages are sent by controller to manage or query the state of the switch. Sub-types of controller-to-switch type messages are:

1. Features used by controller to query switch capabilities.
2. Configuration used to set or query configuration parameters of the switch.
3. Modify-State used to to set or delete flows in the flow table of the switch and also to setup switch ports properties.
4. Read-State used to collect statictic from switches counters.
5. Send-Packet used to send packets out of stated port of the switch.
6. Barrier used to ensure that message dependencies are met and to receive notifications for complete operations.

Asynchronous messages are sent by switch and used to notify controller about network events or changes in switch status. Sub-types of asynchronous messages are:

1. Packet-in used by switch to send packet (or its fraction if switch buffers packet) that did not matched any flow or matched flow where action is sent to controller.

2. Flow-Removed sends whenever flow removed from the table automatically after inactivity time-out or by controller. For every flow that controller add, modify or delete from the switch it may specify whether switch has to send Flow-Removed message or not.

3. Port-Status used to send port status updates to the controller.

4. Error used to inform controller about any problems on the switch.

Symmetric messages might be initiated by controller or switch. Sub-types of symmetric messages are:

1. Hello used upon connection start-up between switch and controller.

2. Echo request/reply message that used to show latency, bandwidth or liveness of connection between controller and switch. Could be sent by any of them.

3. Vendor used by switch to provide additional functionality in OpenFlow message type space.

Switch specification is one of the main but not only document that specify OpenFlow. There are also OpenFlow management and configuration protocol specification, OpenFlow Notifications Framework, OpenFlow Controller-Switch Negotiable Datapath Model, OpenFlow Table Type Patterns and finally OpenFlow Conformance Test Specification. Most of these documents exist in multiple versions.

2.3.1 Evolution Of OpenFlow

OpenFlow as SDN itself is relatively new but already very popular protocol and it is development process is very agile. Since this process is driven by hundreds of researchers

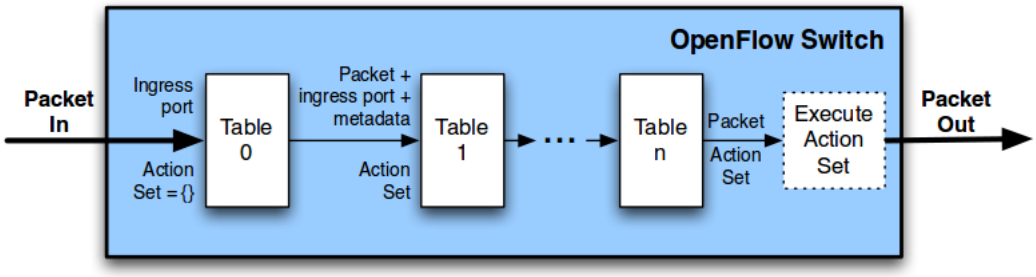

Figure 2.2: Packet Flow in OpenFlow v1.3

as well as all symbolic vendors, the protocol changes significantly every year.

As of this paper was written the last version of OpenFlow switch specification has been 1.5.1 (protocol version 0x06). Nevertheless the most common version is 1.3.x and 1.0.x after it. We will underline differences between versions 1.0 and 1.3 as well as between 1.3 and 1.5 passing versions 1.1, 1.2 and 1.4 as less popular and in result less significant.

The most noticeable changes in version 1.3 [6] comparing to 1.0 are:

1. Multiple flow tables in switch and concept of pipeline processing that shown in figure 2.2. Pipeline processing continues before the instruction set associated with flow entry that packet matches do not specify next flow table, but express a "final" action like forwarding or drop.

2. Group tables is a new abstraction that represent a group of ports as an entity for forwarding packets. The action in flow table may direct packet to the group, which may contain sets of actions used for flooding or advanced forwarding e.g. link aggregation, fast re-routing or multipath forwarding.

Table 2.2: OpenFlow. Group table

Group Identifier	Group Type	Counters	Action Buckets

We will not go deep into specifics of groups and its Action Buckets, note only that this concept add a great deal of flexibility and advanced functionality to OpenFlow switch.

3. Support of MPLS tags that appears first in specification version 1.1, opened new perspectives for OpenFlow protocol usage in provider scale transport networks as well as core networks. Adopting compatibility with MPLS OpenFlow also received a boost to its development from current MPLS users. In [A roadmap for traffic engineering in SDN-OpenFlow networks] Ian F. Akyildiz et al notice that traffic engineering is crucial for Internet providers and OpenFlow with its very reach functionality may provide traffic engineering in absolutely new level.

4. Extensible match field. From version 1.2 match field has no more predefined structure and became flexible instead. All match fields expressed through type-length-value structure called OpenFlow Extensible Match (OXM). It allows to use various length match fields and easily implement new fields without changes in protocol. Usage of TLV structures increased flexibility of protocol and gave to developers possibility to use current switches for experiments with new data structures and behaviours.

5. IPv6 support added in version 1.2 and were extended in version 1.3 to support extensions of IPv6 header as hop-by-hop, router, fragmentation, authentication, encrypted secure payload and other IPv6 extensions.

6. Flexible table miss behaviour was added in version 1.3. Prior versions only allowed three options for packets that did not match any field send to controller, drop or send to normal processing. Version 1.3 introduces table miss as a separate flow entry which may have the same as other flows flexible sets of actions.

Other changes include:

- the structure of flowtable changed see picture:

Table 2.3: OpenFlow. Flow table v1.3

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

- introduction of logical ports which can represent tunnel or aggregated interface,
- implementing cookie into IN-PACKET message that simplified look-up for specific rule into controller increasing overall scalability,
- addition of masks to data layer and network addresses in match fields,
- support of SCTP.

After publishing specification 1.3 the problem of version compatibility and negotions between switch and controller became very clear and in version 1.3.1 a new field was added to Hello message that contains bitmap of all available version supported by device and provide easier negotiations process between controller and switches.

The newest version of OpenFlow specification 1.5.1 [7] was published on March 26th 2015 and has many major changes comparing to version 1.3, main of which is directed on increasing scalability of the OpenFlow environment. [specification 1.5.1]

1. Introduction of egress table in version 1.5, slightly changed the concept of pipeline and packet treatment as shown in figure 2.3. In previous versions all processing was done in the context of input table. Egress flow tables allows packet processing in context of output port.
2. Flow monitoring supports idea of multiple controllers that manage one switch. It allows controller to monitor in real time any changes that done to flow tables of the switch by other controller. Controller defines monitors and assigns subset of flow tables to it. When any change is made to this subset, monitor notifies controller.
3. Optical ports support added to protocol ability to treat layer 1/2 optical networks and embodied potential interest of optical network's providers. In combination with control of data flows, control of optical circuits allows to significantly increase utilisation of optical links without risk of congestions.

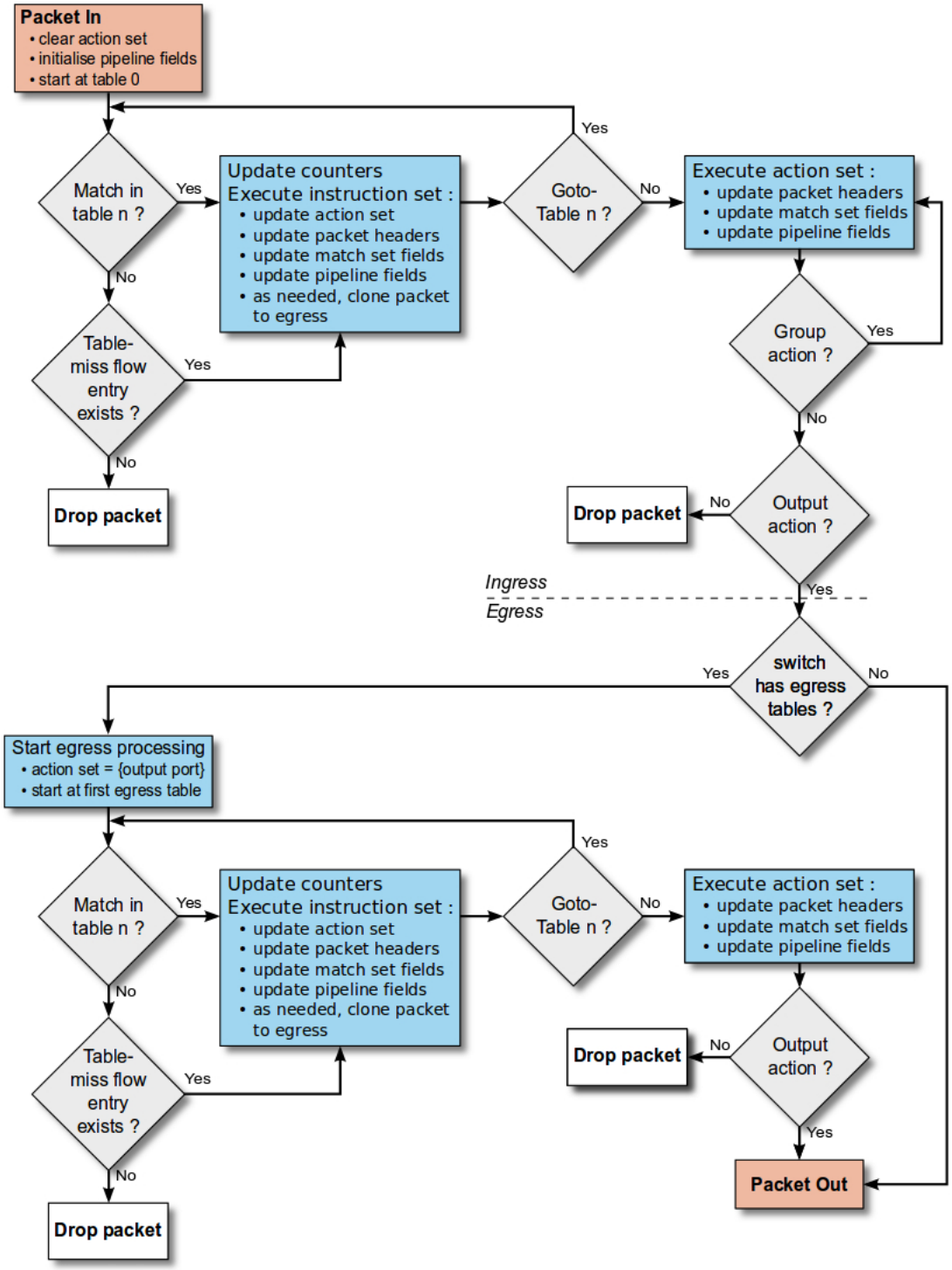

Figure 2.3: Packet Flow in OpenFlow v1.5

4. After successful introduction of TLV structures in match field (OXM), more protocol's previously fixed structures were also completed with TLVs. Additional extensibility allowed easier way to introduce new features to protocol. Also TLVs were introduced in statistic collection. OpenFlow eXstansible Statistic (OXS) structures are based on OXM and compatible with OXM.

5. Bundle mechanism introduced, that allows to treat groups of OpenFlow operations in atomic style. It simplifies synchronization of related changes in one switch as well as across multiple switches.

6. Packet type aware pipeline allows treat packets not only as Ethernet frames, but as PPP or IP data structures. A new OXM was added to specify the packet type in match field of flow entry. Since type is defined in extensible format it allows to easily introduce new types including experimental ones.

Other changes include:

- the structure of flowtable changed see picture:
- eviction ability of switch independently delete flows with low priority to insert higher priority flows in overloaded flow table,
- vacancy events gives to a controller early warning that table nearly full and will not able to add more flows soon,
- IANA allocated port 6653 to openflow, so it became a default port for switches and controllers,
- added wildcard field to SET_FIELD allows to change only specified bits of field.

Message exchanging protocol was also changed during the evolution. In sub-types of controller-to-switch messages Send-Packet message was changed to Packet-Out to be logically paired with asynchronous Packet-In message. Also 2 new sub-types were added:

1. Role-Request used to set and query role and controller ID of OpenFlow channel between switch and controller. Was introduced to support multiple controller's connections to same switch.

2. Asynchronous-Configuration used to set filter of Asynchronous messages controller wants to receive from the switch. Also added to support multiple controllers connected to single switch.

In asynchronous messages three new messages were added:

1. Role-Status used to inform controllers of change of switch role. For example when new controller elects itself master, the switch sends role-status to former master controller.

2. Controller-Status used to inform of change of OpenFlow channel status. This message supports fail-over processing if controllers can not communicate with each other.

3. Flow-monitor used to inform controller about changes in flow tables. Introduced to support multiple controllers connected to single switch.

The Error sub-type was moved from asynchronous messages to synchronous to support bidirectional error notifications and guarantee delivery at OpenFlow protocol level. Another change in synchronous messages is Vendor sub-type was renamed to Experimenter and defined as message that provides way to offer additional functionality from switch to controller within OpenFlow standard message.

There are many more minor changes were done since version 1.0. It should be enough to notice that switch specification's volume increased from 42 pages in December 2009 to 283 pages in March 2015. Protocol became all-embracing and flexible, but in the same time very comprehensive. This increasing comprehensiveness slightly violates one of original intents of SDN to simplify network configuration and management that declared in [3,4],

but developing of highly abstract configuration tools can compensate it. OpenFlow is not only de-facto current standard for SDN control communications, but also very efficient tool for SDN developers and researchers to use in experiment environment.

2.3.2 Port Mirroring With OpenFlow

Unlike the fundamental functions as topology discovery, forwarding or filtering, traffic mirroring usually is not implemented by default and needs additional and in some cases complex configuration. In OpenFlow protocol the port or traffic mirroring is not implemented as complete function or action, but there is a possibility to mirror traffic since specification 1.3, where Group tables were introduced first time. To understand how to configure port mirroring with OpenFlow, we need first to explain what is Group table and how does it work.

