

Ivan Kupalov

# Software Defined Networks Case Study

Bachelor's Thesis

Data Networks Engineering

February 2016



**KYAMK**  
University of Applied Sciences



# KYAMK

University of Applied Sciences

Author (authors) Ivan Kupalov	Degree Bachelor of Data Networks Engineering	Time February 2016
Thesis Title Software Defined Networks Case Study		27 pages 22 pages of appendices
Commissioned by Kymenlaakso University of Applied Sciences		
Supervisor Vesa Kankare, Senior Lecturer		
<p><b>Abstract</b></p> <p>This thesis describes a process of creating a virtual environment and studying materials which should help students to learn the basics of Software Defined Networks.</p> <p>Software Defined Networks is a new approach to building networks which moves part responsible for making routing decisions away from the network device to the centralised controller. SDN are especially useful for cloud and datacenter environment and play important role in creating Internet of Things.</p> <p>The learning environment was built using virtualisation technologies. Virtual machine containing SDN controller and simulated network was used. Network topologies were developed to be used with lessons and challenges. Network simulation was done using lightweight Mininet software. In order to choose the most suitable technologies research was made and comparison of SDN controllers was conducted.</p> <p>The studying materials include guides and challenges for students. An attempt to incorporate theory explanations into the step by step guide was made to keep learning entertaining and easy.</p> <p>As a result, complete learning environment was made which can be used as a part of a networking course or as an introduction into specialised SDN course.</p>		
<b>Keywords</b> networking, SDN, OpenFlow, Mininet		

## CONTENTS

1 INTRODUCTION .....	4
2 PROJECT GOALS.....	5
2.1 Research on technology.....	5
2.2 Evaluating .....	6
2.3 Virtual environment for learning.....	6
2.4 Instructions and challenges .....	6
3 THEORETICAL OVERVIEW OF SOFTWARE DEFINED NETWORK .....	7
3.1 Traditional approach to networking and origins of SDN .....	7
3.2 Theoretical overview of SDN .....	8
3.3 SDN Controller .....	9
3.4 OpenFlow .....	11
4 VIRTUAL LABORATORY ENVIRONMENT .....	15
4.1 Overview .....	15
4.2 Choices .....	16
4.3 Similar projects.....	18
4.4 Implementation.....	18
5 STUDYING MATERIALS .....	20
5.1 Overview.....	20
5.2 Implementation .....	20
5.3 Feedback.....	22
6 CONCLUSIONS.....	23
6.1 Results.....	23
6.2 Challenges experienced .....	24
6.3 Ideas for future use.....	24
6.4 SDN technologies maturity .....	25
LIST OF REFERENCES.....	26

## APPENDICES

Appendix 1. Lab Guide

## 1 INTRODUCTION

Due to the progress in computing many new products and services have emerged. Every day a new application launches. Limits are being pushed and common concepts of possibility are being changed (Vaughn, 2012).

New type of services requires new level of flexibility and dynamical configuration. Devices have to be configurable on-the-fly. Services like Amazon AWS or Google Cloud Platform require automatic scaling and shaping of the network.

In the world of such high requirements it would be impossible to keep everything static like it used to be. Better solutions were developed.

Virtualization is one of the major trends in ICT that helps to satisfy requirements of the new world, especially ones of the cloud computing. Virtualization of storage, hardware, operating systems and networks improved uptime and flexibility. In all areas of IT there were special hardware and vendor differences and in most cases they have been replaced with general-purpose software, inter-vendor compatibility and virtualization.

The same process started to happen in networking. Standards in networking are already in use but we are still far from the point where it is possible to buy devices from any vendor and configure them in the same way or install different operating system. Software defined networking follows virtualization trend. Although, it is more than just virtualized network.

Another reason why software defined networks were designed is security concerns. Cloud is a very attractive target for attackers because usually it contains large amounts of sensitive data. Attacking a cloud may also help hackers to infect programs which users download and get access to even more resources and information.

Distributed Denial of Service attacks (DDoS) became more powerful and frequent. "High-bandwidth, volumetric infrastructure layer (L3 & L4) attacks increased approximately 30 percent" (Krishnan, Durrani and Phaal, n.d., p. 1). DDoS Mitigation market is expected to grow by 18% 25% till 2017 according to DDoS-IDC or by 25% according to DDoS-Infonetics.

Software defined network (SDN) gives real-time analytics and DDOS mitigation capabilities. Client of ISP cannot always mitigate attack himself and requires upstream help. If ISP can provide such a service it can be a new source of revenue. The same applies to cloud data centers. Cloud is usually expected to protect their customers from the DDOS attacks and can use mitigation as a source of revenue.

One of the upcoming trends is Internet of Things (IoT). Internet of Things is a network of embedded devices which exchange data between each other. It is a common thought that IoT is a sign of the new revolution and it is a new market for products and services. (Burrus, n.d.) IoT can provide massive amounts of relevant data and potentially can change many areas of life. Possible applications include warehousing, logistics, city planning, data center infrastructure, energy use, "smart homes". What IoT means for networking is changed nature of traffic and increased complexity. Time has proved that one of the most effective instruments to fight complexity is software. SDN may be one of the ways to fight emerging challenges, it may help us to operate networks in smarter ways.

Teaching future technologies is a task of a great importance. Today's students are tomorrow's workers. Some of them will be responsible for making important decisions and knowledge of more sophisticated methods may help to choose the most appropriate network design.

This paper describes process of creating learning environment and learning materials for the lesson explaining basics of software defined networks and use of OpenDayLight controller.

## 2 PROJECT GOALS

### 2.1 Research on technology

To be able to teach others and evaluate new technologies deep and extensive knowledge is required. One of the ways to obtain it is to study theory, use cases and make attempts to use technology. Doing that is vital for

accomplishing other goals. Making notes in the process of learning may be helpful for making studying materials.

## 2.2 Evaluating

The goals include making an attempt to evaluate new network technologies and trying to check if they are ready for production and teaching or not. Maturity of a technology means ease and cost of deploying and integration, reliability and, the most important, that it has benefits over older technologies. Suitable scenarios and environments for applying should be proposed.

## 2.3 Virtual environment for learning

One of the biggest obstacles for studying new technologies is getting and trying out these technologies. One of the tasks of this thesis is eliminating this barrier by making a pre-built virtual environment including both parts of SDN: Virtual Network and SDN controller. Virtual environment has very strong advantages over physical one including personal work and lower expenses. With little effort labs can be used remotely. Potentially, gamification can be applied.

There are some requirements for a virtual network. Devices in this network should support specific SDN-related technologies and standards. Hardware capabilities of workstations used in the classroom should also be taken into consideration. Giving some flexibility for virtual environment may be future-aware solution.

## 2.4 Instructions and challenges

Almost every learning process can be more effective when students are given proper hints or instructions. Creating detailed and easy to understand lab materials is one of the most important goals. Lab materials should include step-by-step guides, basic theory and explanations how given examples are related to real-life cases. Students are not expected to have deep knowledge of software defined networking, when they start using materials with an exception of some theoretical knowledge and basic networking knowledge.

### 3 THEORETICAL OVERVIEW OF SOFTWARE DEFINED NETWORK

#### 3.1 Traditional approach to networking and origins of SDN

Until the last few years all the networking resources, storage, computing and network, were intentionally separated as well as systems used to manage these resources. Data centers servers were focused on doing a specific task such as mail servers or database servers.

Situation started to change with wide adoption of virtualization. Servers which were running on a “bare metal” before became universal computing and storage systems. The age of *elastic computing* has begun. (Thomas D. Nadeau, Ken Gray, 2013, p.2). It led to massive move of computing power to datacenters.

Datacenter operators, which now were able to move entire operating systems between machines, started to optimize utilization of machines based on metric such as cooling and power. By packing all the operating machines together operator was able to idle or power off big parts of data center. In the same way operator could move resources by geographical demand.

Soon, the biggest players in IT field, especially the ones who experienced exponential growth, such as Amazon, who were forced to increase their computing resources preliminary, realized that they can increase utilization of their computing power by selling it to customers and getting some of their capital investments back. (Ma, 2014) This way, services like Amazon Web Services or Google Compute Engine were invented. Thus, data centers became multitenant. It created a new problem of separating big numbers of potential tenants whose resources were spread across different virtual machines.

Before, devices were connected using relatively simple flat networks which were separated using layer 2 tools or layer 3 VPNs. Address spaces usually were assigned from one address block. In contrast, in multitenant networks each virtual machine should be assigned IP address and every VM has its own unique hardware address. Moving of a virtual machine often means not only address relocation but also changes in layer 3 routing.

While data centers were evolving, network equipment did not change much, apart from improved fabric capabilities and speed since IP and MPLS (multiprotocol label switching) technologies. These technologies allowed network operators to manipulate networks in the way data center operators manipulated virtual machines. Networking devices usually come with multiple ways of management such as command line, XML/NETCONF or Simple Network Management Protocol.

Traditional approach to building networks assumes that network nodes are autonomous devices which are responsible for making decisions themselves. Issues of this approach started to become more obvious when devices started to become more complex. Some networking devices (such as Nexus series developed by Cisco) became so powerful that they can run virtual machines. The more complex the device is the more configuration it needs. Traditional IP networks may be hard to manage. (Kreutz et al. 14-76, p. 1) It means that more knowledge is required from the staff responsible for managing the network. Human factor is one of the biggest drawbacks of the traditional networks. First of all, salaries is one of the biggest expenses in many cases. Secondly, people have limited capabilities in terms of working with abstractions. In every engineering system level of complexity inevitably increases. Software can give "Bird's-eye view" of the network including structure and statistics and also can alter the network automatically, in the ways people cannot alter it. Networks no longer have to be manipulated by programmers and not by special people. (Ward, 2013)

### 3.2 Theoretical overview of SDN

**"Software-defined networks (SDN):** an architectural approach that optimizes and simplifies network operations by more closely binding the interaction (i.e., provisioning, messaging, and alarming) among applications and network services and devices, whether they be real or virtualized." (Thomas D. Nadeau, Kenn Gray, 2013). Usually SDN Controller provides this binding and orchestrating functions. One of the key differences of SDN approach is



flexibility of configuration, not point of applying it. Two approaches exist for managing SDN networks: *proactive* and *reactive*.

Reactive approach implies that when switch receives unknown packet it sends it to the controller with additional information. Controller should program switch to handle such packets automatically afterwards.

In proactive mode switch is programmed by a controller ahead of time according to its knowledge of the network.

Reactive approach makes network suitable for highly dynamic environments, however, it introduces significant overhead and scalability problems. (Moshref, Bhargava, Gupta, Yu, Govindan, 2014, p. 1)

One of the solutions which may help to address these issues is making SDN network hybrid. Network can be configured in a way when most of the traffic flows are predefined while giving network architect granular traffic control when they need it. Hybrid approach can give both low-latency forwarding of proactive approach and flexibility of reactive one. (Salisbury, 2013)

### 3.3 SDN Controller

SDN Controller's interfaces can be divided into two groups: Northbound and Southbound.

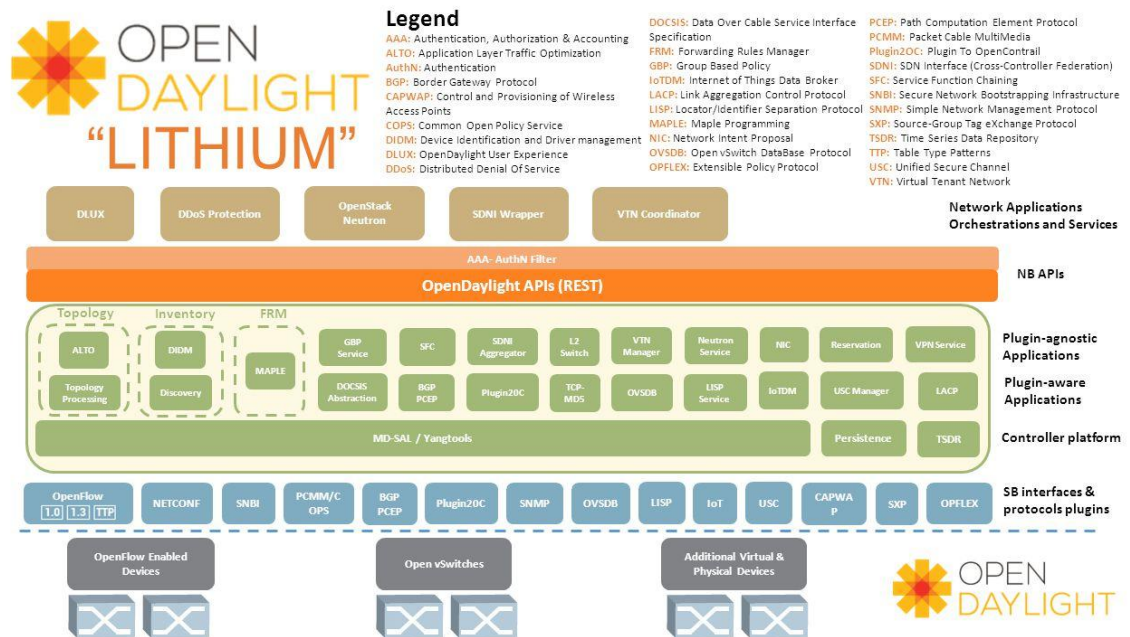


Figure 1. OpenDayLight Lithium Controller architecture (OpenDayLight.org, n.d.)