Group tables are used to implement network functions that need to be aware of group of ports instead of single port as it was before version 1.3. Group table contains group entries which in its way consist of 32-bits ID, type of group, assigned counters and list of action buckets assigned to the entity.

Table 2.4: OpenFlow. Group table

Group Identifier	Group Type	Counters	Action Buckets

Action bucket is another name of action set specific for the groups. When packet hit the group list, actions from one or more action buckets could be applied to it. Action set is associated with each packet and is empty by default. While packet is going through pipeline each flow entry may modify action set by add, remove or modify actions in it. All actions in action set are execute simultaneously when pipeline is over and "next flow table" field is empty or when flow has instruction "Apply-Actions". This instruction is used when it is necessary to change the packet between flow tables or to apply more than one actions of one type to the packet.

Each action bucket may consist of set of actions, but only one action of each type as well as in usual action set. For example action "output" may present only once and thus each action bucket supports only one port.
Groups could be one of four types:

1. All - the group where all buckets are must to execute. This type of group effectively used for flooding, e.g. multicast or broadcast. The packet is cloned for every bucket in the group and proceed independently after.

2. Indirect - the simplest form of group. In contrast to "all" group consist only one bucket. Provides faster convergence, e.g. packet forwarding.

3. Select - executes one of the buckets of the group. Used for effective load balancing. The bucket has to be chosen by special load sharing algorithm that bases on computation of hash or simple round robin. The algorithm may also use bucket weights providing more flexible load sharing.

4. Fast fail-over - executes first live bucket. This group provides network users with changing forwarding path without excessive round trip to controller.

Last two groups are not yet required to implement in OpenFlow switch even though they provide very useful functionality and marked in the specification as optional.
There are two ways to implement port mirroring in OpenFlow v1.3 and later. We will take version 1.3 as most popular and also because it has additional functionality e.g. group table. We will explain how port mirroring works on simplest network design - single switch that connects three hosts: source, destination and the probe as shown in figure 2.4.

Switch has single flow pre-installed, it matches all traffic that hits ingress port and sends it to egress port. it has only one flow table and hence only one variant of the pipeline.
First way to execute port mirroring is to create group table of type "all" with egress and mirror port in it and modify the flow to make it send the traffic to this group instead of

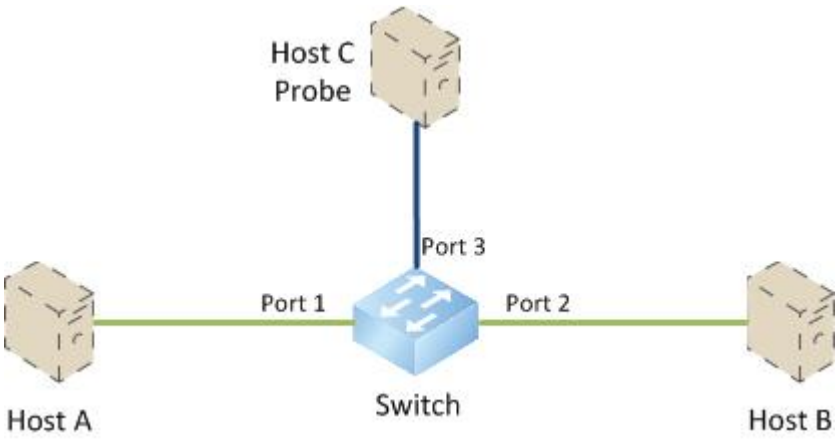

Figure 2.4: Simple network

port. Thus we will have our traffic from ingress port simply flooded to egress and mirror port. Notice there is no changes to traffic itself, hence destination host will have no knowledge of port mirroring in the middle. Since OpenFlow treats traffic as flows and all flows are unidirectional, we will need additional configuration to implement bidirectional port mirroring in this network. To do this we need to create second group table of type "all" which will contain ports ingress and mirror. Then we will need to install a flow that matches all incoming traffic on port egress and sends it to that second group.
The convenience of this solution is inversely to quantity of flows installed on the switch. If we would have hundreds or thousands of flows that treated differently and thus have different pipelines, we then would need to find all the last tables that each flow that came from port we need to mirror is hitting and modify every single pipeline adding the rule to send traffic to the group. The natural advantage of OpenFlow that if you do not need to mirror a port but only mirror some traffic, there is no additional filtering needed and configuration is simple.

The second way is to use "Apply-Actions" instruction in the first flow of the first table. This instruction contains Action list that executes on packet immediately. If action list contains a forwarding action, a copy of the packet will be send to the desired port i.e. mirror port, while packet itself will continue to the next flow in the flow table, which

in case of our simplest scenario will contain forwarding action to the egress port. To implement bidirectional port mirroring, same as in first case, we will need to install a flow that matches traffic from egress port and set instruction "Apply-Actions" to copy packets towards mirror port.

In contrast with first option, this method has no limitation related to quantity of flows as the "Apply-Actions" instruction may be executed in very first table that every incoming packet is hitting. Yet the major drawback of this method is that instruction "Apply-Action" is optional, thus there is no guarantee that OpenFlow switch will support this function.

2.4 Open Network Operation System - ONOS

ONOS is an open source SDN controller which is developing under orchestration of ON.LAB (Open Networks Laboratory) non-profit organization that was founded in 2012 with the assistance of Stanford University and University of Berkeley particular for development of the SDN controller. ON.LAB positions ONOS as SDN controller for telecommunications companies and service providers. Thus the main features of the controller are scalability, high availability and performance. Another major specific of the controller is multi-layer control. It means that controller may orchestrate not only data at L2/L3 and higher levels but also underlying optical network and its links at L0. Providing separate instruments to control those levels, ONOS unites views of them in single interface, equips system administrator with a complex view of the network. By managing both optical and packet layers, controller may provide complex load balancing, thus increasing effectiveness of network even further than SDN concept provides itself. Also it significantly simplifies management of complex networks. Controller itself represented by collection of applications written on Java that interact with each other through Java APIs. ONOS also provides REST APIs with similar functionality as one of Northbound interfaces of the controller. Applications are running in Apache Karaf container. Developers may choose

from deploying their applications into the container and use Java APIs of other components or use REST API and keep application separated. ONOS provides wide range of instruments for development of controller based applications, including templates and simple integration into command line or graphical user interfaces.

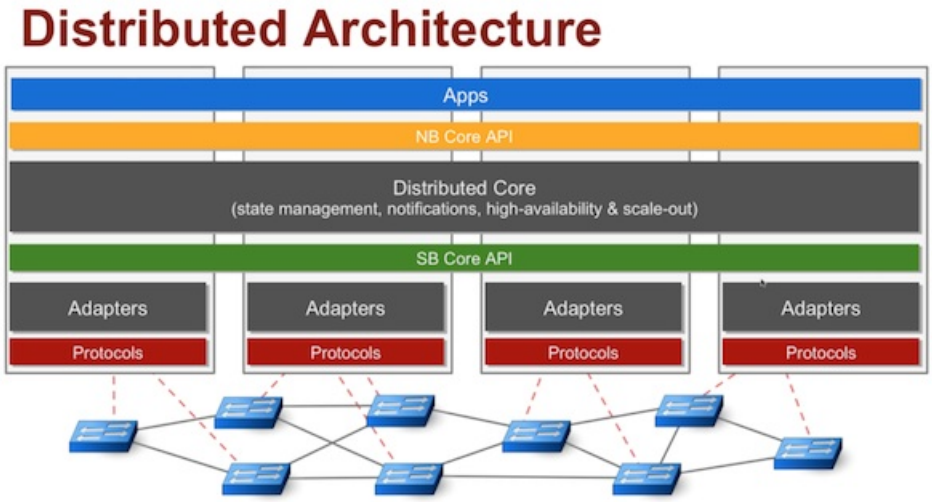

Figure 2.5: Architecture of ONOS [23]

There are two main abstractions that controller provides for Northbound applications: global network topology view and intent framework. Applications use topology API to calculate paths, provision flows and perform other network functions. Since ONOS controller's developers position it as easy scalable, distributed and highly available, the primary concern in topology view is the maintaining of the consistency of the view. Each instance of the controller keeps full global view and synchronize it periodically with another instance that it randomly choose. Intent is an abstraction, which represents the need or desire of connection between two or more points, but does not specify particular path. Such intents could be installed by user or application such as onos-app-ifwd. When intent is installed, controller itself finds the best path and injects flows to the switches. The main benefit of installing intent for user is that in case of changing of topology, controller will dynamically change flows on switches to provide high-available connection while it

is physically possible. Figure 2.6 shows the Intent life-cycle since request till installation and withdrawing. Orange states are transitional and intents exist in these states briefly, while gray states are so called parking states and intent can be kept there for long time.

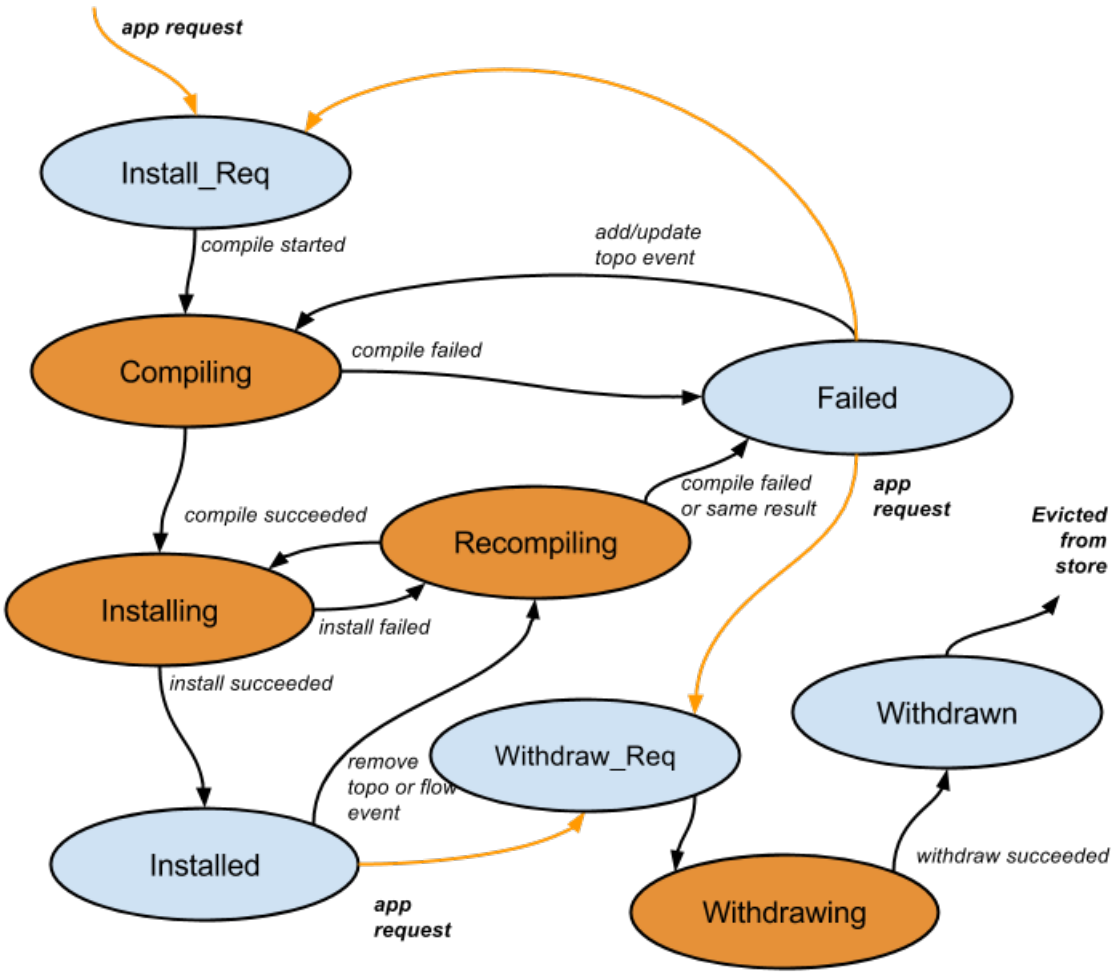

Figure 2.6: Life-cycle of Intents in ONOS [23]

ONOS version 1.2 provides a few interfaces to administrate network. First is REST API that primary used by developers, but also might be used by administrator even it is too comprehensive for day to day use. Next is Command Line Interface that provided by extensible functionality of Apache Karaf's command line interface interface. It provides access to main network functions such as maintaining flows on switches and provision network elements as switches and hosts. CLI also provides management of intent framework

which makes it useful interface for network administrator. The last interface is web based graphic user interface. Its primary function is to provide complex graphic representation of network topology, while it also allows to install (but not modify or delete) intents to the controller in literally three clicks. Interface as shown in figure 2.7 provides functionality to provision intents and flows, dynamically measure load of channel between switches and shows which instance of the controller is master for particular switch.