Simplified architecture of typical SDN controlled is demonstrated on Figure 1. Northbound APIs are facing applications. Controller can receive intentions or commands from the applications and alter network in the appropriate way or return information about the network. Also, applications can be notified by the controller about certain types of events in the network. For example, DDOS protection software can be notified about unusual activity and send signal for change in the network topology to mitigate the attack.

Southbound APIs are facing network. Protocols such as OpenFlow, BGP, NETCONF, LISP, PCEP or SNMP can be used by southbound APIs. With this protocols controller connects to devices to calculate information and configure devices.

Controller platform is the core of the controller. Usually it contains models of the objects in the network and plug-ins. Model is often changed by northbound APIs and these changes can have effect on network.

In engineering popular term “single point of failure” stands for the part of the system on which reliability of the whole system relies. In other words, if this part stops working the whole system stops working as well. It is correct to say

that in some configurations SDN controller is the single point of failure. To solve this issue techniques used for other types of servers may be used because in fact, controller is a server. One of the popular techniques is clustering. Most production-ready SDN controllers support clustering (such as OpenDayLight and ONOS). Clustering is an advanced topic and will not be discussed any further.

### 3.4 OpenFlow

OpenFlow is an open standard being developed by Open Networking Foundation. It is the first standard interface for communications between control and forwarding layers of SDN. The purpose of OpenFlow is to give controller direct access to forwarding planes of network equipment.

Despite the fact that SDN is not directly connected to the OpenFlow protocol, it is *de facto* standard for communications between controller and devices. OpenDayLight controller website states that “while OpenFlow is a useful protocol in many scenarios, SDN is not limited to OpenFlow or any single protocol” (Opendaylight.org, 2016).

There are few technologies that can perform similar capabilities such as I2RS, VxLAN and PCEP. It was considered that this technologies should be out of the scope of the lessons although these technologies as well as BGP and other southbound protocols.

OpenFlow allows move of the control plane from the device to the controller. When an OpenFlow switch receive a packet for which it has no matching flow entries, packet is being sent to the controller. After decision is made how to handle the packet and information is being gathered controller can either drop the packet or install flow entry to the switch.

OpenFlow switch is a device which supports communications with controller using OpenFlow protocol and implements a flow table. OpenFlow was designed to support experiments with existing networks, therefore, it can be enabled only on certain ports of the device. This way device can support both OpenFlow forwarding as well as traditional forwarding at the same time. Researchers can test new routing protocols on a “slice” of real network. Slice

is a part of a network filtered by a common characteristic, for example, communications from wireless hotspots only.

Flows on the devices are organized into tables. Each row in a table is a flow installed on the device. Flows have match fields and action fields. Match fields include Ethernet type, MAC source and destination addresses, IP addresses, ports.

Each flow on the device is associated with a set of actions which should be applied to all matching packets. Instructions can modify the packets state, forward the packet to a port or pass the packet to another table or group. Additional metadata can be added to the packet with passing. Each packet has its own action set and actions from this set will be applied if no further processing is possible. The write and clear instructions provide ways of manipulating the action set. The apply instruction makes the device apply action immediately. The goto instruction passes the packet to another flow table. The meter instruction applies a rate limiter and experimenter instruction provides a way to develop custom extensions and action lists. Figure 2 shows the list of OpenFlow instructions.

<b>Instruction</b>	<b>Description</b>
Apply	Apply action list immediately, bypass action set
Clear	Clear the action set
Write Action	Add actions to action set
Goto	Pass processing to another table
Meter	Apply a rate limiter.
Experimenter	Instruction extensions

*Figure 2.* List of OpenFlow instructions

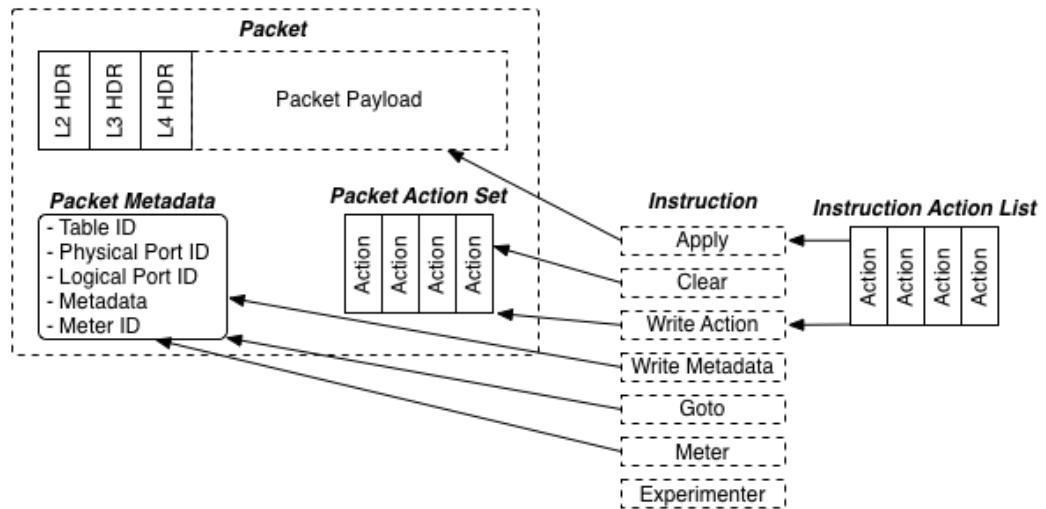


Figure 3. OpenFlow instruction processing. (Flowgrammable.org, 2016)

OpenFlow can manipulate packet in different ways. Device can drop packet, output it to one or more ports, set and push VLAN and MPLS tags, set ARP, IPv4, IPv6, ICMP fields and TCP, UDP and SCTP ports. Figure 3 shows relations between packet payload, instruction and action list in simplified form.

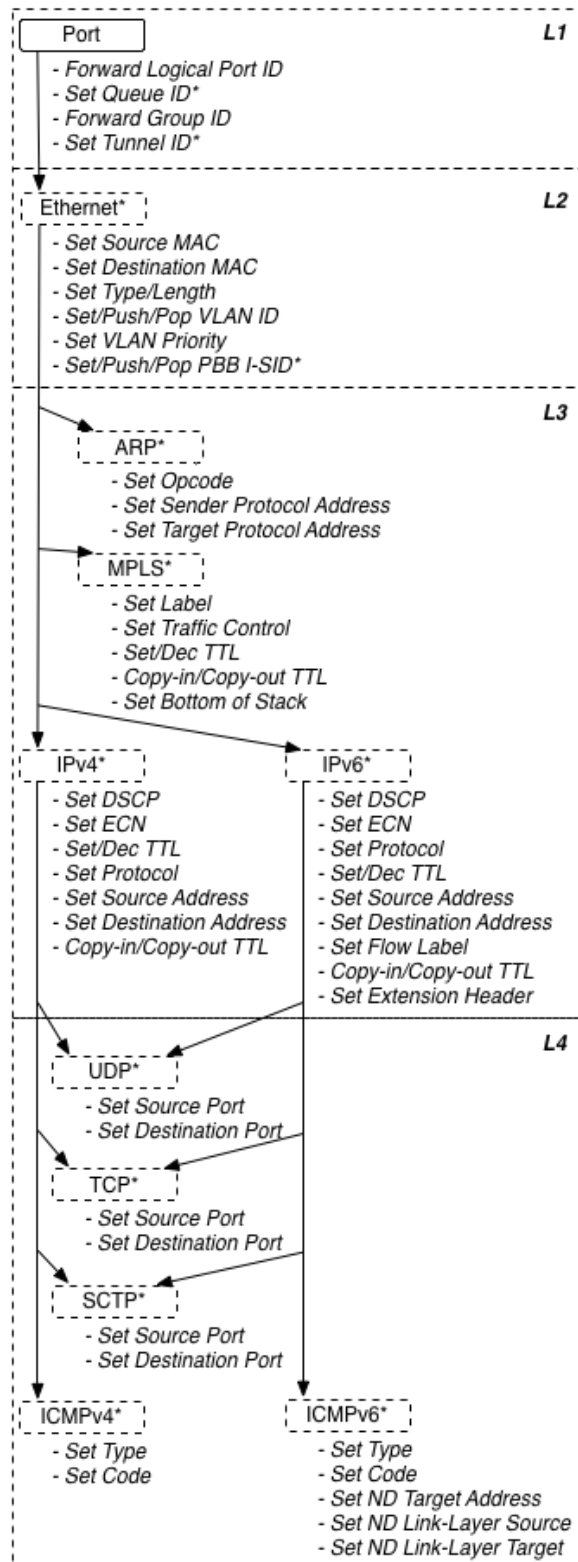


Figure 4. Action dependencies. (Flowgrammable.org, 2016)

OpenFlow standard defines in which order actions should be applied in terms of dependencies as shown on the figure 4. L4 fields will be set first and L1 operations will be done last.

When an OpenFlow device establishes a connection to a controller, it indicates which of the features it supports.

## 4 VIRTUAL LABORATORY ENVIRONMENT

### 4.1 Overview

Virtual environment in case of teaching SDN technologies has several benefits. First, it is much cheaper than using real hardware with support of SDN. Because SDN adoption is still quite low and SDN is mostly used in big networks, hardware which supports SDN is expensive.

Second, hardware from different vendors has its own restrictions and implementation details. Using virtual software switches gives possibility to easily create and use network without burden of tuning SDN settings in real hardware.

Third, virtual environment is extremely flexible. Making a new topology is as simple as writing a new script. Platforms like Mininet are making providing all needed infrastructure to simulate network.

Few requirements have to be met by environment to make it usable for studying:

- Ease and speed of deployment
- Predictable outcome
- Ease of use

Thus, to meet these requirements, a decision was made to make a preconfigured virtual machine. This virtual machine should include:

- Network simulation tools
- SDN controller
- Web browser
- A tool for making HTTP requests
- Network sniffer and protocol analyzer

Users should be able to make HTTP requests because HTTP API is one of the easiest northbound controller's interfaces. While it is not as simple to use as web-based graphical interface it is much more reliable and stable.

Network sniffer should give students ability to investigate the network from the different point of view and prove correctness of their guesses. It is used by the tutorial to demonstrate connection between switches and the controller. Sniffer is even more useful as the lab contains challenges.

## 4.2 Choices

There are many network simulation tools available. During the research three major option were discovered: Cisco Virtual Internet Routing Lab (VIRL), Mininet and virtualizing devices using images.

VIRL is a proprietary Cisco software which includes modified images of real networking equipment operating systems. It has many features such as defining link parameters and graphical topology editor.

As the result of experiments with VIRL it was concluded that it is unstable and requires significant memory resources (four CPU cores, CPU virtualization extensions, 8 GB of DRAM for VM, 70 GB space of disk). More, VIRL is distributed using subscription model. One more disadvantage is a difficulty of linking SDN controller to the devices. It can be done in two ways. First method is putting SDN controller VM inside VIRL. It means deeper virtualization level and can cause performance issues. Second method is making connections from devices to the outside of VIRL to the separate VM which means that additional effort should be put into making topologies as well as resource overhead of running two virtual machines. This method brings unnecessary complexity

Experience of working with virtualized device images was quite good and allowed almost all possible scenarios. Setbacks of this method are still relatively high memory consumption and slow start. Every device requires additional configuration in order to work with OpenFlow. All difficulties of connecting SDN controller still take place while using this approach.



Mininet is an open-source flexible software which was created with support of OpenFlow in mind. It is *de facto* standard for teaching and making experiments with OpenFlow and SDN. Mininet supports different types of underlying controller technologies. For the purpose of this lab open vSwitch was chosen as open source, popular, reliable and efficient solution.

After consideration, Mininet was chosen as a network simulation software. It has few advantages such as fast startup time and low resource requirements. Device, topologies and links behaviour can be easily changed by scripting. More, mininet interfaces can be easily listened by network sniffer.

Setback of Mininet are defined by the fact that it is switch-oriented software thus Layer 3 features such as MPLS or routing protocols should be implemented on the controller. Since the labs will be OpenFlow focused features of the SDN controllers such as BGP-speaker, VPN tunneling or MPLS configuration are out of the main focus of the lab and their lack is not significant.

SDN controller for the lab should satisfy few requirements:

- It should be free, ideally open-source
- It should be ready-to-use without extensive knowledge of programming
- It should have enough documentation and studying materials available
- It should be stable

OpenDayLight Beryllium was chosen as the SDN controller for the lab as one of the most popular options which means more materials and tutorials are available. It satisfies all the requirements but had some problems with stability as it was discovered later.

As for other controllers, ONOS controller was not stable enough during testing and crashed after launch. Floodlight controller and Beacom controller had scarce documentation and did not provide enough studying materials. NOX/POX, Ryu and Cherry controllers required knowledge of programming to operate them which conflicts with requirements of the lab.