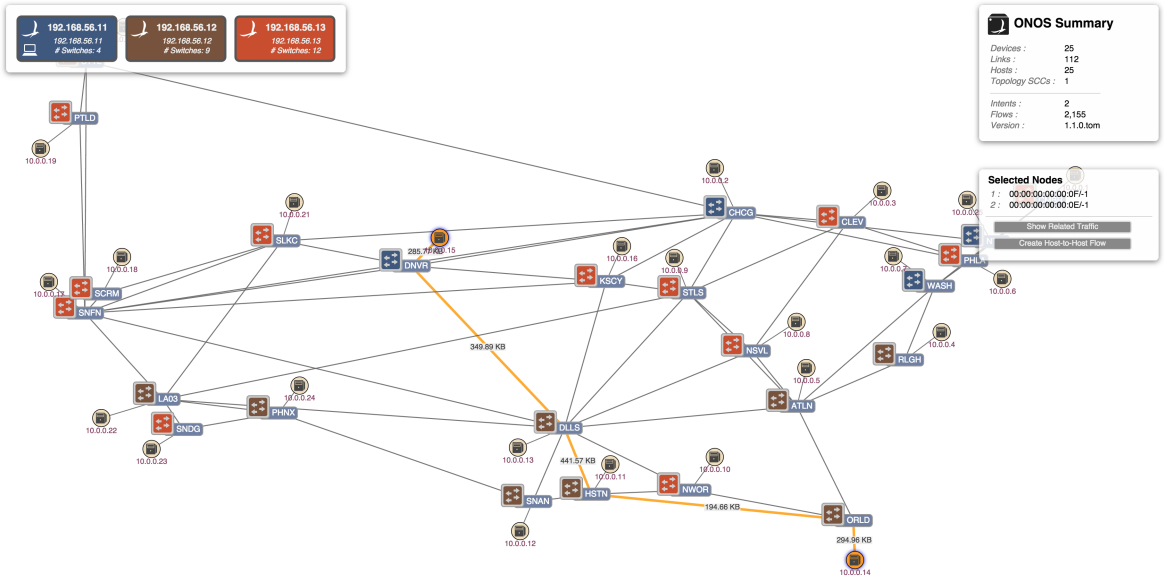

Figure 2.7: Topology view by ONOS graphic user interface [23]

2.5 OpenDaylight - ODL

OpenDaylight is open source SDN controller project which is developing under control of Linux foundation. The project was started in 2013 and widely supported by industry members and researchers. The targets of project are accelerating of adaptation and creating more transparent approach to Software-Defined Networks and also building solid basement for Network Function Virtualization. Software for the controller is written on Java. In opposite to other open source controllers as ONOS and FloodLight, ODL is developing by community without strict lead. The main decisions in development

are making by voting in Technical Steering Committee which elective governing body. Another major governing body is the Board of Directors which is formed of partner's representatives. No industry member can have more than one seat at the Board of ODL; it allows keeping ODL single member independent. The Board of Directors is in charge for business strategy making operational and marketing decisions. From second release Helium controller's software is running in OSGi container (currently Apache Karaf) as it allows to dynamically add and remove modules and to establish interconnections between them. The third and latest by the time of this work was written release Lithium was released in June 2015 and expected to become a basement for more than 20 commercial products. One of the main non-commercial use cases of the ODL is providing network services for OpenStack cloud platform.

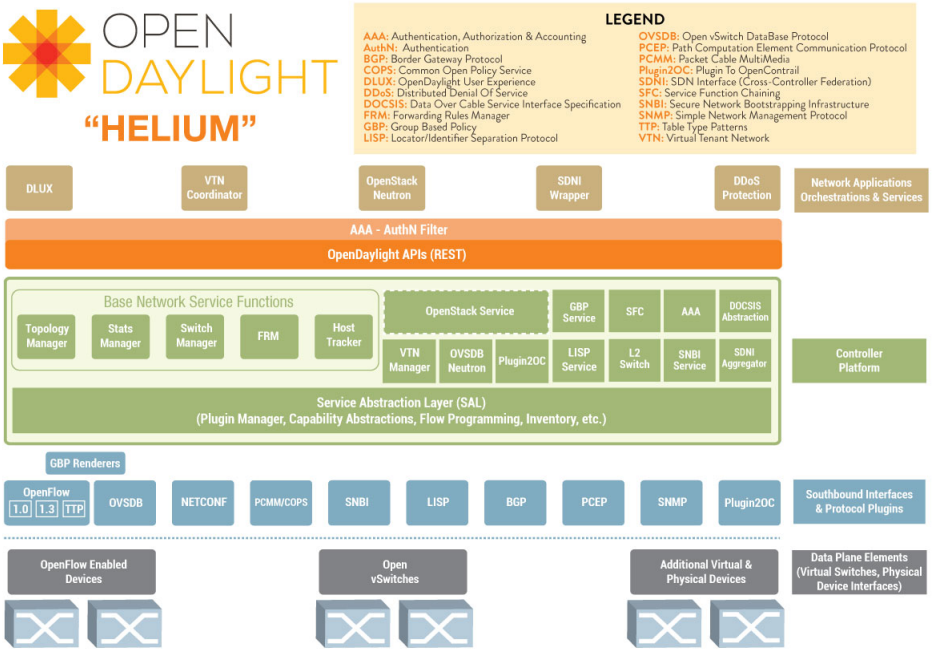

Figure 2.8: Architecture of ODL [24]

Controller provides two interfaces for applications. OSGis interface is accessible for applications that are in the same address space as controller and web-based REST interface for everybody else. Unlike ONOS, ODL does not provide any user-friendly interface

that would allow configuration or data collection by not well trained user. Most of interfaces of the controller are represented in web-based interface DLUX. DLUX is essentially a collection of interfaces which provide access to the controller's functions including visualized REST interface of the controller. Thus to be able to manage network in DLUX, user has to has additional skills in RESTful services, YANG language and comprehensive understanding of the controller structure. As it was said before, DLUX interface contains of modules that provide manageability of different controller's modules. These modules should be installed in Karaf container to be seen. There is only one module installed by default Topology. Topology provides simple schema or map of interconnections of devices (usually switches) connected to the controller and hosts that sent at least one packet through the network. Topology has very limited functionality and shows only IDs of devices, MAC addresses and IP addresses of hosts. Statistic module of DLUX called Nodes provides information about network devices and links including IDs and MAC addresses of all ports of devices under control, detailed packets statistic and shows flows installed into device by OpenFlow protocol.

Another module is YANG user interface. YANG itself is a data modelling language for NetConf protocol that recently became popular in other technologies including OpenDaylight controller. YANG is standardised by IETF and described in RFC 6020. Yang UI module consist of a collection of all accessible REST API's of the controller. Each element unites information about particular component of the REST API and YANG data structure tied to that component. Depending on the essence of the element it may provide HTTP GET, PUT, POST operations as well as some specific instruments as Display Topology in network-topology element. Using YANG UI module user can configure network or get detailed information from the controller, but as it noticed before it is not trivial task.

Group Based Policy (GBP) is technology inherited by OpenDayLight from Cisco. At some degree it is analogue of ONOS's Intent framework and provides similar functionality, but in much more flexible and comprehensive way. It could be accessed through the

DLUX interface or REST API. GBP can not be accessed through CLI interface, but has very well defined graphic interface which provides full functionality of module. GBP has comprehensive architecture and requires advanced skills and deep understanding of the model from the user, while providing a great deal of flexibility of configuration. GBP also provides well defined filtering, allowing user to define intents or connections at transport layer as well as at network and data-link layers.

2.6 Use Case: Probing In Telecommunication Networks

To explain use case we will first take a look on general architecture of mobile operator network, in particular architecture of Long Term Evolution (LTE) network shown in figure 2.9. Then we will discuss specific of port mirroring in SDN.

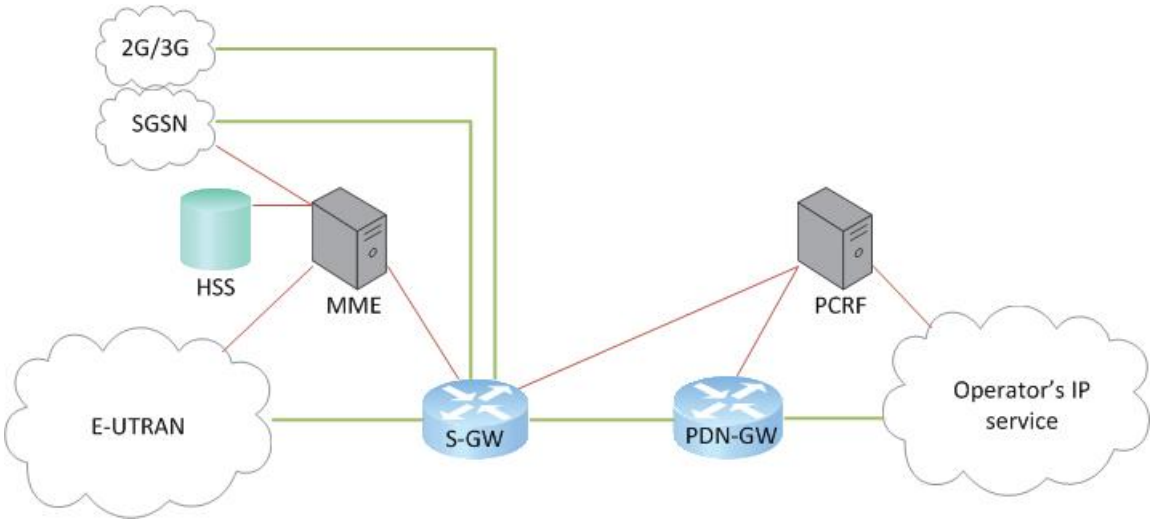

Figure 2.9: Architecture of LTE network

The E-UTRAN element at the left lower corner stands for the Evolved Universal Mobile Telecommunication System's (UMTS) Terrestrial Radio Access Network. E-UTRAN

represents 4G radio network that consists of all 4G base stations ans all connected user equipment e.g. mobile phones, 4G modems, etc. User's traffic, for example HTTP query to WWW, goes from E-UTRAN to Service Gateway (S-GW) then to Packet Data Network Gateway (PDN-GW) and finally to the Internet. To understand specifics of traffic forwarding lets take a look on protocol stack that uses in LTE network for user's traffic that shown in figure 2.10.

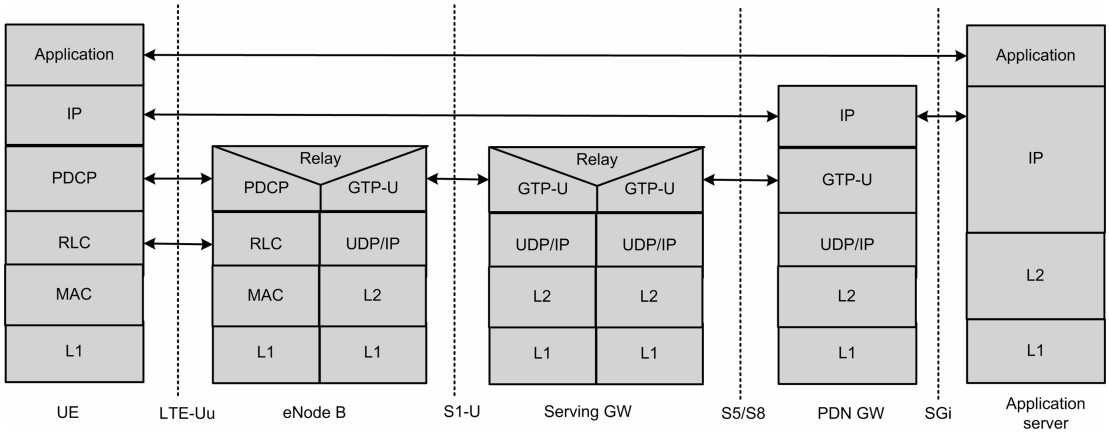

Figure 2.10: Protocols stack of user plane in LTE network [25]

In this picture eNode B stands for equipment of the base station. User equipment (UE) sends IP traffic to the base station encapsulated in specific radio protocol's data units. Base station decapsulates IP traffic and encapsulates it in General Packet Radio Service (GPRS) Tunnelling Protocol for user traffic or GTP-U. Then it adds UDP header with destination port number 2152 and IP header with its own IP address. Next base station sends users encapsulated user traffic towards S-GW through Evolved Packet Core (EPC) network. S-GW main functions are redirection of user's traffic i.e. handover between base stations and interoperating with other networks e.g. 2G and 3G. Then S-GW sends traffic to PDN-GW, which is connecting host between LTE network and other networks as Internet. The reason to use tunnels sending traffic between base station and PDN-GW is to be able to treat different traffic from different users with different quality-of-service and also to ease handover. There is at least one tunnel for every connected UE. UE

may initiate more than one tunnel, for example if it needs to send traffic with different quality-of-service. Since all user traffic encapsulated within tunnels and forwarded based on tunnel's header and IP/UDP headers of EPC network, neither eNode, nor Serving Gateway have knowledge of user's query IP/TCP/UDP headers. PDN-GW in its way has no knowledge of what base station is servicing user at the moment as the only information it has is IP address of Serving Gateway and S-GW maps tunnel ID to eNode IP address. Probing of users traffic while it is still encapsulated in GTP is very source-demanding operation. Thus such probing on core network speed is very expensive and inefficient. Instead of this operators accomplish probing off line using port mirroring. This allow to cache a portion of traffic first and execute analyse on it with lower speed. Figure 2.11 presents simplified design of EPC network with a probe .