Chromium browser and Postman add-on were chosen as web browser and http request tool. Postman has a feature that allows sets of requests to be stored and reused. It can be used for making sets of requests to SDN controller which students can use and customise. It is important since requests are made through REST API and have long URLs and hard to remember request bodies.

Special build of Wireshark with support for OpenFlow 1.3 protocol was used as a packet analyser. It is open source popular solution.

It is a common practice to make pre-built virtual machine for the purposes of making SDN learning materials. However, by making customised solution few benefits can be achieved:

- Smaller size by removing unused components and using special file system
- Tested versions of software give predictable outcomes which can be compared by students with their own results
- Predefined state of the SDN controller ensures predictable results

#### 4.3 Similar projects

“OpenFlow Tutorial” by Brandon Heller and Yiannis Yiakoumis. Tutorial shows usage of Mininet and use of various controllers from programmer’s perspective. For the reason that tutorial being developed is targeted on ICT students it would be preferable to avoid programming if possible.

Flowsim is a web-based learning environment. Project description promises to help to study OpenFlow without the need of configuration. However, there are concerns that simulations is accurate enough. Also, it is not possible to get inside the real distribution and change its contents. It should be possible with the real SDN controller.

#### 4.4 Implementation

Virtual environment gives some freedom of experimentation and unburdens us from managing state. Developing an environment is just two steps: configure virtual machine like the real machine and export it.

As a base for the virtual machine Ubuntu 14.04 Server was used as a distribution that is guaranteed to work with both OpenDayLight and Mininet. Server distribution was chosen to make image as small as possible. Parts of XFCE were used as desktop environment because of their small size and low hardware requirements.

OpenDayLight controller was downloaded using pre-built distribution from the official website. Installing it was as simple as unzipping archive. ODL proved to be unstable so multiple versions were tested to find the most stable release.

Some ODL versions work only with specific versions of Java Runtime Environment (JRE). More, by default Ubuntu ships with the Open JRE. ODL developers recommend using Oracle JDK for better performance. It was concluded that it makes sense to install Oracle JDK.

Then, some OpenDayLight packages were installed to allow student to work with DLUX web interface and let ODL manage the network out-of-the-box. Including these packages have several advantages. First, set-up is easier and faster. Second, lab will continue to work even if packages and distribution are no longer supported. Third, user of the lab will not make the mistake during installation. Fourth, the results should be closer to the ones in the instructions.

Mininet, open vSwitch and Wireshark installed by building from sources to get the most recent version and to have OpenFlow support in Wireshark. This bundle is provided by the Mininet developers.

Custom topology for Mininet were created. Mininet uses Python as scripting language for writing topologies. Topologies are developed in the imperative way which is easy but not simple. It could benefit from making topologies declarative.

Any difficulty unrelated unrelated to the subject of the lesson should be removed if possible. Launch scripts were made in Bash to help students start faster without the need to remember launch commands. With it, even students who does not have extensive experience of using command line can use the lab.

Desktop icons were made for the browser, Postman program and documentation. No icon were made for Wireshark because it has to be launched with root privileges.

## 5 STUDYING MATERIALS

### 5.1 Overview

Studying materials play very important role in educational process. Quality of instructions is as important as quality of instruments used and in tandem they can provide enjoyable and useful learning experience.

One more component that can make studying better is challenges. It is not unique to networking but it can be easily applied there and it was proven by many projects in KyUAS.

It was concluded that the materials will include instructions and challenges. Balance between making the lab too hard and too easy should be found as well as balance between between making it too long and too short.

### 5.2 Implementation

As a result, of observation of other students it is remarkable that for student it is hard to read all theory at once. Therefore, the theory parts are mixed with the practical parts. Required concepts are introduced only at the point where they are needed.

Simple topology of three switches and two hosts is used for the guide. First, user is going to verify that controller is operating properly and layer 2 functions are working. Then, one link is brought down and two hosts cannot reach each other. It happens because no feature in controller tracks link state and change flows accordingly. Student has to redirect traffic manually. First, instructions are given how to add flow on one device and then student has to change all other flows to make ping between hosts work. After the link is brought up again behaviour of switches and traffic flow are observed and explained.

Guide starts with brief explanation of what SDN and SDN controller is and what student will do following this tutorial. Also, explanations about lab design are given.

Step one starts with launching the controller. Explanations are given on how to shut down controller properly. It is important because unexpected shutdown can make controller dysfunctional.

Step two explains managing packages in OpenDayLight. Students are told how to install and list features (packages). This step is not required for the lab but is important if the user will decide to experiment with controller on his own.

Step three explains basic Mininet usage and gives some troubleshooting information.

Step 4 shows OpenDayLight web interface. Student make hosts ping each other. This gives controller enough information to display switches in the interface. This is the last preparation step, new steps start to give explanations on the technologies themselves.

During Step 5 explains OpenFlow flows, shows how to modify them and how to check flows in the controller model and on the device. Student gets flows of one the switches first via controller and then on the virtual device. During this step, students learns basics of communicating with controller using HTTP REST API.

In Step 6 use of Wireshark for capturing OpenFlow packets. Student can see how switches are sending ARP messages.

Step 7 is the first step in which student has to modify flows. One link is being shut down and student is challenged to figure out why controllers cannot ping each other despite having functional link between them. After that new flow should be pushed to the device using controller. Sniffing will show that traffic is flowing but only one way.

In the step 8, student is given the list of the flows which should be installed to fix the network. Process is almost the same as the previous push but student has to do it himself.

After applying the fix and bringing broken link back up new problem is discovered: duplicate packets are received. In step 9, student is given explanation on how to fix this but no detailed instructions are given.

Some steps in the tutorial were invented as the result of the struggles while making the lab. This can possibly make experience of going through the lab more challenging and interesting. More than a half of the work should be done by the student himself. This kind of lab works best when aid by a teacher because of its challenging nature.

### 5.3 Feedback

The work on the lesson and lab environment was done with few iterations. Iteration cycle can be described as following: a student tries to follow the instructions. In case if the student experience difficulties a note is made about it.

Instructions and lab environment were tested on two students. Both students are exchange students who did Bachelor degree in IT at KYAMK.

Most of the difficulties were related to the unclear instructions or missing explanations.

Both students said that they found exercise interesting but slightly longer than they expected it to be.

After collecting feedback, the following changes were made:

- Elaborate explanations about Karaf systems were removed
- Explanations about host discovery were added
- Section about global configuration using XML was removed
- More pictures were added to make instructions clearer.
- Instruction to not close terminals was added
- More detailed Wireshark instructions were given
- More detailed instructions on pushing additional flows were added
- During the pinging between virtual host student can see the message "0% dropped". As this may be confusing, explanations were given that this is a desired result.

- During the tests it became clear that students may not have deep understanding of HTTP protocol thus parts of the tutorial which involved making HTTP request we made simpler and more approachable  
It should be noted that students' observation showed how important it is to test materials and environment. Without any testing outcome would be harder to use.

## 6 CONCLUSIONS

### 6.1 Results

Result can be considered as a success and the virtual environment can be either used as-is for self-studying or it can be extended to be used in part of a bigger course about SDN. This lab is unable to prepare one for the job in the SDN field but it can serve as an introduction and help to understand what is SDN and how OpenFlow works. Teaching SDN is inevitable and it is better to be prepared for new technologies.

Using Mininet as a simulation tool proved that it is a robust, lightweight and malleable tool for network simulations. It can be used for many types of research and teaching. There is a reason to think that most teaching in the networking field will be done using simulated networks. There is a huge room for improvement. Simulated network can augment real ones for creation of interesting scenarios.

Because most of the components of the lab are small the product is relatively small and can be easily moved over the network. The lab contains only open-source software and thus can be freely distributed and improved. These traits make the product applicable in the wide range of scenarios.

Extending the lab does not involve any difficulties. Any topology can be added using simple script and if some configuration should be applied to controller, its folder can be just copied and altered as needed without the need to compile whole virtual machine.

## 6.2 Challenges experienced

During the development of the lab many issues arose. Most of them were related to the immaturity of OpenDayLight and lack of up-to-date documentation on it.

OpenDayLight project founded just in 2013 and the platform experienced at least one major change. The biggest change is moving to Karaf platform. Some materials about ODL produced before this move became irrelevant. Even after the move platform is being actively developed.

Immaturity of ODL caused issues with using it. Some actions could break the system completely. Not only ODL is unstable itself, but most of the functionality is provided by the components which are developed separately. This made reinstallation of the platform harder because some versions of components could be unstable and it lead to unpredictable results. For this reason, not all features of DLUX web interface are used and HTTP request are made using Postman instead which hurts user interaction and makes studying more confusing. There is a positive side-effect, though: http requests are less likely to change than the user interface and the lab will stay actual longer. Maturation of the platform should resolve most of these issues.

## 6.3 Ideas for future use

Right now the lab can serve only as an introduction or a first practical lesson on SDN. Despite this, it has potential to be a foundation for a course on SDN. More topologies and challenges can be added. The lab can work with real devices or with other types of simulations. For example, VIRL can be launched in a separate virtual machine and be connected to the controller using bridge.

Advanced topics such as clustering can be taught. There are a lot of southbound protocols controller can work with, for example, BGP can be used.

It is possible to use controller as part of the Service Provider Networks course. There is a lot of configuration to be done and big network is used. Students



can see the difference between manual configuration and using software based solutions.

Altered lab can be used as part of the Network Security or Hacking courses. Cases such as DDoS attack mitigation can be simulated. It can be done as a challenge: some part of students are doing an attack on the network while another tries to mitigate it.

#### 6.4 SDN technologies maturity

At this moment, SDN technologies are still very young and their application is still limited to the specific use cases. While stability of different SDN controllers may differ there is still lack of materials on the topic available, especially on edge-cases.

Implementing SDN makes most sense in multi-tenant data centers, provider networks and in dynamic environments right now. In the future, we can expect increased adoption across different types of networks because of the benefits SDN gives such as flexibility, transparent overview of the network and research capabilities.

SDN adoption is also likely to increase demand in programmers with knowledge of SDN who can build a connection with high-level services and network.

In a dynamic environment, we can expect increased demand for networks architects who can plan and invent new types of networks on the fly to be implemented right away.

Combining SDN with machine learning can lead to new self-learning networks which can be improved by tries and feedback.

## LIST OF REFERENCES

Burrus, D. (n.d.). *The Internet of Things Is Far Bigger Than Anyone Realizes*. [ONLINE] WIRED. Available at: <http://www.wired.com/insights/2014/11/the-internet-of-things-bigger/> [Accessed: 1 May 2016].

Chen, C. DDoS Attack Detection & Mitigation in SDN, 2015  
<http://www.slideshare.net/chaochen5245961/cc3736-slides>

D. Nadeau, T. and Gray, K. (2013). *SDN - Software Defined Networks*. Sebastopol, CA

Distributed ONOS - ONOS - Wiki. 2016. Distributed ONOS - ONOS - Wiki. [ONLINE] Available at: <https://wiki.onosproject.org/display/ONOS/Distributed+ONOS>. [Accessed: 04 September 2016].

Flowgrammable. 2016. Instructions & Actions. [ONLINE] Available at: <http://flowgrammable.org/sdn/openflow/actions>. [Accessed: 13 April 2016].

Grady, J., Price, C., Christiansen, C. and Richmond, C. (2013). *Worldwide DDoS Prevention Products and Services 2013-2017 Forecst*. 1st ed. Framingham, MA USA: International Data Corporation, p.3.

Kreutz, Diego et al. *Software-Defined Networking: A Comprehensive Survey*. Proceedings of the IEEE 103.1 (2015): 14-76. Web.

Krishnan, R., Durrani, M. and Phaal, P. (n.d.). *Real-time SDN Analytics for DDoS mitigation*. Brocade Communications, p.2. Available at: <http://opennetsummit.org/archives/mar14/site/pdf/2014/sdn-idol/Brocade-SDN-Idol-Proposal.pdf> [Accessed: 1 May 2016].

Ma, Chloe. *SDN Secrets Of Amazon And Google* [ONLINE] InfoWorld (2014): Web. Available at: <http://www.infoworld.com/article/2608106/sdn/sdn-secrets-of-amazon-and-google.html> [Accessed: 2 Nov. 2016].

Moshref, Masoud; Bhargava, Apoorv; Gupta, Adhip; Yu, Minlan; Govindan, Ramesh, *Flow-Level State Transition As A New Switch Primitive For SDN*, University of Southern California, 2014. [ONLINE] Available at:

<http://dl.acm.org/citation.cfm?id=2631439> [Accessed: 25 September 2016]

OpenDayLight FAQ. 2016. OpenDayLight FAQ. [ONLINE] Available at:

<https://www.opendaylight.org/faq>. [Accessed: 22 September 2016].

OpenDaylight Project, *OpenDaylight Controller:MD-SAL:Architecture:Clustering* 2016

[ONLINE] Available at: [https://wiki.opendaylight.org/view/OpenDaylight\\_Controller:MD-SAL:Architecture:Clustering](https://wiki.opendaylight.org/view/OpenDaylight_Controller:MD-SAL:Architecture:Clustering). [Accessed: 04 September 2016].