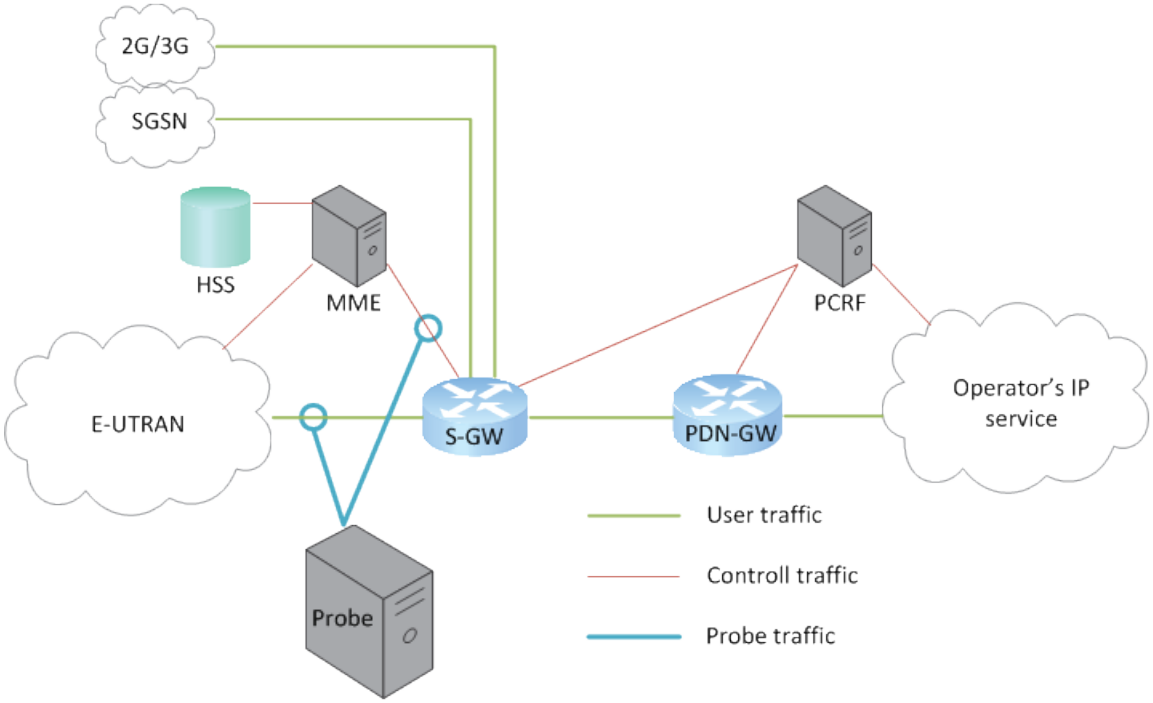

Figure 2.11: Probe in EPC network

Packet inspection at this point gives both understanding of payload and base station address. To implement probe between E-UTRAN and S-GW, operator uses switch with port mirroring function configured to copy traffic and send it to the probe. It makes port

mirroring one of compulsory functions for carrier networks.

The telecommunication networks are currently switching from physical devices to virtual machines, from middle boxes to virtual network functions (VNF). There is a new word in industry emerging - virtual telecommunication providers. Tesco mobile is one of the first operator which has nor radio network neither core network. Instead Tesco simply leases infrastructure along with core network [17]. Core networks for virtual operators are built with virtualized network functions rather than physical devices [4], which makes it easy to start a new virtual provider. There were 943 mobile virtual network operators by 2014 [1] and there are no doubts that with development of NFV this number will increase significantly in near future. To support common trend, telecommunication companies look forward to adopt SDN technologies to improve network functionality, increase utilization and flexibility. To provide probing functions in SDN networks operators will need to be able to use port mirroring function. Further in this work we will configure this function using various interfaces of ODL and ONOS controllers.

In this chapter we explained the problem of probing in telecommunication networks, introduced OpenDaylight and ONOS SDN controllers with interfaces that we will use for configuration of port mirroring that supports probing. We also took a close look at OpenFlow protocol, its evolution process and determined it is fully suitable to configure port mirroring. We also took a look on some classic network functions provided by layer 2 protocols of OSI model and how this and some other functions are provided in SDN environment.

Chapter 3

Design

In this chapter we will define design for port mirroring using test network and four northbound interfaces of open source SDN controllers. First we will define test network topology, then we will speak about ONOS initial setup design and describe design of port mirroring with Intent framework of ONOS and REST API of ONOS. Then we will describe OpenDaylight's initial design and describe port mirroring design with Yang interface and Group Based Policy interface.

3.1 Test Network Design

For the purpose of this work we used test network that described below. Test network is simulated with Mininet network simulator and has one switch with 3 ports, three hosts with one connection to the switch and SDN controller directly connected to the switch. Host A represents E-NODE from use case, host B represents S-GW and host C represents the Probe as shown in figure 3.1. All hosts are in the same IP address space, so port mirroring could be done with switching.

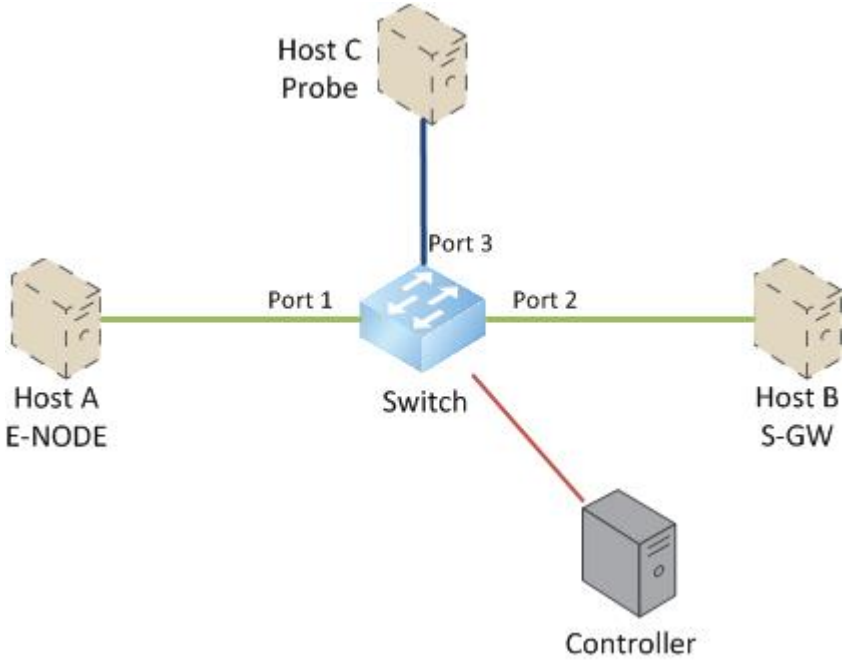

Figure 3.1: Test network

3.2 ONOS

In order to use northbound interfaces, ONOS version 1.2.1 Cardinal is used. There are at least two machines needed to run ONOS, first machine physical called "build" machine, second - virtual called "run". Since ONOS designed to work in virtual environment, "run" machine is virtual and has run under VirtualBox hypervisor installed on "build" machine. "Build" machine compiles Maven project and installs it to "run" machine. ONOS is running in Karaf container. The access to the controller's interfaces provided by "build" machine. In order to use Intent framework special application org.onosproject.ifwd should be installed in Karaf container.

3.2.1 Intent Framework

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: 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 will use CLI to install intents. A detailed look and analysis of ONOS command-line-interface's commands that manage network at layer 2 of OSI model provided below. All commands could be divided in two groups: show commands and configuration commands.

Show commands:

1. Devices shows all devices (usually switches) currently connected to the controller or the cluster if it is distributed controller implementation. Contains information of availability (true/false), its role (master/), device ID inside cluster, device type (switch, router, etc.), manufacturer, hardware, software it runs, devices serial number and version of OpenFlow protocol it is running.

2. Hosts shows all hosts that connected to the network and had network activity (disclosed themselves). Information consists of ID in cluster (mac-address plus identifier in case there are similar mac-addresses in the network), mac-address, ID of device it connected to, vlan ID and ip addresses of host.

3. Links shows all links between devices. All links are unidirectional, general connection between switches will be represented as two links with opposite sources and destinations. Information consists of source of link, destination, is it direct link and state (active/inactive).

4. Flows shows information about all flows currently installed on devices in cluster. Information consists of device ID the flow installed on, number of rules in the flow, flow ID, flow state (pending added, added, pending removed, removed), the counter of flow (how many packets matched the rules of the flow), the timer of flow, flows priority, id of application on the controller which installed this flow and the rules itself. In its turn rule consist of selector (the match rule) and treatment (action). Flows could be unidirectional in this case they consist of one rule or bidirectional and consist of two rules, in some cases flows could consist more rules. Flows could

be installed in different ways. First it could be installed by applications, for example by regular ONOS forwarding application (onos-app-fwd), as reaction on packet that switch is sending to the controller when such packet arrives to the switch and do not match any existing rule. By default in OpenFlow networks when switches receive packets that do not match any rule they send first 100 bytes of the packet to the controller and wait for its decision. This approach called reactive forwarding. There are other options. Packets that do not match the rules could be dropped with or without sending notification to the controller this approach called proactive forwarding. Switches also may proceed such packets with traditional forwarding using forwarding tables other than flow table. Usually this approach called hybrid forwarding. In proactive and hybrid approaches flows could be installed both by application automatically, for example as reaction on policy or intent installed into controller, or by user in command line interface (onos-app-cli) or graphical user interface (onos-app-ui).

5. Paths this command takes IDs of two devices as arguments and shows shortest paths between them calculated by the controller. The information about path consists of IDs and ports of all devices on the path and paths cost. The output of this command is currently the main source of information for applications that want to insert flows to the devices. For example if application needs to setup connection between points A and B, it first will ask topology application about paths from A to B and B to A. Based on output it may construct particular flows and inject them to switches.

6. Intents shows all intents currently installed in the cluster. The information shown by command contains ID of intent, its type, state (submitted, compiling, compiled, installing, installed, recompiling, withdrawing, withdrawn, failed), ID of application that installed intent and constraints (Weights applied to a set of network resources, such as bandwidth, optical frequency, and link type). Detailed view can be invoked by using -i argument, shows selector , treatment and path that used to implement

these intents.

Configuration commands:

1. Add(Remove)-flow installs/removes specified flow to/from specified device. The flow may have timer to live. For example flows that installed dynamically by onos-app-fwd have timer set to 60 seconds, that means if no packet hit the rule in 60 seconds, such flows will be removed from device.

2. Add-hosts-intent, add-point-intent, add-single-to-multi-intent, add-multi-to-single-intent - install different types of intents to the cluster.

3. Remove-intent removes specified intent. As arguments this command takes ID of application and ID of intent. Controller keeps removed intents marking them with state withdrawn. To completely remove intent administrator should use -r argument.

To configure port mirroring with Intent framework, one has to create single-point-to-multipoint intent where define ingress port and two egress ports. To discovery currently installed intents "intents" show command has to be executed. To modify intents one should first delete old intent and create a new one after that. To delete intent, "remove-intent" command should be used.

3.2.2 REST API

REST API of ONOS allows user to check the network topology, browse switches, hosts and links. Also it allows to insert, modify and delete flows and intents. Configuring of port mirroring is not as simple and native as with CLI, but still possible and might be preferable in some cases. For example if access to REST API implemented in users application.

There are two options to configure port mirroring with REST API: to use intent framework

or flow framework. There is significant difference in these two options, if one has to implement port mirroring in network with more then one path between probe and mirrored port, where probe does not connect to the same switch as mirrored port. If controller dials with expressed intent it defines a path itself and decides which flows to install in order to provide particular intent. Doing this controller relies on own information about network and able to choose optimal configuration. It also provides high availability and load balancing for intents where it is possible, means if underlying network has excess physical connections. If one configures port mirroring with flow framework i.e. statically installing OpenFlow flow tables, such configuration will not be changed automatically in case of network failure even if alternative path exists. This difference is reflected in figure 3.2.

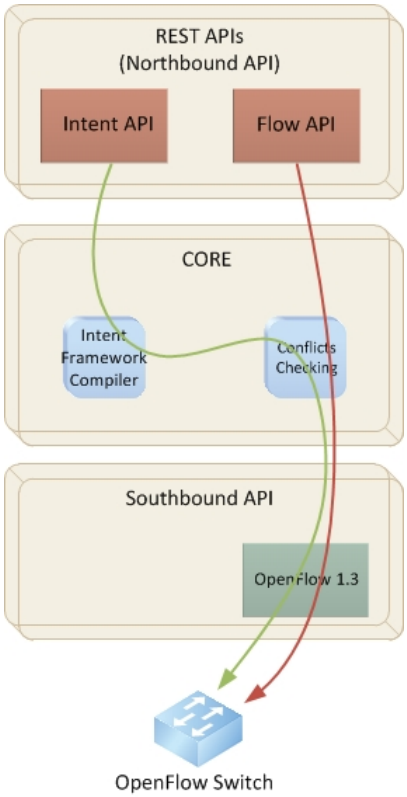

Figure 3.2: Intent API versus Flow API

Two configure port mirroring with REST API using intents, one has to install one intent for every flow or traffic direction. To do same using flows, one has to install

corresponding flows to the switch.

3.3 OpenDaylight

In order to use northbound interfaces of OpenDaylight controller, Lithium release should be installed. OpenDaylight controller as well as ONOS, represents a stack of applications that are running inside Karaf container. There are basic applications that installed and run by default and applications that provide specific features. From second group of applications one has to install yang, restconf, l2 switch host tracker and dlux graphic interface.

3.3.1 Yang User Interface

Yang user interface is an application based on DLUX graphic user interface in ODL. Its main function is to generate simple user interfaces based on YANG model loaded in OpenDaylight controller to simplify and facilitate development of such applications. Resulted interface allow to add, modify and delete data structures that application works with. Figure 3.3 shows the view of Yang interface in DLUX GUI.

Figure 3.3: Yang user interface

We will use Yang UI to access data structures of application "opendaylight-inventory" (ODLI). This application provides to user ability to configure network devices with same commands as they would use in pure OpenFlow protocol. The main difference is that commands do not go strictly to the switch, but comes to the controller's core i.e. service abstraction layer where process of compilation is executed. If changes expressed with the command do not conflict with existence policies, controller propagates them to south-bound interface, where one of drivers sends it further to the switch. Thus policy that was expressed in OpenFlow data structures might be implemented into the switch with different protocol e.g. OVSDB or NETCONF as shown in figure 3.4.