Ram (Ramki) Krishnan, Muhammad Durrani, Brocade Communications, *Real-time SDN Analytics for DDoS mitigation*,

<http://opennetsummit.org/archives/mar14/site/pdf/2014/sdn-idol/Brocade-SDN-Idol-Proposal.pdf>

Salisbury, Brent. *Openflow: Proactive Vs Reactive*. NetworkStatic., 2013. [ONLINE]

<http://networkstatic.net/openflow-proactive-vs-reactive-flows> [Accessed: 25 September 2016]

Vaughn, M. (2012) *Virtualization makes big data analytics possible for SMBs*.

[ONLINE] TechTarget. Available at:

<http://searchservvirtualization.techtarget.com/feature/Virtualization-makes-big-data-analytics-possible-for-SMBs> [Accessed: 2 November 2016].

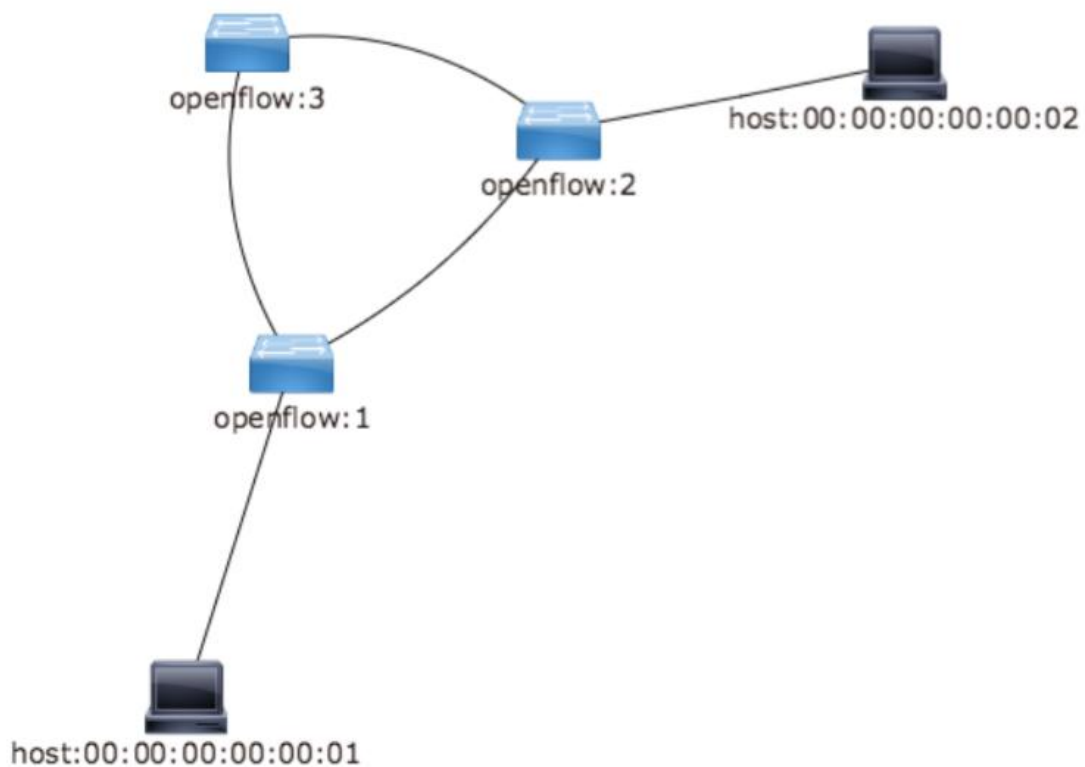
Ward, D. 2013. Foreword by David Ward. In: D. Nadeau, T. and Gray, K. (2013). *SDN - Software Defined Networks*. Sebastopol, CA

# OpenDayLight SDN Controller Lab Guide 1

## Objectives

After going through this tutorial you will understand basic concepts of SDN, get skills of configuration and use of OpenDayLight and basic knowledge of OpenFlow. Also you will understand how to use pre-defined Mininet topologies, will use Postman and Wireshark as instruments.

## Topology



## Studying environment

This Lab guide is intended to be used with prepared virtual machine, which includes pre-installed ODL controller, network simulator Mininet, packet analyser Wireshark, browser Chromium with Postman add-on.

Ubuntu credentials:

user: lab, password: lab

ODL credentials::

user: admin, password: admin

## Theory background

You're going to work with Software Defined Network controller and few SDN-enabled virtual switches. Each switch has multiple tables similar to routing tables. They're called flow tables and they're used by the controller to determine which action should be taken (send through the port, drop, pass to another table or send packet to the controller). Flow tables can match packets using different fields.

We're going to manipulate flow tables by making request to the controller.

## Steps

---

### Step 1 Launching the controller

First launch ODL controller.

**To do it open terminal and and issue following command**

```
./distribution-karaf/bin/karaf
```

After some time you should see the following prompt message:

```
llab@ubuntu-14:~$ ./distribution-karaf/bin/karaf  
  
Hit '<tab>' for a list of available commands  
and '[cmd] --help' for help on a specific command.  
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown OpenDaylight.  
opendaylight-user@root>
```

This is a OpenDayLight console. It is used to manage controller.

You may want to make a terminal window wider.

On any stage pressing Tab will make controller autocomplete typed command.

To stop the controller do one of the following:

type

```
system:shutdown
```

or type

```
logout
```

or press Control-D.

You should keep OpenDayLight and Mininet terminal windows open until the end of the tutorial, otherwise you will stop one of them.

---

## Step 2 Features installing

To make controller do certain tasks certain features have to be installed. To list all available features type

```
feature:list
```

To filter only installed features

```
feature:list -i
```

Virtual machine is distributed with pre-installed features but to install them following command can be used:

**IT IS NOT RECOMMENDED TO INSTALL FEATURES IN THE LAB ENVIRONMENT**

It may lead to unpredictable results

This command is just for the reference.

```
feature:install odl-l2switch-switch odl-restconf-all odl-mdsal-apidocs odl-dlux-all
```

Following features are:

odl-l2switch-switch will add switch capabilities to our controller and will allow our simulated devices be ruled by the controller.

odl-restconf-all will add REST HTTP interface and will allow to change models of the controller using HTTP requests

odl-mdsal-apidocs will install API explorer which will allow us to browser through the API of the controller

odl-dlux-all will install DLUX web interface

After installing features verify that they're present in the controller

```
feature:list -i
```

```

opendaylight-user@root>feature:list -l

```

Name	Version	Installed	Repository	Description
odl-yangtools-yang-data	0.8.1-Beryllium-SR1	x	odl-yangtools-0.8.1-Beryllium-SR1	OpenDaylight :: Yangtools :: Data Binding
odl-yangtools-common	0.8.1-Beryllium-SR1	x	odl-yangtools-0.8.1-Beryllium-SR1	OpenDaylight :: Yangtools :: Common
odl-yangtools-yang-parser	0.8.1-Beryllium-SR1	x	odl-yangtools-0.8.1-Beryllium-SR1	OpenDaylight :: Yangtools :: YANG Parser
odl-l2switch-switch	0.3.1-Beryllium-SR1	x	l2switch-0.3.1-Beryllium-SR1	OpenDaylight :: L2Switch :: Switch
odl-l2switch-hosttracker	0.3.1-Beryllium-SR1	x	l2switch-0.3.1-Beryllium-SR1	OpenDaylight :: L2Switch :: HostTracker
odl-l2switch-addresstracker	0.3.1-Beryllium-SR1	x	l2switch-0.3.1-Beryllium-SR1	OpenDaylight :: L2Switch :: AddressTracker
odl-l2switch-arphandler	0.3.1-Beryllium-SR1	x	l2switch-0.3.1-Beryllium-SR1	OpenDaylight :: L2Switch :: ArpHandler
odl-l2switch-loopremover	0.3.1-Beryllium-SR1	x	l2switch-0.3.1-Beryllium-SR1	OpenDaylight :: L2Switch :: LoopRemover
odl-l2switch-packethandler	0.3.1-Beryllium-SR1	x	l2switch-0.3.1-Beryllium-SR1	OpenDaylight :: L2Switch :: PacketHandler
odl-mdsal-models	0.8.1-Beryllium-SR1	x	odl-mdsal-models-0.8.1-Beryllium-SR1	OpenDaylight :: MD-SAL :: Models
odl-akka-scala	2.11	x	odl-controller-1.6.1-Beryllium-SR1	Scala Runtime for OpenDaylight
odl-akka-system	2.3.14	x	odl-controller-1.6.1-Beryllium-SR1	Akka Actor Framework System Bundles
odl-akka-clustering	2.3.14	x	odl-controller-1.6.1-Beryllium-SR1	Akka Clustering
odl-akka-leveldb	0.7	x	odl-controller-1.6.1-Beryllium-SR1	LevelDB
odl-akka-persistence	2.3.14	x	odl-controller-1.6.1-Beryllium-SR1	Akka Persistence
odl-aaa-api	0.3.1-Beryllium-SR1	x	odl-aaa-0.3.1-Beryllium-SR1	OpenDaylight :: AAA :: APIs
odl-aaa-authn	0.3.1-Beryllium-SR1	x	odl-aaa-0.3.1-Beryllium-SR1	OpenDaylight :: AAA :: Authentication - NO CLUSTER
odl-config-persister	0.4.1-Beryllium-SR1	x	odl-config-persister-0.4.1-Beryllium-SR1	OpenDaylight :: Config Persister
odl-config-startup	0.4.1-Beryllium-SR1	x	odl-config-persister-0.4.1-Beryllium-SR1	OpenDaylight :: Config Persister:: Config Startup
odl-config-netty	0.4.1-Beryllium-SR1	x	odl-config-persister-0.4.1-Beryllium-SR1	OpenDaylight :: Config-Netty
odl-openflowplugin-southbound	0.2.1-Beryllium-SR1	x	openflowplugin-0.2.1-Beryllium-SR1	OpenDaylight :: Openflow Plugin :: SouthBound
odl-openflowplugin-flow-services	0.2.1-Beryllium-SR1	x	openflowplugin-0.2.1-Beryllium-SR1	OpenDaylight :: Openflow Plugin :: Flow Services
odl-openflowplugin-nsf-services	0.2.1-Beryllium-SR1	x	openflowplugin-0.2.1-Beryllium-SR1	OpenDaylight :: OpenflowPlugin :: NSF :: Services
odl-openflowplugin-nsf-model	0.2.1-Beryllium-SR1	x	openflowplugin-0.2.1-Beryllium-SR1	OpenDaylight :: OpenflowPlugin :: NSF :: Model
odl-openflowplugin-app-config-pusher	0.2.1-Beryllium-SR1	x	openflowplugin-0.2.1-Beryllium-SR1	OpenDaylight :: Openflow Plugin :: app - default c
odl-openflowplugin-app-lldp-speaker	0.2.1-Beryllium-SR1	x	openflowplugin-0.2.1-Beryllium-SR1	OpenDaylight :: Openflow Plugin :: app lldp-speake
odl-mdsal-binding-base	2.0.1-Beryllium-SR1	x	odl-yangtools-2.0.1-Beryllium-SR1	OpenDaylight :: MD-SAL :: Binding Base Concepts
odl-mdsal-binding-runtime	2.0.1-Beryllium-SR1	x	odl-yangtools-2.0.1-Beryllium-SR1	OpenDaylight :: MD-SAL :: Binding Generator
pax-jetty	8.1.15.v20140411	x	org.ops4j.pax.web-3.1.4	Provide Jetty engine support
pax-http	3.1.4	x	org.ops4j.pax.web-3.1.4	Implementation of the OSGi HTTP Service
pax-http-whiteboard	3.1.4	x	org.ops4j.pax.web-3.1.4	Provide HTTP Whiteboard pattern support
pax-war	3.1.4	x	org.ops4j.pax.web-3.1.4	Provide support of a full WebContainer
standard	3.0.3	x	standard-3.0.3	Karaf standard feature
config	3.0.3	x	standard-3.0.3	Provide OSGi ConfigAdmin support
region	3.0.3	x	standard-3.0.3	Provide Region Support
package	3.0.3	x	standard-3.0.3	Package commands and mbeans
http	3.0.3	x	standard-3.0.3	Implementation of the OSGi HTTP Service
war	3.0.3	x	standard-3.0.3	Turn Karaf as a full WebContainer
kar	3.0.3	x	standard-3.0.3	Provide KAR (KARaf archive) support
ssh	3.0.3	x	standard-3.0.3	Provide a SSHd server on Karaf
management	3.0.3	x	standard-3.0.3	Provide a JMX MBeanServer and a set of MBeans in K
odl-dlux-all	0.3.1-Beryllium-SR1	x	odl-dlux-0.3.1-Beryllium-SR1	OpenDaylight dlux all features
odl-dlux-core	0.3.1-Beryllium-SR1	x	odl-dlux-0.3.1-Beryllium-SR1	OpenDaylight dlux minimal feature
odl-dlux-node	0.3.1-Beryllium-SR1	x	odl-dlux-0.3.1-Beryllium-SR1	Enable nodes in OpenDaylight dlux
odl-dlux-yangui	0.3.1-Beryllium-SR1	x	odl-dlux-0.3.1-Beryllium-SR1	Enable Yang UI in OpenDaylight dlux
odl-dlux-yangvisualizer	0.3.1-Beryllium-SR1	x	odl-dlux-0.3.1-Beryllium-SR1	Enable Yang visualizer in OpenDaylight dlux
odl-netty	4.0.33.Final	x	odparent-1.6.1-Beryllium-SR1	OpenDaylight :: Netty

Part of the output.