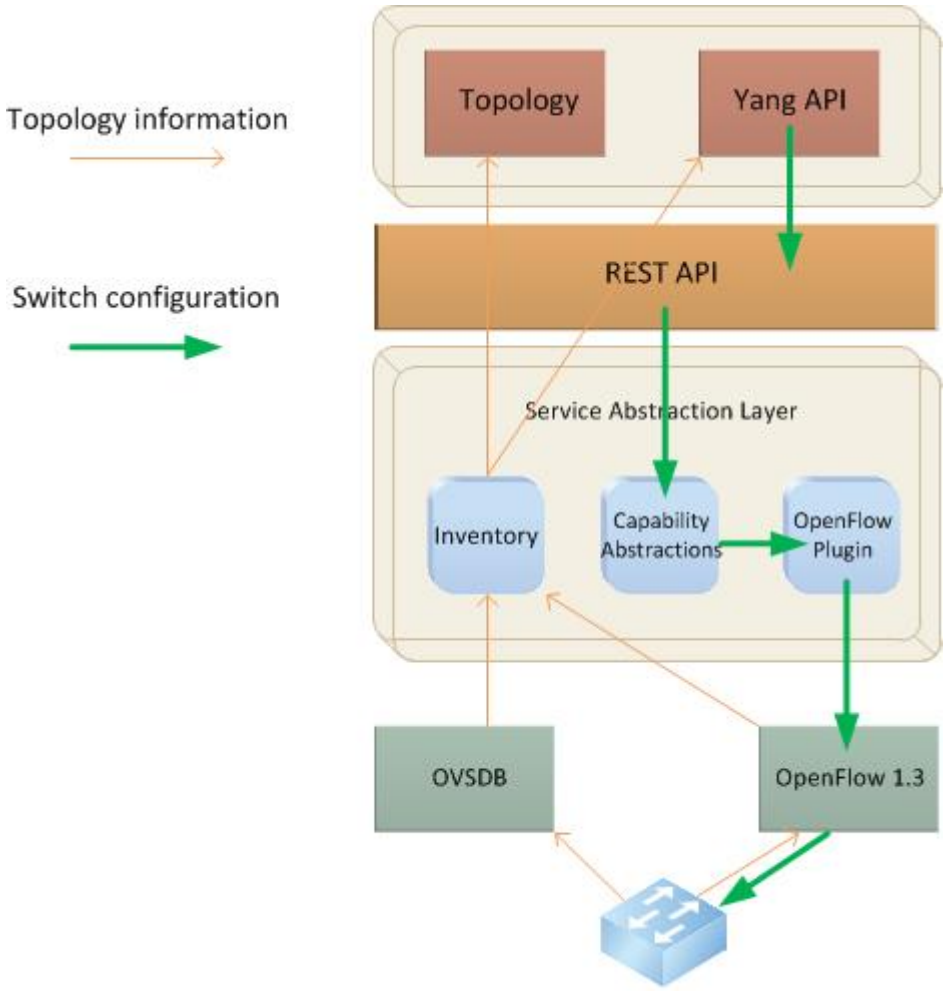

Figure 3.4: Yang UI and Service Abstraction Layer

ODLI Yang interface provides graphical representation of OpenFlow structures e.g. flows, tables, instructions etc. as a tree structured sequence of fields, pull-down menus and check-boxes. To execute command, user has to choose one of RESTful methods and press "send" button **??**. When command is executed, Yang UI creates JSON string and sends it towards RESTCONF API of the ODL. Thus Yang UI, in this particular case, should be considered as auxiliary interface.

To configure port mirroring in test environment with ODLI, requires to install two flows. First flow matches traffic with ingress port 1 and consequentially sends copies of it to ports 2 and 3. Second flow matches traffic with ingress port 2 and sends copies of it to

ports 1 and 3.

3.3.2 Group Based Policy

OpenDaylight Group Based Policy allows users to express network configuration in a declarative versus imperative way [18]. GBP's functionality could be reached in two ways: through REST API and GUI that implemented in DLUX interface. For the purpose of this work GUI interface was chosen.

To work with GBP in GUI one has to understand GBP's model shown in figure 3.5.

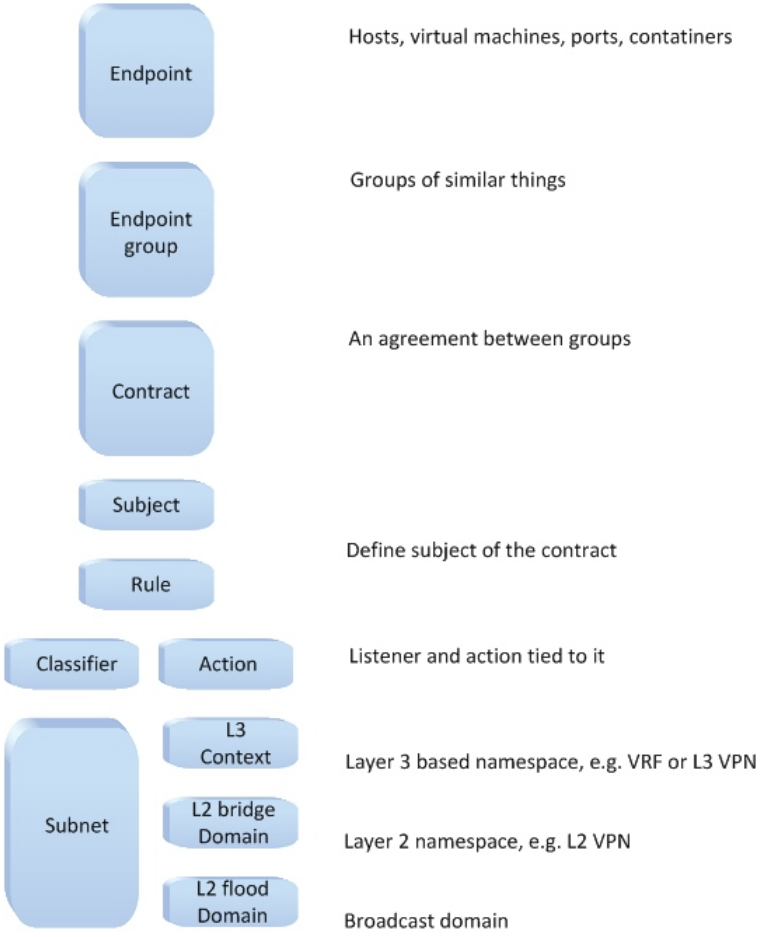

Figure 3.5: GBP model

To expose simple policy in GBP, it is need to define end point groups, endpoints themselves, layer 2 and 3 network contexts, define contract and include this contract in

corresponded provider and consumer selector lists. This lists bind end point groups with contracts with explicit notation who is provider and who is consumer of this contract. Then in contract itself one should define subjects, rules sets for this subjects, classifiers and actions for these rules. Currently there are only two options in actions: allow and send to service function chain application of the controller. The allow action does not provide flooding or copying of traffic, thus can not be used for purpose of port mirroring. Service functions chain currently does not provide port mirroring service, but such service might appear soon as SFC is currently under development, hence we will provide design for future needs. Classifiers are also limited. There are only three options of match fields for classifiers are currently exist: Ethernet type, IP protocol, L4 protocol. There is no notion of port in GBP classifiers, thus pure port mirroring is currently can not be configured. Instead for the purpose of specified use case, traffic of Ethernet type 0x0800 and 0x0806 might be matched and mirrored.

It follows from the above that it is currently impossible to configure port mirroring using this interface. Nevertheless we anticipate flooding behaviour to be added during further development process, thus we provide hypothetical design for this interface as if flooding behaviour exist.

To configure traffic mirroring using group based policy, one has to create two groups and assign a contract to them. In contract one has to specify classifier to match IP and ARP traffic. Then one has to use flood action to broadcast traffic to the group that consists of destination host and probe.

3.4 Design Overview

In this chapter we took a look on design of port mirroring while using different northbound interfaces of ONOS and OpenDaylight controllers. It is safe to say that from the point of view of design there is no significant difference between Yang interface of OpenDaylight and REST interface of the ONOS. In other hand Intent framework of ONOS

simplifies configuration by providing easy-to-understand abstractions, while also provides high availability where and when it is physically possible. Finally Group Based Policy model appear not to be developed enough to provide functions distinct of forwarding and filtering. Nevertheless we provided hypothetical design as if flooding behaviour exists in GBP interface.

Chapter 4

Implementation

In this chapter we will give detailed description of configuration of port mirroring with chosen interfaces as well as describe test environment and its installation steps. We will also take a close look on controllers installation and configuration process. First we will describe how to install and configure ONOS controller and how to configure port mirroring using Intent framework and ONOS's REST API. Then we will describe how to install and configure OpenDaylight controller continuing with configuration of port mirroring using Yang interface. We will take a look on hypothetical configuration process og port mirroring using GBP, as we anticipate required changes to be done in this interface during further development process.

4.1 Test Environment Configuration

In test environment we use physical machine with Ubuntu 14.04 LTS on it. On this computer Virtual Box hypervisor installed, that required to create virtual machine for ONOS. To create virtual network with OpenFlow switch and three virtual hosts, Mininet network simulator is used. The command to run test network is: *sudo mn –topo single,3 –controller remote, address=127.0.0.1, port=3356*. To test the absence of reachability before configuration command *pingall* used inside simulation. There is also possible to

define virtual hosts IP and MAC addresses using *h[1,2,3] ifconfig* command. To check port mirroring functionality Wireshark packet sniffer has to be installed.
To compile and run ONOS controller, Java 1.8 should be installed as well as Maven tool. To run OpenDaylight Java 1.7 should be used, thus it also has to be installed on physical machine. Both controllers run in Apache Karaf container, thus it also should be pre-installed.

4.2 ONOS

To install ONOS to the test environment one has to create virtual machine using VirtualBox hypervisor and install Ubuntu server 14.04 on it. Virtual machine minimal requirements are 2 GB RAM, 2 processors and 5 GB of disk space. SSH keys has to be generated on both machines to support secured tunnelling between them. Git version control has to be installed on physical machine to download ONOS distributive. To do this command *git clone https://gerrit.onosproject.org/onos* has to be executed. Then one has to export a set of environment variables by executing provided script */onos/tools/dev/bash-profile*. Then one has to build ONOS with maven tool, by executing *mvn clean install* command from ONOS root folder. After that one has to specify cell definition file which include address of "run" machine, address of network for inter-ONOS communication and list of applications that should be installed by default. One has to create new cell file in */onos/tools/test/cells/* folder and name it "tutorial", where define parameters as shown below:

- export OC1="192.168.56.101"
- export OCI="192.168.56.101"
- export `ONOS_APPS`="drivers,openflow,proxyarp,mobility,ifwd"
- export OCN="192.168.56.102"

- export `ONOS_NIC`="192.168.56.*"

Then one has to execute the cell script.

To be able to export ONOS to "run" machine, one first has to execute command *onos-package* which creates tar archive in */tmp* that contains binary version of ONOS. Next step is to export ONOS to "run" machine with command *onos-install -f $OC1*. When ONOS is running its CLI interface might be accessed with executing *onos $OC1* command. ONOS web based graphic user interface might be accessed at:
http://192.168.56.101:8181/onos/ui/index.html.

4.2.1 Intent Framework

To implement port mirroring using Intent framework one has to create two intents that will match all incoming traffic on ports 1 and 2 of test switch and broadcast it to port 2,3 and 1,3 respectively. It has to be done by executing next commands in Karaf's CLI:

- 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

After execution of these commands two intents will be created and automatically translated to required OpenFlow's commands that will be installed to a switch. To verify configuration one can execute *intents* command of CLI. As result of this command next two intents should be printed as shown in figure 4.1. As another option one can verify installed intents by looking into graphic user interface and using *show all related intents* option as shown in figure 4.2.

```
id=0x10003c, state=INSTALLED, key=0x10003c, type=SinglePointToMultiPointIntent, app
Id=org.onosproject.cli
    constraints=[LinkTypeConstraint{inclusive=false, types=[OPTICAL]}]
    ingress=ConnectPoint{elementId=of:0000000000000001, portNumber=1}, egress=[Conn
ectPoint{elementId=of:0000000000000001, portNumber=3}, ConnectPoint{elementId=of:00
00000000000001, portNumber=2}]
id=0x10003f, state=INSTALLED, key=0x10003f, type=SinglePointToMultiPointIntent, app
Id=org.onosproject.cli
    constraints=[LinkTypeConstraint{inclusive=false, types=[OPTICAL]}]
    ingress=ConnectPoint{elementId=of:0000000000000001, portNumber=2}, egress=[Conn
ectPoint{elementId=of:0000000000000001, portNumber=1}, ConnectPoint{elementId=of:00
00000000000001, portNumber=3}]
onos>
```