---

### Step 3 Mininet network

While it may take some time to install and launch new features we can launch our network simulation.

**Open another terminal tab or window and type**

```
./start_topo.sh
```

Followed by password 'lab'.

```
lab@ubuntu-14:~$ nano start_topo.sh
lab@ubuntu-14:~$ ./start_topo.sh
++ sudo mn --custom simple.py --topo three_simple --mac --controller remote --switch ovsk,protocols=OpenFlow13
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (s1, s2) (s1, s3) (s2, h2) (s2, s3)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> █
```

If output will include 'cannot connect to remote controller' please type 'exit' and try again later. Usually it means that controller didn't boot properly.

This script launches Mininet network with custom topology.

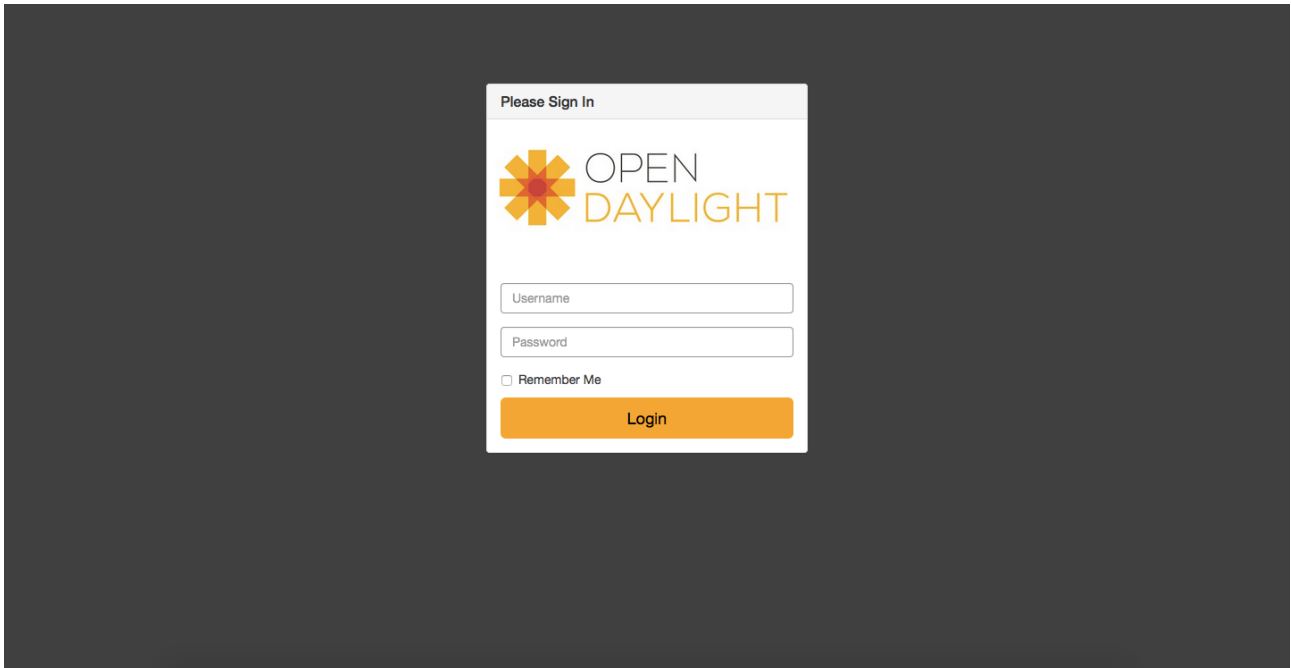
—controller remote by default assumes ip=127.0.0.1 and ports 6633 and 6653 (OpenFlow).

---

## Step 4 DLUX web interface

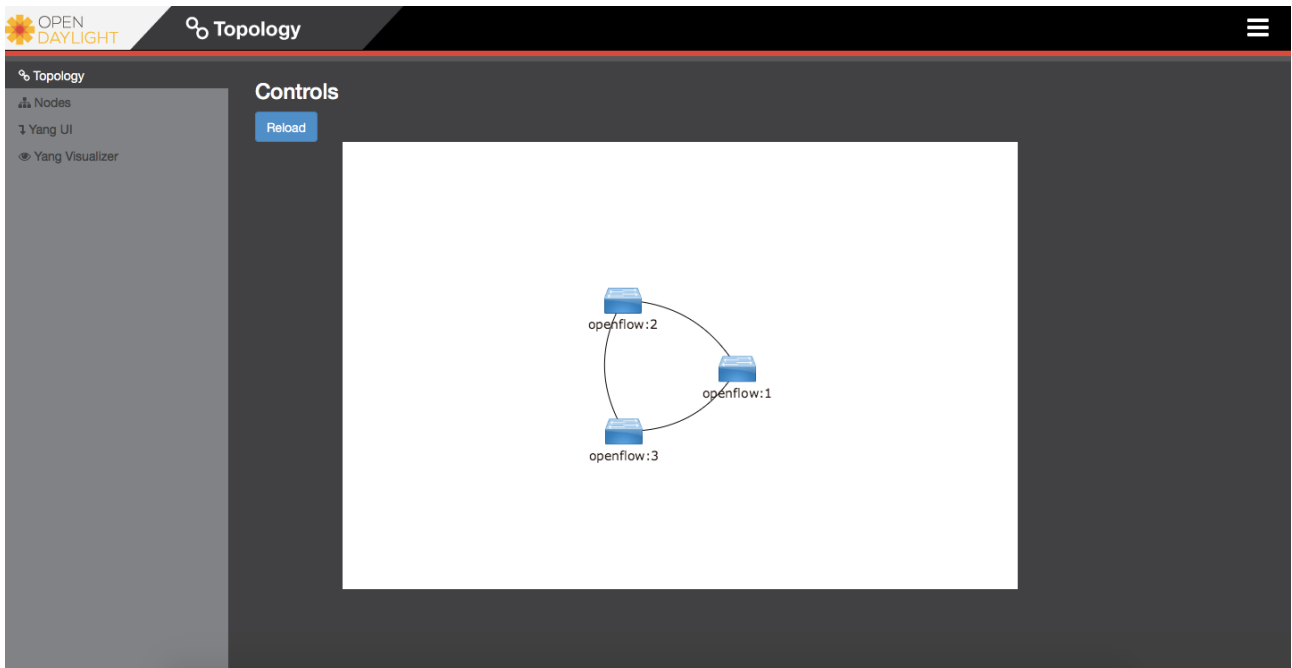
To access DLUX web interface **open web browser and open url:**

127.0.0.1:8181/index.html



Log in to the DLUX interface with username and password *admin*.

In case of failed login try to wait more.



DLUX web interface contains 4 sections: Topology, Nodes, Yang UI and Yang Visualiser.

Topology displays current operational topology discovered from messages sent to controller.

Nodes represent information about each single device (flow information may not work properly).

Yang UI allows to view and change models of the controller (we're gonna use HTTP requests instead).

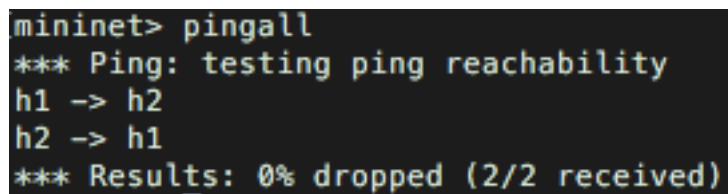
Yang Visualiser allow to view models as graphs.

Due to instability of DLUX only first two sections will be used in this tutorial. You're free to explore them yourself.

Right now we can use only switches and we don't see any hosts. That's because no ARP requests were sent by the hosts and sent to the controller.

**Go to the Mininet prompt and type**

```
pingall
```



```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
```

The following result should be visible:

*0% dropped* means that all packets reached their destination.

This will make all hosts to ping each other. We could also do

```
h1 ping h2
```

*h1* in the front tells Mininet to do this command from the perspective of the host *h1*.

We could do

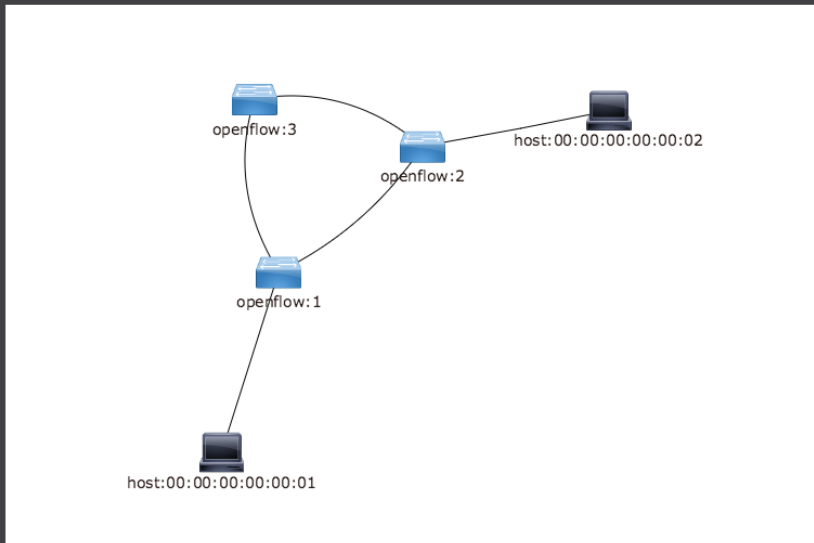
```
h1 ifconfig
```

To view all interfaces of *h1*.

Ping should be successful.

## Controls

Reload



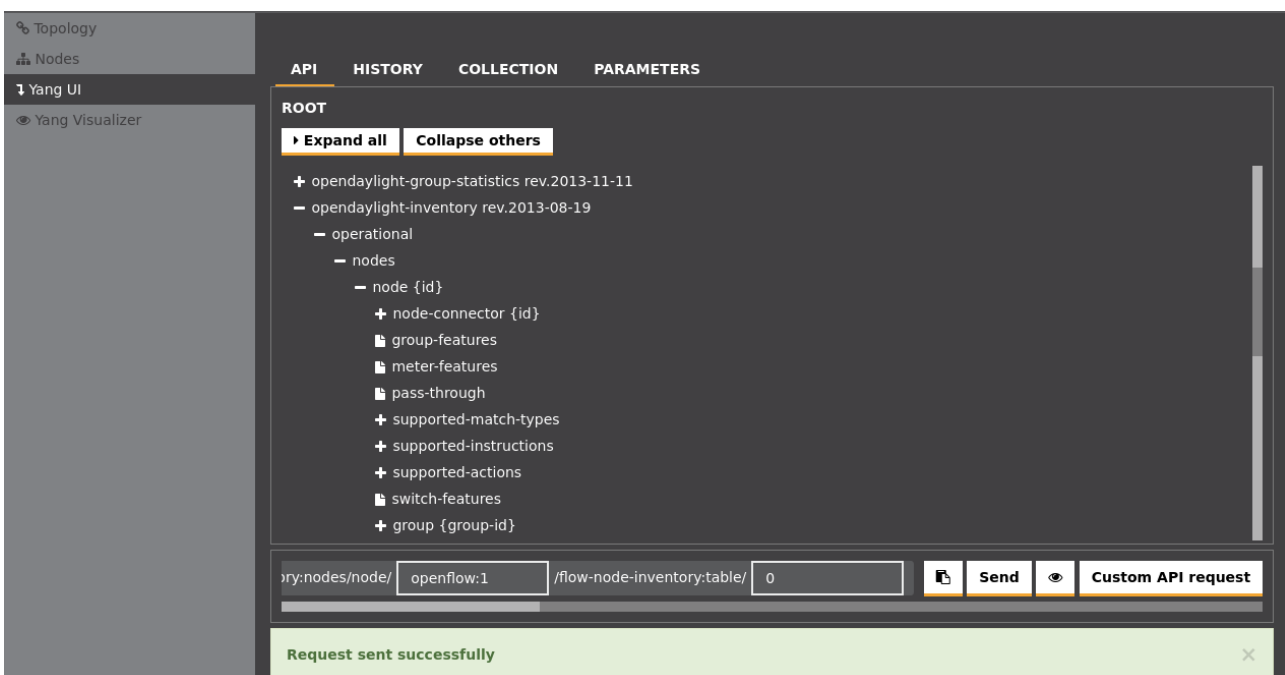
After pinging press *Reload* button in DLUX topology interface. Hosts should be seen now (hosts' IPs can be seen by hovering over them).

## Step 5 OpenFlow

Flow defines what action should device take upon receiving a packet. Flows are separated into the tables. Each flow has a priority, match and action fields. In case if packet had no match in the current table it is processed using the next one. If packet had no match in any table default action is taken. In ODL default action is configured in XML configuration is out of scope of this tutorial. Specification states that only the flow with the top priority will be used in case of overlapping flows.

A purpose of the odl-l2switch-switch package is to provide layer 2 switching functionality to the devices. By default it installs flow for the shortest path. We can check installed flows in 2 ways: HTTP request to the controller and issuing command on the device.

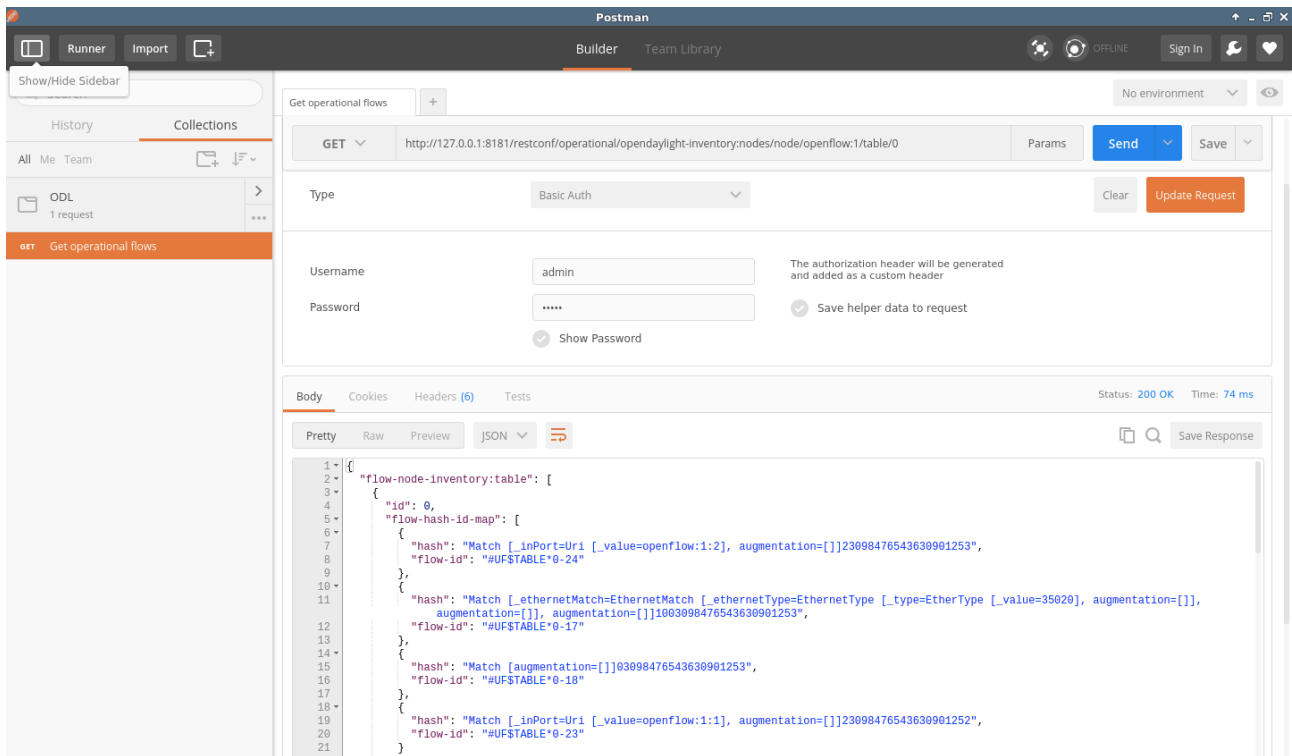
*Note: You can reach REST API using Yang UI in case if it works for you.*



Example of using Yang UI

### Open Postman app.

From the left panel choose *ODL* collection and **select *get operational flows* request.**  
**Press send.**



Example of using Postman to send HTTP requests.

After making this request we should see flow table 0 of the switch 1. Look at the URL and try to understand how it's formed. *openflow:1* stands for the first device while number in the end is indicating the number of table on the device.

Since we requested the *operational* topology, we see some flows even though we didn't install any yet. Flows whose id start with *#UF\$TABLE* are installed automatically. In this case they're installed by I2switch feature.

We're interested at *match* and *instructions* fields of flows.

```
25     "id": "#UF$TABLE*0-5",
26     "opendaylight-flow-statistics:flow-statistics": {},
34     "cookie": 3098476543630901252,
35     "match": {
36         "ethernet-match": {
37             "ethernet-type": {
38                 "type": 35020
39             }
40         }
41     },
42     "table_id": 0,
43     "flags": "",
44     "idle-timeout": 0,
45     "hard-timeout": 0,
46     "instructions": {
47         "instruction": [
48             {
49                 "order": 0,
50                 "apply-actions": {
51                     "action": [
52                         {
53                             "order": 0,
54                             "output-action": {
55                                 "output-node-connector": "CONTROLLER",
56                                 "max-length": 65535
57                             }
58                         }
59                     ]
60                 }
61             }
62         ]
63     }
```

One of them have matches packets with *ethernet-type* 35020. It matches all LLDP packets (more information about ethernet types can be found here <http://www.iana.org/assignments/ieee-802-numbers/ieee-802-numbers.xhtml>). Look at the *instructions* section of this list. It has one action which states that *output-node-connector* should be *CONTROLLER*. What this flow does is sending all LLDP packets to controller. This way controller know states of the links.

Do not be confused with two *order* fields in *instructions* section. One of them means that *apply-actions* section should be used first (OpenFlow has not only output actions but has other types of actions).The second one is significant to action list inside *apply-actions* block.

```

    "id": "#UF$TABLE*0-3",
    "opendaylight-flow-statistics:flow-statistics": {←},
    "cookie": 3098476543630901252,
    "match": {
      "in-port": "openflow:1:1"
    },
    "table_id": 0,
    "flags": "",
    "idle-timeout": 0,
    "hard-timeout": 0,
    "instructions": {
      "instruction": [
        {
          "order": 0,
          "apply-actions": {
            "action": [
              {
                "order": 1,
                "output-action": {
                  "output-node-connector": "CONTROLLER",
                  "max-length": 65535
                }
              },
              {
                "order": 0,
                "output-action": {
                  "output-node-connector": "2",
                  "max-length": 65535
                }
              }
            ]
          }
        }
      ]
    }
  }
}

```

Let's look at two other flows on the switch. One of them matches packets from port 1 (port going to the host) and sends it to the port 2 (to switch 2) and to the controller. Packets from this port are sent to the controller because it's a host port. Likely it's ARP handler feature which installed this rule. This way controller discovers hosts (like we saw on the step 2).

Let's look at the flows on the device itself. We're using Mininet with open vSwitch - software switches.

### Open new terminal tab and type

```
./dump_flows.sh
```

This script uses a command `sudo ovs-ofctl -O OpenFlow13 dump-flows s1`

```

lab@ubuntu-14:~$ ./dump_flows.sh
++ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
cookie=0x2b00000000000004, duration=6231.763s, table=0, n_packets=6, n_bytes=420, priority=2,in_port=1 actions=output:2,CONTROLLER:65535
cookie=0x2b00000000000005, duration=6231.739s, table=0, n_packets=7, n_bytes=490, priority=2,in_port=2 actions=output:1
cookie=0x2b00000000000005, duration=6237.605s, table=0, n_packets=2496, n_bytes=212160, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x2b00000000000005, duration=6237.605s, table=0, n_packets=23, n_bytes=1866, priority=0 actions=drop
lab@ubuntu-14:~$

```

In the output we can see all flows we've seen in the RESTCONF output plus one more. It says that packets which didn't match any other flow should be dropped. It is a default behaviour and can be changed statically through XML files.



## Step 6 Wireshark sniffing

**Open new terminal window** (or use the one without Mininet or ODL running) and **type**

```
sudo wireshark
```

This will launch Wireshark packet analyser with root privileges.

In the interface list **choose lo and press start**. Filter packets by **typing of** into the field on the top. Now you see all messages between devices and controller. Mostly it's metrics.

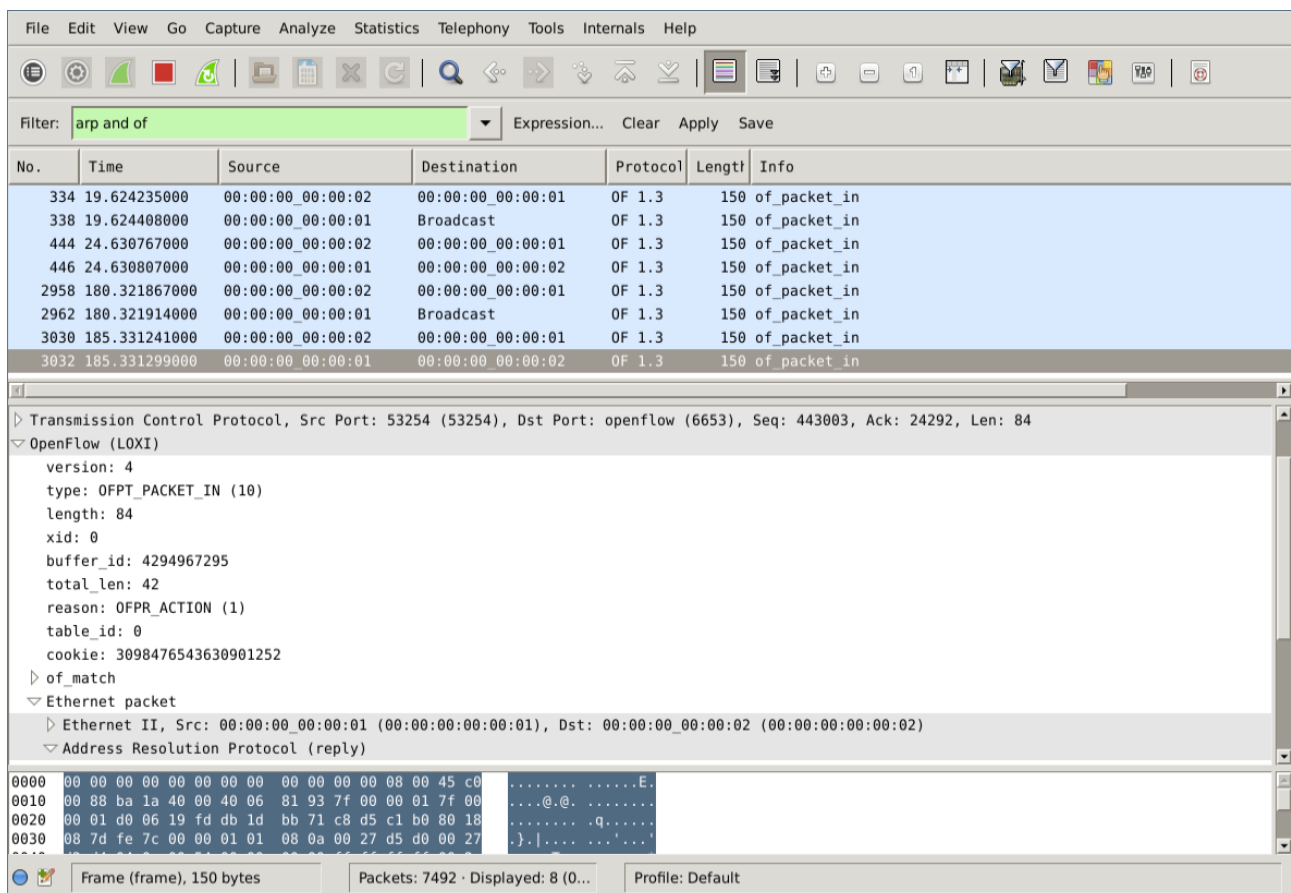
**Change the filter to *arp and of***. Now go to the terminal window with Mininet running and **do two following commands**:

```
h1 arp -d 10.0.0.2
```

```
h1 ping h2 -c 3
```

The first one deletes entry for host 2 from ARP table, the second one is a regular ping.

After that come back to the Wireshark window. Now we should see some packets.



The screenshot shows the Wireshark interface with a packet capture filter set to 'arp and of'. The packet list pane shows several packets, with the selected packet (No. 3032) having a time of 185.331299000. The packet details pane shows the following structure:

- Transmission Control Protocol, Src Port: 53254 (53254), Dst Port: openflow (6653), Seq: 443003, Ack: 24292, Len: 84
- OpenFlow (LOXI)
  - version: 4
  - type: OFPT\_PACKET\_IN (10)
  - length: 84
  - xid: 0
  - buffer\_id: 4294967295
  - total\_len: 42
  - reason: OFPR\_ACTION (1)
  - table\_id: 0
  - cookie: 3098476543630901252
  - of\_match
    - Ethernet packet
      - Ethernet II, Src: 00:00:00\_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00\_00:00:02 (00:00:00:00:00:02)
      - Address Resolution Protocol (reply)

The packet bytes pane shows the raw data for the selected packet, including the Ethernet II header and the ARP payload.

We see OpenFlow packets which contain ARP messages inside them.