Figure 4.1: Show intents command

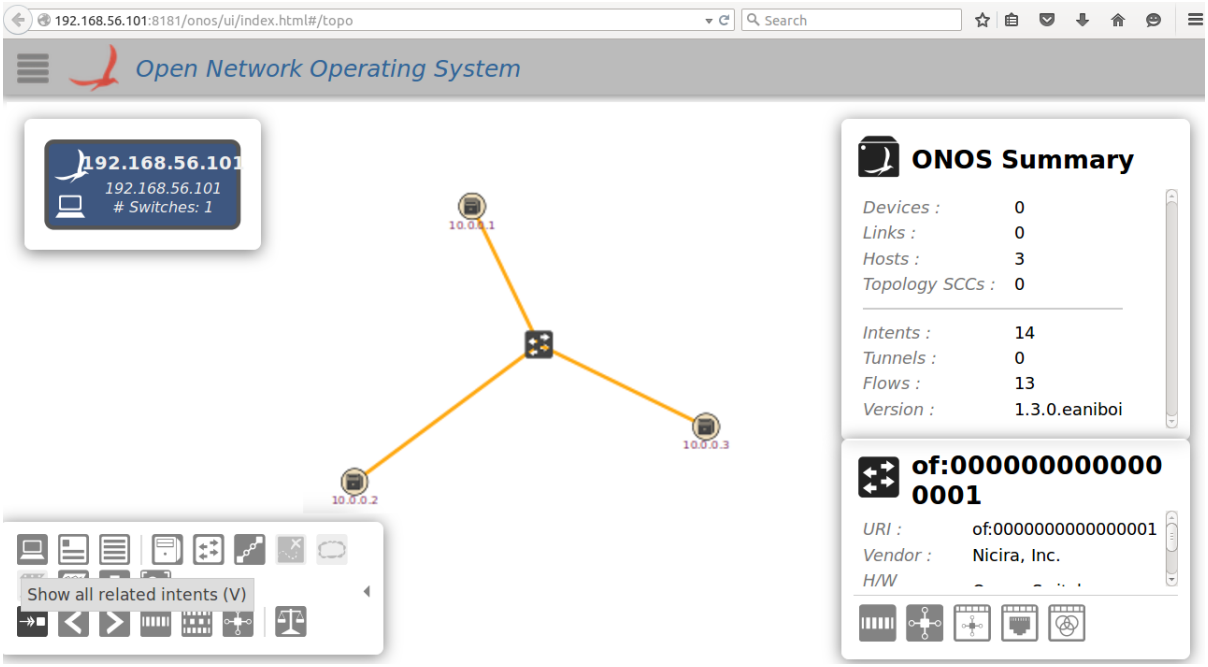

Figure 4.2: Intents verifying in GUI

4.2.2 REST API

To implement port mirroring using REST API, one has to use REST client program. For the purpose of this work we used Google Chrome's extension - Postman. This application has simple interface that satisfies the needs of current work fully. There are two options to configure port mirroring through REST API. First to install Intents and second to

install Flows. To install intents, one has to execute post command providing information in format of JSON string of specific structure. As REST API of ONOS is symmetrical, one can take data format using GET method and modify it to install new intents. In case there are no currently installed intents, one may install it using Intent framework first. The format of JSON string to install intent in order to provide port mirroring in our testing environment is as shown below:


```
{"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=2},
egress=[ConnectPoint{elementId=of:0000000000000001,portNumber=1},
ConnectPoint{elementId=of:0000000000000001, portNumber=3}],
types=[OPTICAL]}]}"}
{"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}],
types=[OPTICAL]}]}"}
```


The address of intent REST API is:
http://192.168.56.101:8181/onos/v1/intents Where 192.168.56.101 - is IP address of virtual machine on which ONOS controller is running, /onos/v1 is common address for all REST interfaces of ONOS. There are no possibility to change intent, instead one has to remove intent using DELETE method and add new intent. To remove intent, its ID

should be specified. To verify installed intents, one can use GET method in the same API.

To install flows one has to use REST API with address: *http://192.168.56.101:8181/onos/v1/flows* JSON string format for flow also can be find using GET method as REST API is symmetrical. To configure port mirroring, one has to install two flows as follows:

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

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

Both of this methods will provide port mirroring for our use case in test environment, but if in first case controller will decide itself how to provide required service, in second case we controller receives explicit instructions to install flows to particular switch. The difference between these approaches was covered in section corresponding section of design chapter of this work.

4.3 OpenDaylight

To install OpenDaylight to the test environment, one has to download pre-build package of Lithium release from official web site of the project. Unix-like operating system has to be used. Then unpack to desired folder and run /OpenDaylight/bin/karaf. The Karaf container will be opened and core applications of the controller will be started in it. To use Yang interface one has to install additional packages using commands:

- feature:install odl-dlux-all
- feature:install odl-restconf-all
- feature:install odl-openflowplugin-all-li
- feature:install odl-yangtools-all

Provided commands install indicated packages as well as packages they are depend on. When these packages are installed, Yang interface is available in DLUX GUI at: *http://localhost:8181/index.html/yangui/index*.

OpenDaylight currently does not support removing of packages. There are also some packages that can not be installed at the same time, for example on of packages of Group Based Policy is not compatible with odl-openflowplugin package. Hence to use OpenDaylight's GBP interface one has to has new installation of the controller. Having new installation, one has to install additional packages using next commands:

- feature:install odl-dlux-all
- feature:install odl-l2switch-switch-hosttracker
- feature:install odl-groupbasedpolicy-base
- feature:install odl-groupbasedpolicy-ui
- feature:install odl-groupbasedpolicy-ofoverlay

When these packages are installed, Yang interface is available in DLUX GUI at: *http://localhost:8181/index.html/gbp/index*. To prevent l2switch from proactive install of flows that will provide any-to-any availability, configuration file for this application should be changed.
In *OpenDaylight/etc/opendaylight/karaf/57-hosttracker.xml* parameter *pro-active switching* should be marked as false.

4.3.1 Yang User Interface

To implement port mirroring using this interface, one has to create two flows, each with two actions in action list and instruction "Apply-Actions". In interface one has to consequentially expand "opendaylight-inventory" item, "config" item, "nodes" item and "node" item. Then one has to choose "table" item and fill the fields of node ID and table ID in action menu with "openflow:1" and "0" respectively. Table with table ID=0 exists by default in any OpenFlow switch, thus if one sends "GET" request, the information about table will be received and shown below. There are no flows in table 0 by default. One has to press "plus" symbol beside "flow list" item to create a new flow in the flow table 0, fill in ID in corresponded field, expand "match" item and write "openflow:1:2" into field " in-port. This will match all traffic incoming in port number 2 of the switch. After that, one has to expand item "instructions" and add new "instruction list" by clicking on "plus" symbol. In new instruction list one has to choose "apply-actions-case" in pull-down menu, expand "apply-action" item and create two action by clicking "plus" symbol beside "action list" item. In actions one has to choose "output-action-case" in pull-down menu, expand "output-action" item and fill-in "output-node-connector" fields with 1 for first action and 3 for second action. One also has to define "max-length" as 65535 and specify order of action in corresponded field as shown in figure 4.3.

Then one has to choose "PUT" method from pull-down menu above and press "send" button. The command results in next JSON string:


```
{"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"}},{"output-action": {"output-node-connector": "2",
"max-length": "65535"},"order": "0"}]}}]},"priority":"1",
"idle-timeout": "1800","hard-timeout": "3600",
```

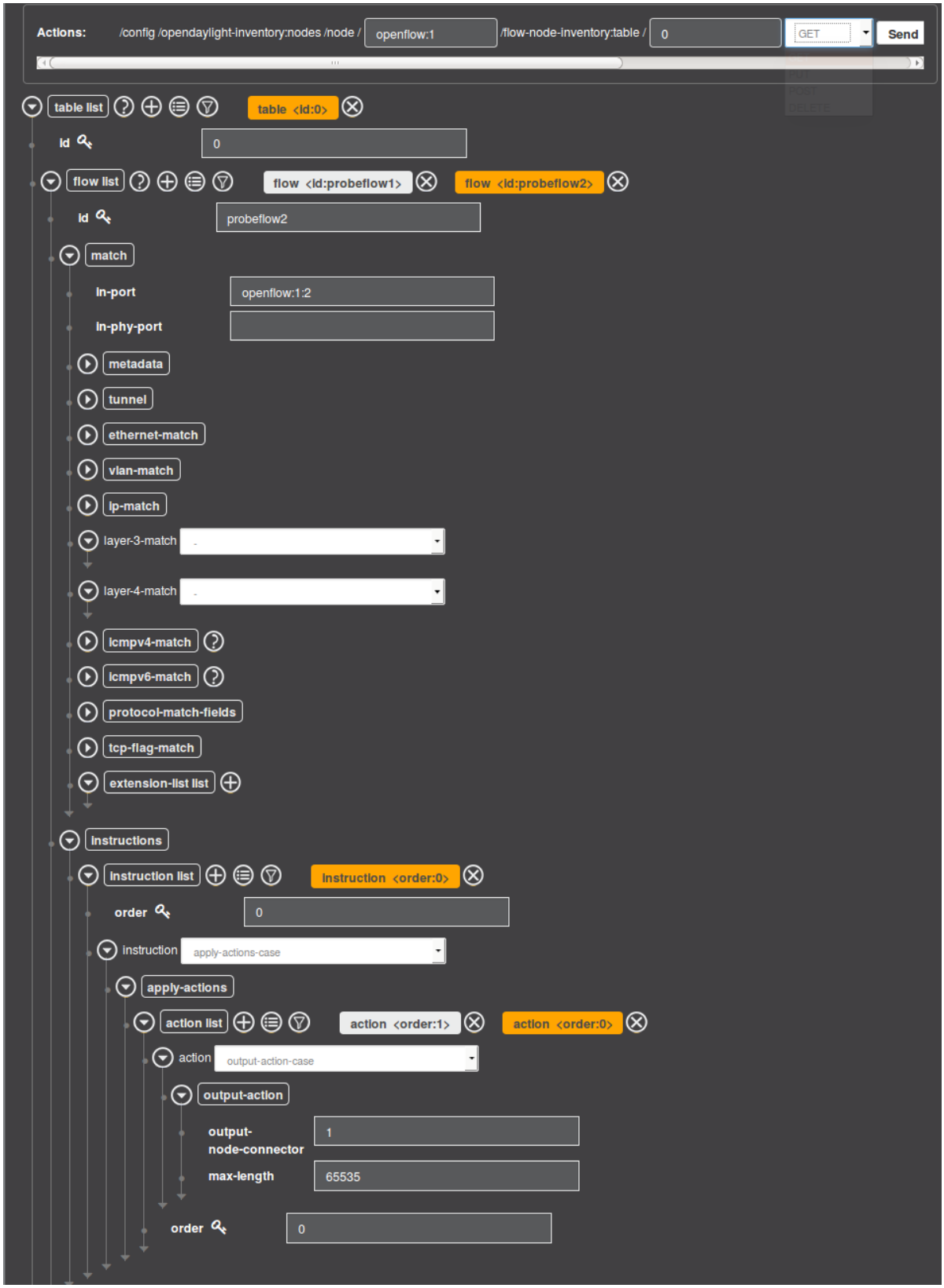

Figure 4.3: Inventory Yang Interface

"table_id":"0" },{"id": "probeflow2","match": {"in-port":
"openflow:1:2"},"instructions"{"instruction": [{"order":
"0","apply-actions":{"action": [{"order": 1,"output-action":
{"max-length":65535,"output-node-connector": "3"}},
{"output-action":{"output-node-connector":"1","max-length":
"65535"},"order":"0"}]}}]},"priority":"10","idle-timeout":
"1800","hard-timeout":"3600","table_id":"0"}]}]}

which is sent to: http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/flow-node-inventory:table/0.
Using REST function "GET" one can verify configuration and using "DELETE" function erase rules from switch.

4.3.2 Group Based Policy

As it was discussed in corresponding section of design chapter, it is currently impossible to configure port mirroring using group based policy interface, due to absence of flooding behaviour. But as we anticipate this functionality to be added soon into interface, we will present hypothetical implementation. Currently it is possible to configure standard forwarding from one group of hosts to another. This functionality is provided by rule that has classifier which matches all IP and ARP traffic and action "allow" or, in other words, "forward". In following we will first define steps that one has to complete to configure normal forwarding, then we will explain required changes towards port mirroring and explain difference between current and required configurations.

To configure forwarding, one has to access GBP interface in DLUX at address: *http://localhost:8181/index.html/gbp/index* Then press "Policy expression" button to enter policies configuration interface and create a tenant. Then one has to create network contexts in "L2/L3" section of "Policy" menu. The sequence of steps is: add L3 context, add L2 bridge context and tie it to created L3 context, add L2 flooding domain and tie

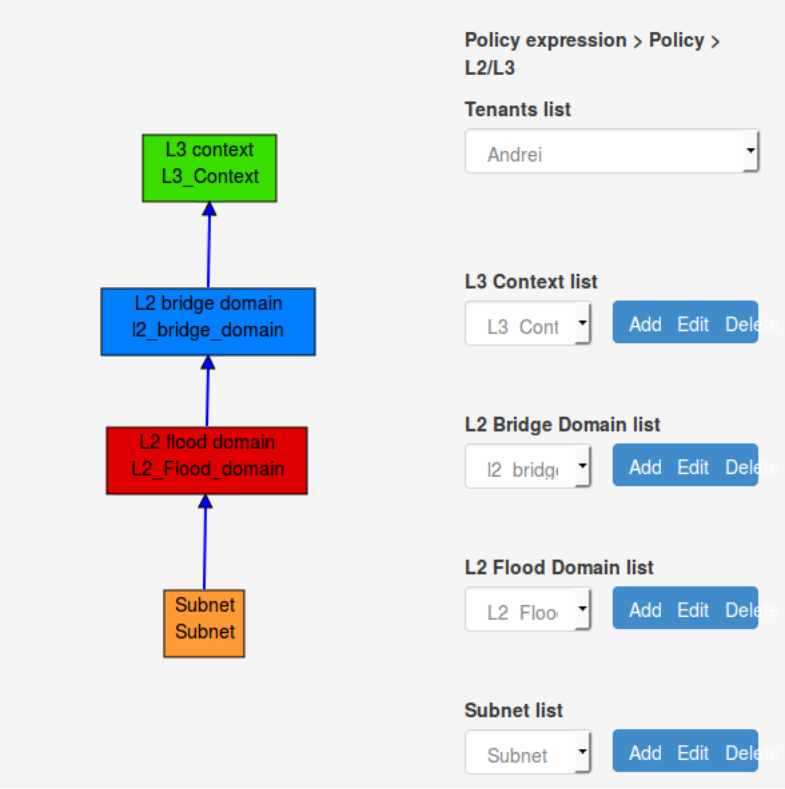

Figure 4.4: Defining network context in GBP

it to L2 bridging domain, finally add subnet with network address 10.0.0.0/24 and tie it to flooding domain as show in figure 4.4.