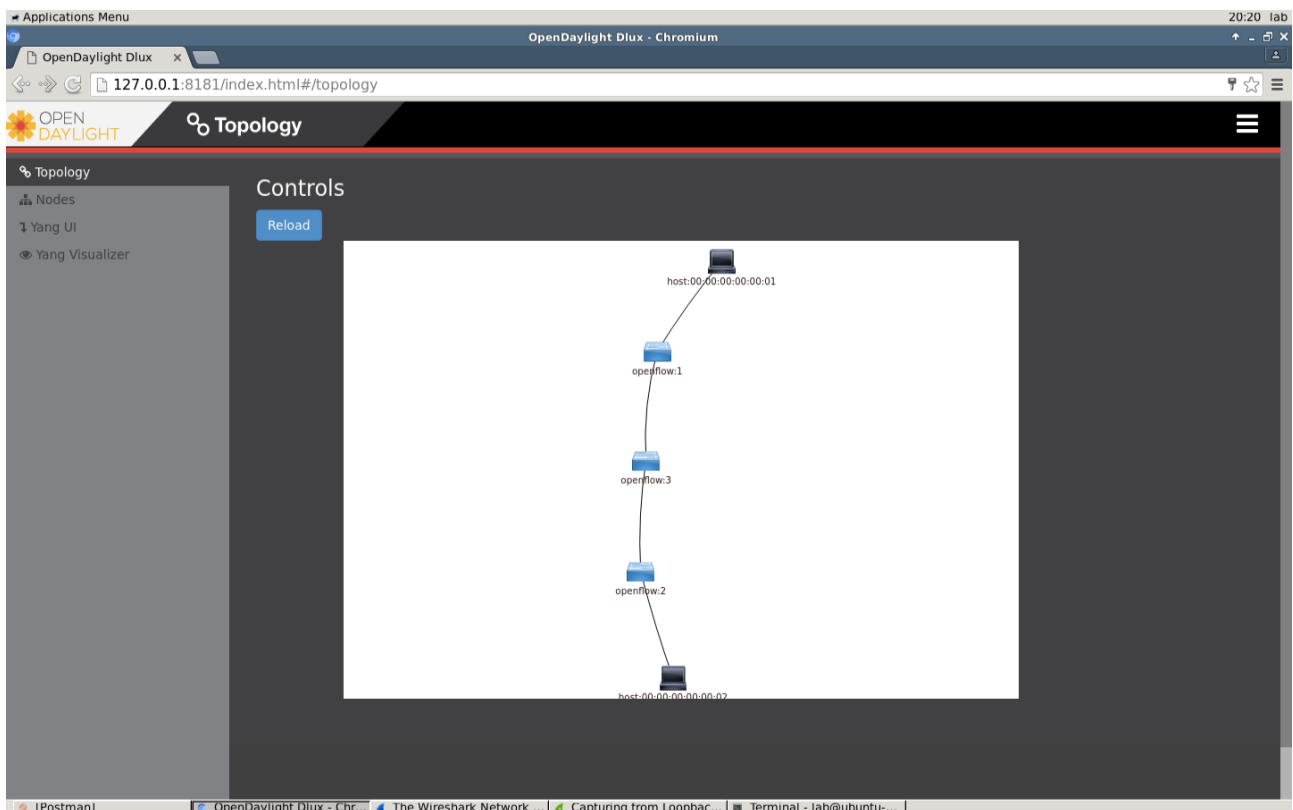
## Step 7 Static flow

### Go to the Mininet window and type

```
link s1 s2 down
```

Try to ping between h1 and h2. Does the ping work?

Go to the DLUX web interface and check the topology. It should look like this now.



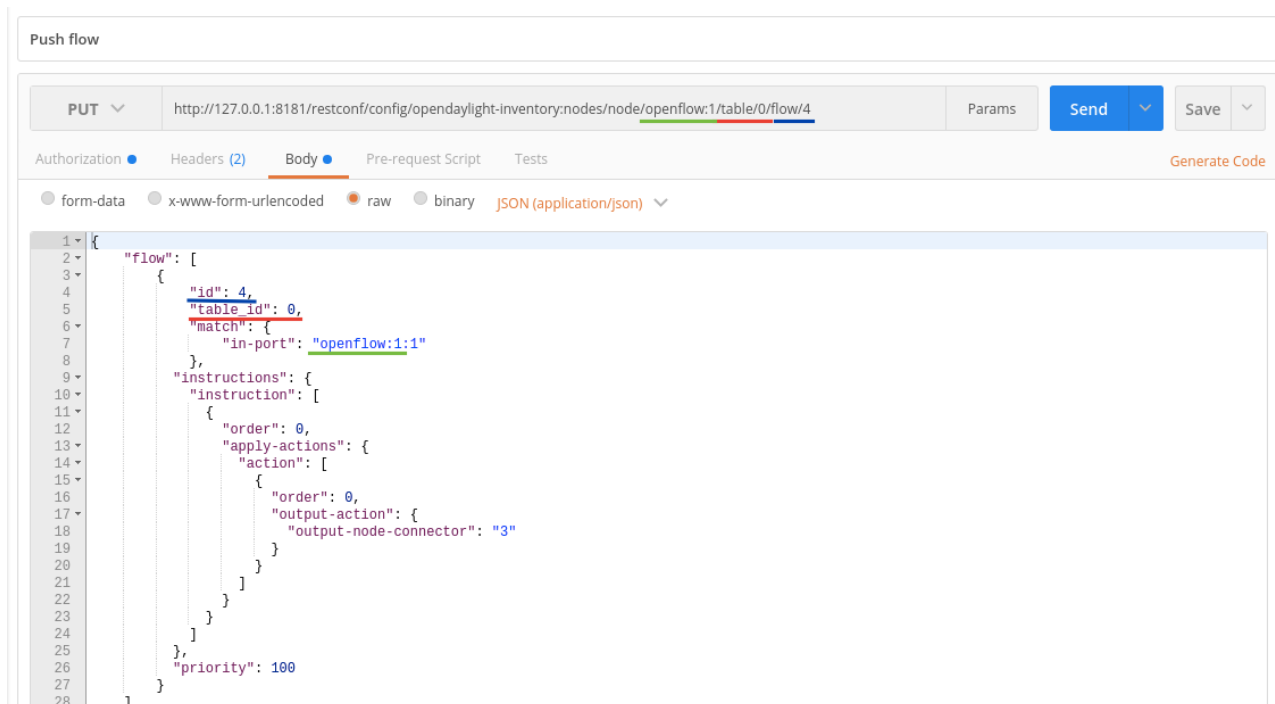
Use Wireshark to capture interface `s1-eth3` and check if ARP or ICMP packets are sent through the link. Try to think of a reason why. Press round grey button on the left, stop the capture, uncheck all interfaces and check `s1-eth3`. Start the new capture (you don't have to save the old capture).

The solution to the problem lies in the flow. If you will check the *operational* flow table now, you will see that it did not change. Our controller miss features that track if the link is down and change flows accordingly. Dynamic flows are out of scope of this tutorial.

To put a static flow to the switch we're going to push our model to the controller. After that controller automatically creates and sends update signal to the devices.

To push a flow switch to the Postman window and get *config* flows for the device *openflow:1*. Request is written for this switch by default so you can just launch it. Examine the output below the request. It should say that data model is missing. This is because we did not create any model for it.

To push flow into the device **choose *Push flow* command and choose the *Body* section below the URL.**



The screenshot shows a Postman interface for a PUT request. The URL is `http://127.0.0.1:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/4`. The request body is a JSON object with the following structure:

```
1 {
2   "flow": [
3     {
4       "id": 4,
5       "table_id": 0,
6       "match": {
7         "in-port": "openflow:1:1"
8       },
9       "instructions": {
10        "instruction": [
11          {
12            "order": 0,
13            "apply-actions": {
14              "action": [
15                {
16                  "order": 0,
17                  "output-action": {
18                    "output-node-connector": "3"
19                  }
20                }
21              ]
22            }
23          }
24        ]
25      },
26      "priority": 100
27    }
28  ]
29 }
```

Request body looks similar to response body.

Request body has to be consistent with URL - if you change the device, flow id or the table in the URL you should change them in the table and vice versa.

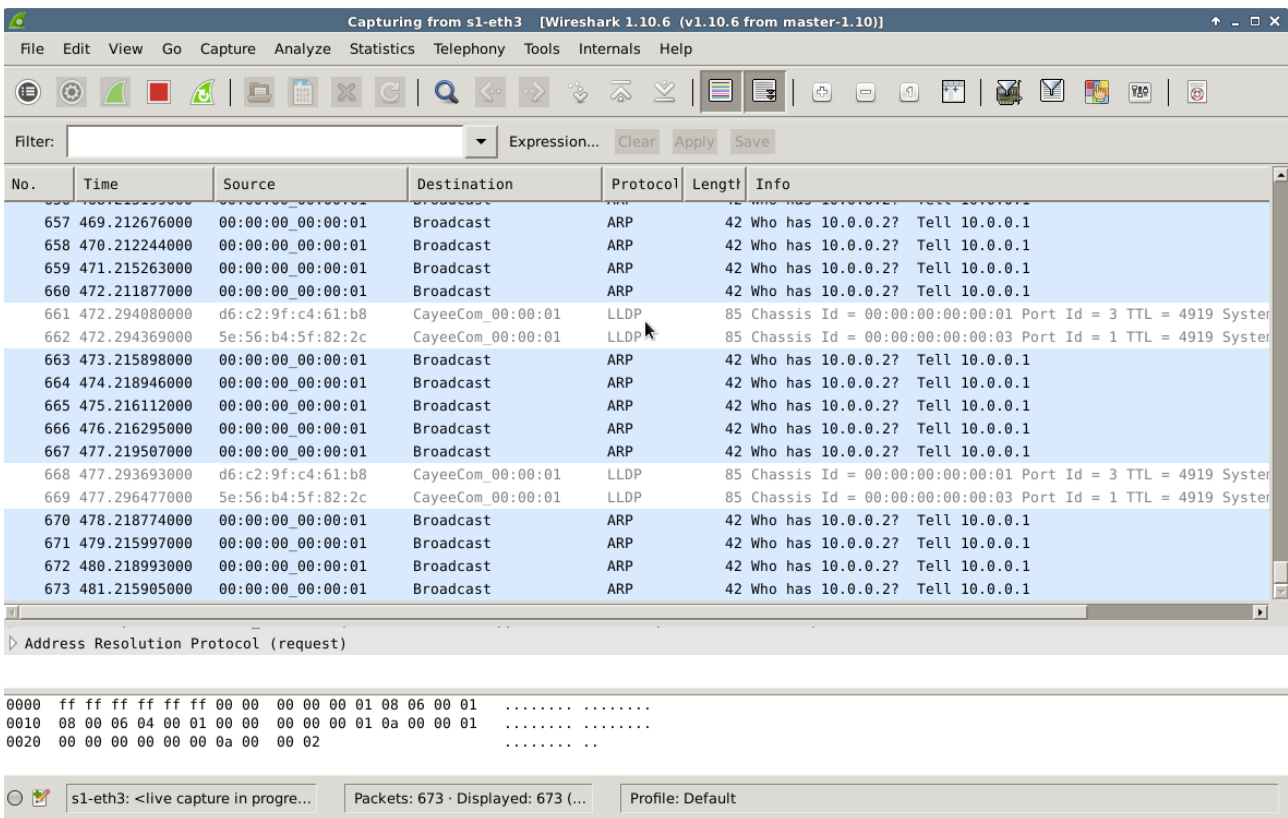
In this case we're pushing a flow on the device *openflow:1* to the *table 0* and flow id is *4*. Send request. Result should be with empty body and status code 200.

Note that for changing flows we're sending HTTP PUT request to the controller. It means that we're changing the flow tables.

**Troubleshooting note:** check authentication of the request. It should be *Basic*, login and password both *admin*.

After sending the request check port *s1-eth3* in Wireshark (leftmost grey circle button with list inside). **Stop the capture.** Uncheck previous interface and check *s1-eth3*. **Start the capture.** You don't have to save the captured data.

If pings from h1 to h2 are still going you should see ARP packets originating from h1.



Check Mininet. Are pings successful? Think of a reason why. Use Mininet and check the flows to answer the question. The answer is on the next page.

If you will check *openflow:3* switch, you will see, that there's no models associated with it.