Then one has to specify endpoint groups and add endpoints from test network to them. In first group one has to add host A of test network, and in second group add hosts B and C.

Creation of policy itself should be started with creation of classifiers and actions in "Renderers" menu. Then one has to go back to "Policy" menu and create a contract in corresponding sub-menu, specifying clause, subject and rules. One has to create two rules: one for matching and forwarding IP traffic, second for ARP traffic. Then one has to go back to endpoint groups and apply the contract, by specifying it in "Provider named selector list" of group 1 and "Consumer named selectors list" of group 2. Thus group 1 will provide service defined in the contract and group 2 will consume it. Figure 4.5 shows delivered policy demonstrated in GBP interface.

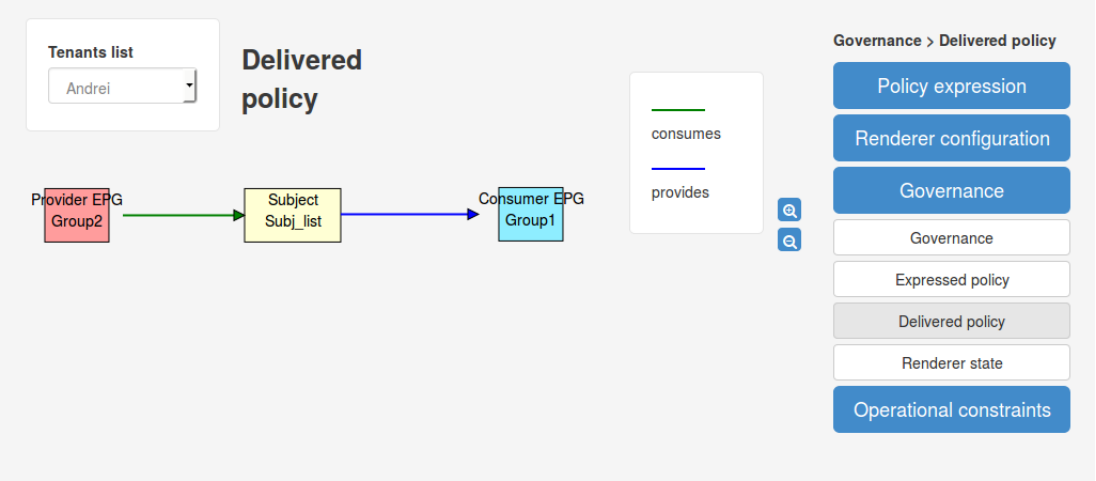

Figure 4.5: Delivered policy in GBP

These steps will provide one with forwarding from group 1 to group 2. To implement traffic mirroring, new action should be added to interface - flooding. We anticipate this action as well as other will be added during the process of interface development. When the new action appear, there is only one change to this configuration should be done to obtain desired functionality. In rules action should be changed from "allow" to "flood".

4.4 Implementation Overview

In this chapter we provided detailed steps of port mirroring configuration using Intent framework and REST API of ONOS controller and Yang interface of OpenDaylight controller. We also analysed hypothetical implementation of port mirroring using Group Based Policy interface, although it requires further development to provide this functionality. All implementations were done in identical test environment and were verified with packet analyser WireShark that analysed traffic at port 3 of test switch that was connected to the "Probe" host.

Chapter 5

Evaluation

In this chapter we will evaluate different aspects of interfaces that we used for implementation of port mirroring. In order to evaluate these interfaces we defined four basic functions that support port mirroring as a service. First of all to configure port mirroring, one has to find out what device IDs and port numbers of ingress, egress and mirroring ports. One also has to be able to see current configuration of devices. For purpose of work we united this functions in one and called it Discovery. We will take a look on how easy to discover topology in each of interfaces as well as how current configuration is represented. Next part of functionality is implementation itself. Here we will discuss how user friendly it is, how clear are steps of configuration and how clear are data structures. For short we will call this part of functionality Setup. Next part of functionality - Change - is responsible for changing of configuration. We will keep its definition very narrow since part of its functionality is covered with previous two functions: Discovery and Setup. Thus we only interested in possibility of changing of configuration on the fly, for example changing number of mirroring port without need to stop the service. The last part of functionality is Removal of configuration for port mirroring. As with Setup, we will discuss how user friendly or intuitive is interface in case of this function.

We will start evaluation with comparison of functions by interfaces, discussing the difference between the same functions in different interfaces. Then we will compare interfaces

by functions and analyse how suitable every interfaces is for the port mirroring, what are its benefits and limitations. While describing interfaces, we will also move in same order: ONOS's Intent Framework, ONOS's REST API, ODL's Yang interface and ODL's Group Based Policy.

5.1 Comparison Of Functions By Interfaces

In table 5.2 we presented short comments on implementation of functions in different interfaces. We will explain them in detail in following section.

Table 5.1: Comparison of functionality

	Intent Framework ONOS	REST API ONOS	Yang Interface ODL	Group Based Policy
Discovery	Simple but not full with GUI, full but not as simple with CLI	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	Simple in CLI, not implemented in GUI	Moderate, data structures not documented	Moderate/easy after previous step, not documented	Not implemented

5.1.1 Discovery

Discovery consist of two main functions: topology view and presentation of current configuration. In fact, topology view is always provided with different component of controller, but we discuss it because some of interfaces are integrated with topology view. For example ONOS Intent Framework graphic user interface is integrated into topology view, one

can identify or change intent configuration by clicking on topology view. Intent Framework's CLI in its turn is united with topology CLI, thus one can use one interface for both, topology view and for intent configuration. REST API of ONOS also provides topology view, although it is not so representative, as has a form of JSON string. It is recommended to use web based GUI of ONOS to discover topology instead of REST API.

ODL's Yang interface and GBP are built-in into DLUX interface that has graphic topology representation by default, although it is not same informative as in ONOS. It is also possible to query topology through Yang interface, but JSON string, that one will receive in result is difficult to read, especially when network is not small.

Configuration representation in Intent Framework is simple with both interfaces, GUI and CLI. The difference is that in GUI one can see path that packets are forwarded through tied to current topology as shown in figure 4.2. Clicking on device one can see all related intents as lighted up connections. This is also a shortcoming of GUI as user can not distinct different intents in case of multiple intents are installed. In its turn CLI has a good representation of intents as shown in figure **??**. User can easily understand intents functionality as well as define ingress and egress ports. REST API provides very similar representation of intents in JSON objects, which also contain some control information, but stays readable.

ODL's Yang interface is providing user with good representation of configuration, supported with Yang data structures. Instead of raw JSON string or XML, user can observe information already parsed and placed to corresponding windows, pull down menus, radio buttons, and comment fields of interface. This automatic parsing significantly levels up quality of representation. The major drawback is a lack of documentation for Yang interface. It might be very difficult for people that not familiar with interface to define address of REST API where GET request had to be sent. The troubleshooting is also difficult as messages that interface send in answer to wrong actions are often inconsistent. Group Based Policy interface has a graphical representation of current configuration, which shows all configured policies in form of topology. This representation is natively understandable

with condition that person familiar with abstractions provided by interface, e.g. endpoint groups and contracts.

5.1.2 Setup

Intent Framework has two ways to install intent. First by allocating two host's icons in GUI and choosing the function "setup intent" from pull down menu of right mouse button. This method allows to setup only host to host intent which is not suitable for port mirroring. Second way is through the CLI. Addition of new intent is very simple due to the presence of context-sensitive help which offers to the user all options of further configuration. This benefit might be not applicable for very big networks, but works well for our use case.

Due to significant lack of documentation, to setup new intent with REST API one has to manually sort out the structure of JSON string that need to be sent. If controller already has similar intents, one can use JSON object received in answer to GET query as a starting point. Then one can use error messages that indicate wrong structure and use them as hints. This investigation-like process of learning is very difficult and barely suitable for configuration of production networks. Since ONOS is fully open source project, another way to find out data structures, to take a look at source code, altough it requires additional skills.

Configuration of intents in Yang interface is also difficult. In contrast to ONOS REST API, Yang interface is visualizing data from JSON string and presents it in form of named text areas, radio buttons, pull down menus, as shown in figure 4.3. To configure port mirroring, one has to install specific OpenFlow rules, or flows, which in detail described in OpenFlow specifications [5, 6, 7]. The problem is that neither in the documentation of Yang interface nor at specification of OpenFlow protocol, user may find information of which fields are compulsory and which are optional. In other hand, once become familiar with interface one can use it for wide choice of configurations.

As we described in design and implementation chapters, Group Based Policy is currently not suitable for port mirroring. Nevertheless it is quite easy to operate in this interface in order to create new rules. As other interfaces, GBP is very new and still under development, thus we anticipate required functionality to be added. When it happens, the configuration of port mirroring will become simple operation. As with other interfaces, GBP suffers from lack of documentation.

5.1.3 Change

ONOS Intent framework, currently does not support change of intents. Instead of it one has to delete old and create new intent. According to our use case, port mirroring while using for probing is not critical function, thus interruption caused by reconfiguration is acceptable.

To configure port mirroring, we used REST API to install both intents and flows. In case with intents it is same situation as with Intent Framework interface, change is not allowed. If we configure port mirroring with flows, it is possible to change flows on the switch by sending new configuration with same table/flow id. The difficulty is same as with setup functionality, data structures not documented.

When using Yang interface, one creates flows to install them to switches. This flows might be easily changed by sending new configuration with same table/flow id. This operation is very easy if one already learned how to use interface to create flows. In other way same difficulty as with setup is applied.

Although GBP can not be used for port mirroring, we expect that by the time that required functionality will be added, change of configuration will be the same as change of configuration for forwarding. Thus we expect it to be simple and straightforward as well as initial configuration.

5.1.4 Removal

Removal of intents in GUI is not allowed, but it is very easy to locate and delete intent using CLI interface. To do this one has only to specify intents id in "remove-intent" command. Another useful feature that after removal intents do not disappear, but change status to "withdrawn" and remain visible in intents database.

Removal of intents in REST API is easier then creation, but still far from obvious. Again the main problem is lack of documentation. Removal of flows is easy if one learned how to install them, cause command has same structure.

Removal of flows with Yang API is very easy as command is symmetrical to discovery. Once user successfully used GET method for required API, he only needs to switch to DELETE method and press one button. However if one has never worked with interface before, it is same problems applied as for discovery.

Group Based Policy interface provides easy methods to delete any part of configuration. To delete forwarding rule, one has just to exclude specific contract from contract list of group or endpoint from group, depends on what configuration one needs in result of removal.

5.2 Comparison Of Interfaces By Functions

Table below is a copy of table from previous section but with rows and columns swapped. In following section we will describe interfaces by functions.

5.2.1 Intent Framework ONOS

As it was discussed above Intent Framework allow to user an easy way to discover, create and remove configuration for port mirroring through two different interfaces. This interface is the only from all reviewed interfaces that has detailed documentation. A few tutorials are also posted for it on official website of the project [23], which makes it easiest

Table 5.2: Comparison of functionality

	Discovery	Setup	Change	Removal
Intent Framework ONOS	Simple but not full with GUI, full but not as simple with CLI	Straightforward with CLI, limited with GUI	Not implemented	Simple in CLI, not implemented in GUI
REST API ONOS	Moderate	Difficult, data structures not documented	Not implemented	Moderate, data structures not documented
Yang Interface ODL	Moderate, not documented	Difficult, not documented	Difficult/ easy after previous step, not documented	Difficult/ easy after previous step, not documented
Group Based Policy	Simple, fully visualized	Not implemented	Not implemented	Not implemented

to learn.

From limitations of discovery functionality of interface we can note the limited functionality implemented in GUI. If one will use functionality "show related intents" to find out intents related to a switch or host, all related intents will be shown together. It is impossible to distinguish intents from each other in this mode, which makes this function uncertain. In other hand usage of show command "intents" in CLI does not provide to user information of current traffic's path, which is not necessary for our use case, but definitely is a useful feature for big networks. From benefits we should notice ease of usage both GUI and CLI to define or verify configuration of intents.

Setup functionality also has its limitation in GUI, one can only setup host-to-host and switch-to-switch intents, but not host-to-multi intents, which is required for port mirroring in our use case. In other hand there is very simple setup procedure provided with CLI interface. It is very intuitive and has very good context-sensitive tab completion, which compensate the need to type in long IDs of network devices. Although benefits of it might be not so obvious in very big networks with hundreds or thousands of devices, it is working perfect for our use case.

This interface also provides very simple way to delete configuration in CLI. To delete

existent intent one has only to specify its ID. Removal is not available in GUI, due to inability to distinguish intents from each other. Thus this functionality might be expected to be added in conjunction with intents separation in GUI.

There is no way to change intents on the fly, but for our case it is not necessary. Based on what was said before, we can surely said that Intent Framework is fully suitable for port mirroring configuration in our use case in terms of functionality.

5.2.2 REST API ONOS

REST API is less documented interface of all that reviewed. The only information we could found was the web addresses of interfaces. No tutorials or any other documentation is officially posted. The interface is easier to use to those who have background in RESTful services.

Discovery of current configuration is the only functionality that is relatively easy to use, as one needs only to send GET request towards specific address of REST API and does not need to provide any data. The data representation is also not user friendly, due to specific of RESTful services, one has to read some times very long JSON strings. In favour of interface we have to notice that user friendly representation of information is not one of its goals.

Due to lack of documentation, addition of new configuration is very difficult. One has to select the structure of message in the POST method manually, while only focusing on errors, that controller is sending in answer to incorrect structure. More reliable but more skill demanding way to find out structure is to go strictly to source code of controller.

As in case with Intent Framework interface, there is no way to change intent on the fly. However if one use REST API to configure port mirroring by installing flows, he might change it by sending changes with the same ID of the flow.

Removal of intents and flows is not as difficult as setup, since in contrast to POST method of API, DELETE method requires symmetric structure of the message as GET.

Thus one can first GET structures from the controller and then delete them by using DELETE method of API for received object.

The REST API can be used to configure port mirroring in our use case, but it is not preferable due to its complexity and lack of documentation. The only case that this interface is preferable if RESTful service is only way to access installed ONOS controller and Intent Framework interfaces are not available.

5.2.3 Yang User Interface ODL

Yang interface is a graphical representation of REST API's of ODL applications, that automatically generated from REST API and Yang data structures defined in those applications. For the purpose of this work we used Yang user interface of OpenDaylight-Inventory application. All applications provide to Yang interface all the data structures they work with as well as all the methods, described by YANG modelling language. Whereby, in contrast to ONOS REST API, Yang UI may visualize required data structure, which makes interface much more user friendly than raw REST API. However there is also lack of documentation for this interface. Despite the fact that there are many tutorials posted in the web, non of them provides deep understanding of how to use interface. Part of them cover only design specifics and explain how to use topology interface, other are outdated as interface was dramatically changed recently. Nevertheless once became familiar with this interface, one may use it for port mirroring as well as another configuration.

Discovery functionality of interface is hampered by lack of documentation. In other hand, as its discussed earlier, graphic representation of JSON objects received from REST API makes interface more user friendly then raw REST API. To define topology one can use either topology Yang interface or DLUX built-in module "Topology", which nonetheless is not as preventative as GUI in ONOS.

Adding new configuration with Yang interface is tricky for not familiar user. As example if user wants to add a flow there are dozens fields that might be filled. It would

be great functionality if interface had distinct fields that are compulsory for flow setup. However when one gets familiar with interface, setup process become routine.

Once again if, one familiar with interface, changing of configuration is easy. To change flow, one can request it by GET, put changes into resulted form and send it it back using POST or PUT methods without going to different windows or menu. The removal of flow is same easy as its changing, one query flow and send back DELETE request.

Yang interface is provides good deal of flexibility and is fully suitable for configuration of port mirroring in our use case. However it needs to be better documented for new users. As Yang interface is one of the priorities of OpenDaylight project, we expect that more background material will be issued.

5.2.4 Group Based Policy ODL

Group Based Policy is looks like interface that developers community put a lot of effort to. Having a very comprehensive model, this interface intents to make a breakthrough in network configuration, by moving it from expression of how to do things to just what things to do. As others, interface is very new and underfulfilled. The lack of functionality does not allow to configure port mirroring at the moment. Nevertheless we anticipate this functionality to be added and then interface will be fully suitable for our use case.

Currently we only can note that interface is fully visualized, which makes it easy to verify configuration and add or change it. The main shortcoming of interface is its comprehensive model, which takes time from new comers to understand it. But same model may become its main benefit as provides very flexible configuration.

Chapter 6

Conclusions And Future Work

This thesis was motivated with problem of adaptation of SDN for use in telecommunication networks. OpenDaylight and ONOS SDN controllers were chosen as most advanced and perspective open source projects. Port mirroring was chosen as the function to be configured using these controllers as compulsory function for the use case that motivated this work.

This thesis presented qualitative analysis of OpenDaylight and ONOS controllers through analysis of their main northbound interfaces: Intent Framework and REST API for ONOS, Yang interface and Group Based Policy interface for OpenDaylight. The analysis was done by evaluation and comparing of configuration steps for port mirroring. For this purpose process was divided on 4 stages: discovery, setup, change and removal.

This chapter discuss summarised results of the work and discuss suitability of reviewed interfaces for a use case.

6.1 Achievements

Software-Defined Networking is relatively new, but very fast growing area of data networks. Many different approaches are appearing and developing while industry moves towards a standardization. Most advanced of them present themselves as ready for use

solutions and publish a lot of use cases. Nevertheless they stay poorly documented and possibility of their implementation remains uncertain. In this work we checked chosen SDN controllers and their interfaces in particular on possibility of configuration of port mirroring. We chose port mirroring as it is one of mandatory functions for telecommunication networks that suffers from significant lack of attention from developers of SDN controllers. Main results of this work are elaborated testing methodology that could be used to test other controllers as well as other interfaces of controllers tested in this work and specific result that gives conclusion on possibility of using tested interfaces in telecommunication networks.

We find out that Intent Framework interface of ONOS is well documented and meets all requirements to be used for port mirroring. It has user friendly intuitive interface and there are several tutorials for it published on official website that will help to newcomers. It still has minor shortcomings that we discussed in corresponding section of evaluation chapter. It will be safe to say that interface is simplest to use from those been reviewed. REST API of ONOS in opposite to first interface is almost not documented at all and definitely does not suit for a newcomers. Still it provides all required functionality to support port mirroring.

Yang interface of OpenDaylight also provides full functionality for port mirroring, but also provides relatively simple interface comparing to REST API of ONOS. It is also more flexible in configuration than other reviewed interfaces and allows to change configuration on the fly. Developers supported interface with documentation and tutorials, that is not as good as in case with Intent Framework, but still helps a lot and makes interface possible to use for port mirroring.

Group Based Policy interface of OpenDaylight is the most unusual interface of reviewed as it offers fancy model for configuration that provides users with abstractions of network parts and functions. In this meaning it will be safe to call this interface the most SDN-like interface. However interface is currently on early stage of development, it remains immature and does not suit for usage in telecommunication networks as can not be used

for port mirroring.

Besides of answers on the question if particular interfaces are suitable to use for port mirroring, this work contributes elaborated methodology to compare northbound interfaces. It also consists practical step by step configuration guide for reviewed interfaces, which is hopefully will be find useful in the absence of detailed documentation for some of them.

6.2 Future Work

As it was mentioned before, this work proposed a testing methodology and proved environment. Thus results of these work might be used to help evaluate other controllers as well as new interfaces of ONOS and OpenDaylight.
Same is applicable to functions other than port mirroring, that will be easier to test on reviewed interfaces using results of this paper.

There are two possible ways to configure port mirroring with OpenFlow: by using group list that will flood packet towards ports in this group and by using "Apply Actions" instruction that allows to send copy of packet towards the egress port before finishing pipeline process. This leads to obvious question which way provides highest throughput on same hardware/software.

This work also showed significant diversity of northbound interfaces, while if we want to treat controller as operating system for networks, that have to be standard interfaces through applications interact with it. It is clear that SDN area in general is yet very unstable due to its novelty. Thus we anticipated future work towards standardization of northbound interfaces.

Appendix A

Abbreviations

Short Term	Expanded Term
ARP	Address Resolution Protocol
ATM	Asynchronous Transfer Mode
E-UTRAN	Evolved UMTS Terrestrial Radio Access Network
GPRS	General Packet Radio Service
DCAN	Devolved Control of ATM Network
IP	Internet Protocol
JSON	JavaScript Object Notation
LTE	Long Term Evolution
MAC	Media Access Control
MPLS	Multi Protocol Label Switching
NETCONF	Network configuration protocol
ODL	Open Daylight Controller
ODLI	OpenDaylight inventory application
ONF	Open Networking Foundation

Short Term	Expanded Term
ONOS	Open Network Operating System
OVSDB	Open Virtual Switch Data Base
PDN-GW	Packet Data Network Gateway
REST	Representational State Transfer
SDN	Software-Defined Network
S-GW	Serving Gateway
SNMP	Simple Network Management Protocol
TCP	Transport Control Protocol
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunication System
VLAN	Virtual Local Area Network
VNF	Virtual Network Function
WWW	World Wide Web

Appendix B

Test Environment

For the purpose of this work we have been using next hardware and software:

- Computer HP ElitBook, Intel Core I5 CPU 2.00GHz x 4, 8 Gb RAM, 200 Gb SSD
- Operating System Linux Ubuntu Desktop 14.04 LTS
- Operating System Linux Ubuntu Server 14.04 LTS
- Virtual machine emulator Virtual Box v4.3
- Virtual network emulator Mininet 2.1.0
- Java Runtime Environment 1.8 for ONOS
- Java Runtime Environment 1.7 for ODL
- Apache Maven 3.3.1
- Apache Karaf 3.0.3
- ONOS Cardinal Release 1.2.1
- OpenDaylight Lithium Release karaf distribution-0.3.0
- Packet analyser Wireshark 1.10.6
- Google Chrome Postman extension

Bibliography

[1] Dewar Calum. *GSMA Intelligence report: The global MVNO landscape, 2012 14*. 2014.

[2] Andrew T Campbell, Irene Katzela, Kazuho Miki, and John Vicente. Open signaling for atm, internet and mobile networks (opensig'98). *ACM SIGCOMM Computer Communication Review*, 29(1):97–108, 1999.

[3] Martin Casado, Nate Foster, and Arjun Guha. Abstractions for software-defined networks. *Commun. ACM*, 57(10):8695, Sep 2014.

[4] Shanzhi Chen, Jian Zhao, Ming Ai, Dake Liu, and Ying Peng. Virtual rats and a flexible and tailored radio access network evolving to 5g. *Communications Magazine, IEEE*, 53(6):52–58, 2015.

[5] ONF Technical cometee. *OpenFlow Switch Specification Version 1.0.0 (Wire Protocol 0x01) December*, 2009.

[6] ONF Technical cometee. *OpenFlow Switch Specification Version 1.3.0 (Wire Protocol 0x04) June*, 2012.

[7] ONF Technical cometee. *OpenFlow Switch Specification Version 1.5.0 (Protocol version 0x06)*, 2014.

[8] Marschke Doug, Doyle Jeff, and Moyer Pete. *Software-Defined Networking: Anatomy of Openflow*. Lulu publishing, 2015.

[9] Rob Enns. Rfc4741: Netconf configuration protocol. *Std. Track, http://www. ietf. org/rfc/rfc4741. txt*, 2006.

[10] Albert Greenberg, Gisli Hjalmtysson, David A Maltz, Andy Myers, Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, and Hui Zhang. A clean slate 4d approach to network control and management. *ACM SIGCOMM Computer Communication Review*, 35(5):41–54, 2005.

[11] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. Nox: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.

[12] Soheil Hassas Yeganeh and Yashar Ganjali. Kandoo: a framework for efficient and scalable offloading of control applications. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 19–24. ACM, 2012.

[13] Juha Heinanen. Rfc1483: Multiprotocol encapsulation over atm adaptation layer 5. *Std. Track, https://tools.ietf.org/html/rfc1483. txt*, 1993.

[14] Brandon Heller, Srinivasan Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. Elastictree: Saving energy in data center networks. In *NSDI*, volume 10, pages 249–264, 2010.

[15] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. B4: Experience with a globally-deployed software defined wan. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 3–14. ACM, 2013.

[16] Hyojoon Kim, Jose Ronaldo Santos, Y Turner, M Schlansker, Jean Tourrilhes, and Nick Feamster. Coronet: Fault tolerance for software defined networks. In *Network Protocols (ICNP), 2012 20th IEEE International Conference on*, pages 1–2. IEEE, 2012.

[17] Chengchao Liang and F Richard Yu. Wireless virtualization for next generation mobile cellular networks. *Wireless Communications, IEEE*, 22(1):61–69, 2015.

[18] Linux Foundation. *OpenDaylight User Guide*.

[19] Rajesh Narayanan, Saikrishna Kotha, Geng Lin, Aimal Khan, Sajjad Rizvi, Wajeeha Javed, Hassan Khan, and Syed Ali Khayam. Macroflows and microflows: Enabling rapid network innovation through a split sdn data plane. In *Software Defined Networking (EWSDN), 2012 European Workshop on*, pages 79–84. IEEE, 2012.

[20] Bruno Nunes, Manoel Mendonca, Xuan-Nam Nguyen, Katia Obraczka, Thierry Turletti, et al. A survey of software-defined networking: Past, present, and future of programmable networks. *Communications Surveys & Tutorials, IEEE*, 16(3):1617–1634, 2014.

[21] University of Cambridge. Dcan, devolved control of atm networks. `http://www.cl.cam.ac.uk/research/srg/netos/old-projects/dcan`, 2015. [Online; accessed 22-August-2015].

[22] ONF. Open networking foundation. `http://opennetworking.org`, 2015. [Online; accessed 08-August-2015].

[23] Onosproject.org. Open network operating system. `\textbf{http://opendaylight.org}`, 2015. [Online; accessed 08-August-2015].

[24] Opendaylight.org. The OpenDaylight Platform — OpenDaylight. `http://opendaylight.org`, 2015. [Online; accessed 19-August-2015].

[25] Jean-Gabriel Remy and Charlotte Letamendia. Lte standards and architecture. *LTE Standards*, pages 1–112.

[26] Jennifer Rexford, Albert Greenberg, Gisli Hjalmtysson, David A Maltz, Andy Myers, Geoffrey Xie, Jibin Zhan, and Hui Zhang. Network-wide decision making: Toward a wafer-thin control plane. In *Proc. HotNets*, pages 59–64, 2004.

[27] Andrew S Tanenbaum and David J Wetherall. *Computer Networks - 5th Ed.* Pearson, 2011.

[28] David L Tennenhouse, Jonathan M Smith, W David Sincoskie, David J Wetherall, and Gary J Minden. A survey of active network research. *Communications Magazine, IEEE*, 35(1):80–86, 1997.

[29] David L Tennenhouse and David J Wetherall. Towards an active network architecture. In *DARPA Active NEtworks Conference and Exposition, 2002. Proceedings*, pages 2–15. IEEE, 2002.

[30] Wenfeng Xia, Yonggang Wen, Chuan Heng Foh, Dusit Niyato, and Haiyong Xie. A survey on software-defined networking. *Communications Surveys & Tutorials, IEEE*, 17(1):27–51, 2014.