---

## Step 8 Adding more flows

**Try to fix the ping** using the same technique as before. At the end, you should have the following flows:

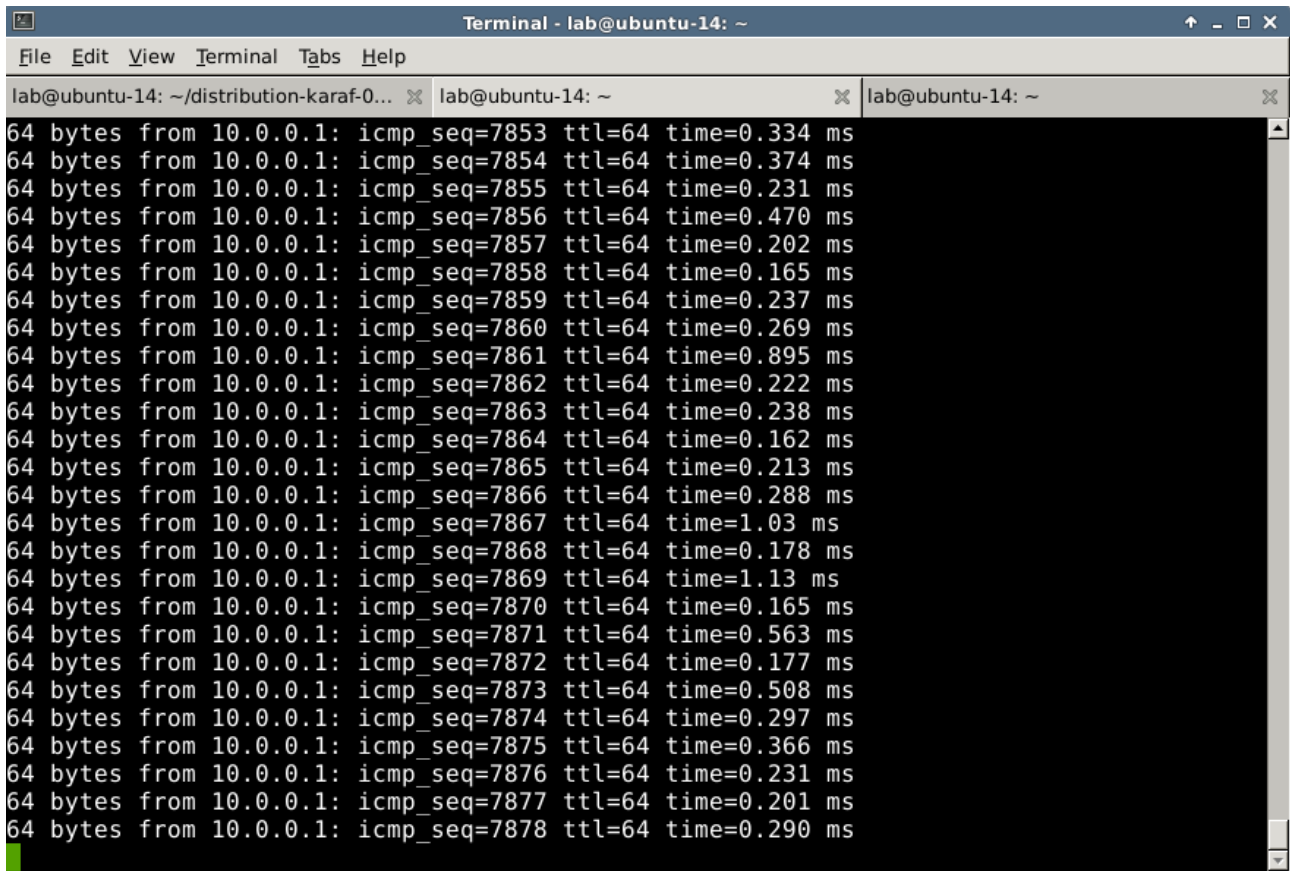
Input	Output
openflow:1:1	3
openflow:3:1	2
openflow:3:2	1
openflow:1:3	1

The first number of the input port name is the number of the device, the second number is the number of the port itself. Use the given example and change:

- device port number in the URL
- flow number in the URL
- match port in the body
- output port in the body

Flows on *openflow:2* are installed by *l2switch* and ping should work even without creating a model for this switch. **Do not add flows on the s2 yet. Don't forget to change the flow id** when you're adding a new one (they are device local).

After applying all flows from end to end, pinging should work.

A terminal window titled "Terminal - lab@ubuntu-14: ~" with a menu bar (File, Edit, View, Terminal, Tabs, Help) and three tabs. The active tab shows the output of a ping command. The output consists of 25 lines, each representing a successful ping response. Each line starts with "64 bytes from 10.0.0.1:" followed by "icmp\_seq=" and a sequence number from 7853 to 7878, "ttl=64", and "time=" followed by a response time in milliseconds. The response times vary, with most between 0.15 and 0.5 ms, and two outliers at 1.03 ms and 1.13 ms. The terminal window has a dark background and a light-colored border.

```
lab@ubuntu-14: ~/distribution-karaf-0... x lab@ubuntu-14: ~ x lab@ubuntu-14: ~ x
64 bytes from 10.0.0.1: icmp_seq=7853 ttl=64 time=0.334 ms
64 bytes from 10.0.0.1: icmp_seq=7854 ttl=64 time=0.374 ms
64 bytes from 10.0.0.1: icmp_seq=7855 ttl=64 time=0.231 ms
64 bytes from 10.0.0.1: icmp_seq=7856 ttl=64 time=0.470 ms
64 bytes from 10.0.0.1: icmp_seq=7857 ttl=64 time=0.202 ms
64 bytes from 10.0.0.1: icmp_seq=7858 ttl=64 time=0.165 ms
64 bytes from 10.0.0.1: icmp_seq=7859 ttl=64 time=0.237 ms
64 bytes from 10.0.0.1: icmp_seq=7860 ttl=64 time=0.269 ms
64 bytes from 10.0.0.1: icmp_seq=7861 ttl=64 time=0.895 ms
64 bytes from 10.0.0.1: icmp_seq=7862 ttl=64 time=0.222 ms
64 bytes from 10.0.0.1: icmp_seq=7863 ttl=64 time=0.238 ms
64 bytes from 10.0.0.1: icmp_seq=7864 ttl=64 time=0.162 ms
64 bytes from 10.0.0.1: icmp_seq=7865 ttl=64 time=0.213 ms
64 bytes from 10.0.0.1: icmp_seq=7866 ttl=64 time=0.288 ms
64 bytes from 10.0.0.1: icmp_seq=7867 ttl=64 time=1.03 ms
64 bytes from 10.0.0.1: icmp_seq=7868 ttl=64 time=0.178 ms
64 bytes from 10.0.0.1: icmp_seq=7869 ttl=64 time=1.13 ms
64 bytes from 10.0.0.1: icmp_seq=7870 ttl=64 time=0.165 ms
64 bytes from 10.0.0.1: icmp_seq=7871 ttl=64 time=0.563 ms
64 bytes from 10.0.0.1: icmp_seq=7872 ttl=64 time=0.177 ms
64 bytes from 10.0.0.1: icmp_seq=7873 ttl=64 time=0.508 ms
64 bytes from 10.0.0.1: icmp_seq=7874 ttl=64 time=0.297 ms
64 bytes from 10.0.0.1: icmp_seq=7875 ttl=64 time=0.366 ms
64 bytes from 10.0.0.1: icmp_seq=7876 ttl=64 time=0.231 ms
64 bytes from 10.0.0.1: icmp_seq=7877 ttl=64 time=0.201 ms
64 bytes from 10.0.0.1: icmp_seq=7878 ttl=64 time=0.290 ms
```

---

## Step 9 Flow priority

Bring the link up and observe how network works.

**In Mininet:**

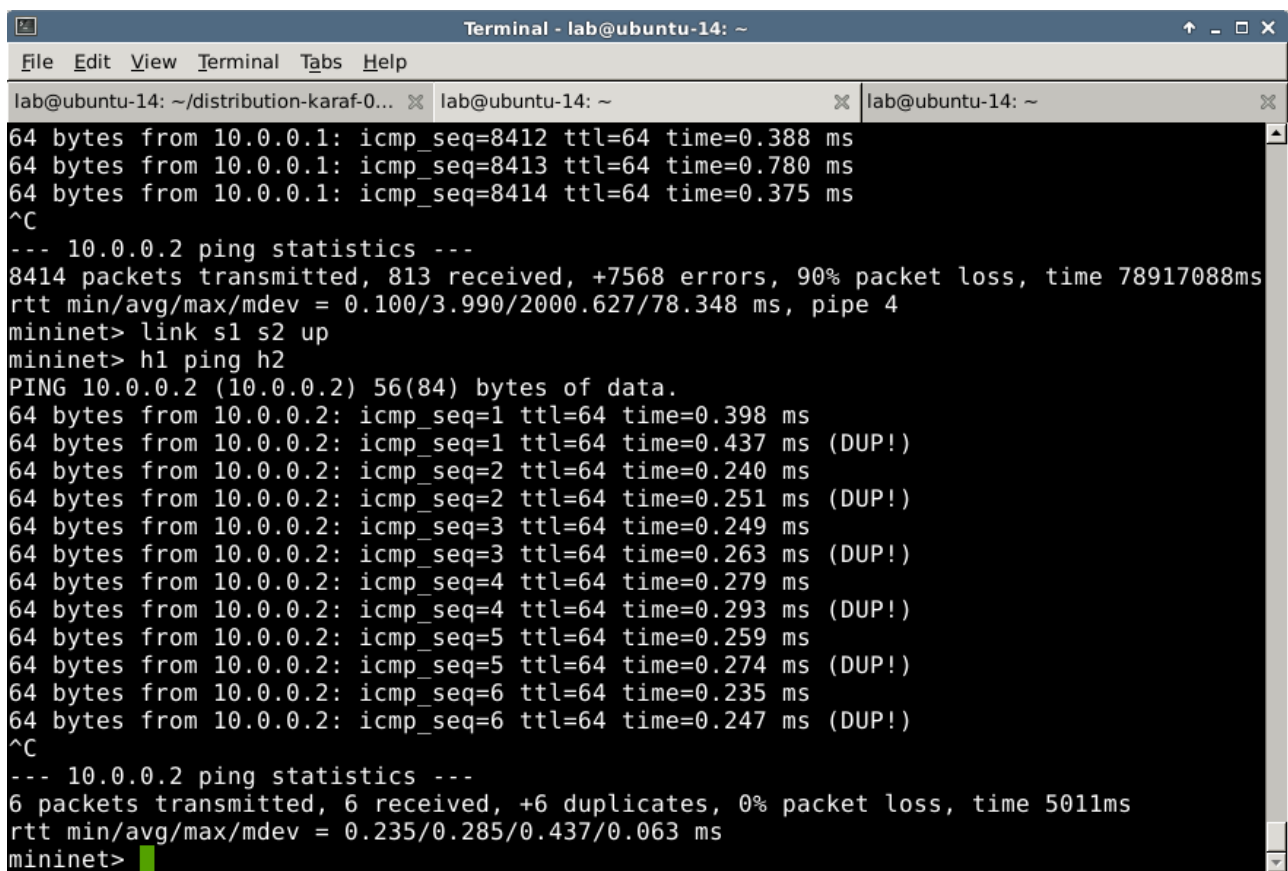
```
link s1 s2 up
```

**Try to ping.**

```
h1 ping h2
```

Following results should be observable.

To see all interfaces in Wireshark now choose *Capture - Refresh Interfaces*.

A terminal window titled "Terminal - lab@ubuntu-14: ~" showing network test results. The output includes ping statistics for 10.0.0.2, showing 8414 packets transmitted, 813 received, and a 90% packet loss. It also shows the execution of 'link s1 s2 up' and 'h1 ping h2', followed by a successful ping to 10.0.0.2 with 6 packets received, 6 duplicates, and 0% packet loss.

```
lab@ubuntu-14: ~ /distribution-karaf-0... x lab@ubuntu-14: ~ x lab@ubuntu-14: ~
64 bytes from 10.0.0.1: icmp_seq=8412 ttl=64 time=0.388 ms
64 bytes from 10.0.0.1: icmp_seq=8413 ttl=64 time=0.780 ms
64 bytes from 10.0.0.1: icmp_seq=8414 ttl=64 time=0.375 ms
^C
--- 10.0.0.2 ping statistics ---
8414 packets transmitted, 813 received, +7568 errors, 90% packet loss, time 78917088ms
rtt min/avg/max/mdev = 0.100/3.990/2000.627/78.348 ms, pipe 4
mininet> link s1 s2 up
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.398 ms
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.437 ms (DUP!)
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.240 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.251 ms (DUP!)
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.249 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.263 ms (DUP!)
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.279 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.293 ms (DUP!)
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.259 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.274 ms (DUP!)
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.235 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.247 ms (DUP!)
^C
--- 10.0.0.2 ping statistics ---
6 packets transmitted, 6 received, +6 duplicates, 0% packet loss, time 5011ms
rtt min/avg/max/mdev = 0.235/0.285/0.437/0.063 ms
mininet>
```

Host h1 receive duplicated responses. Try to answer why. Use Wireshark and **check flows**.



Operational flows on *s2* send packets to both interfaces. They have *priority* value set to 2 by default. The template used in the Postman template has the *priority* value set to 100. To fix the problem complete the flow on the device *s2*. According to OpenFlow switch specification, in case of using wildcard and overlapping flows, switch should only use the flow with the highest priority.

Note: during the experiments *s1* was sending the packets through interface *eth-2* but the packet count didn't change. It is probably the implementation problem.

After redirecting traffic from *s3* to *h2* and vice versa duplicated ping packets should be gone.

## Summary

After finishing this tutorial, you should have understanding how controller is manipulating network using OpenFlow. Some aspects of the OpenFlow are left out, such as multiple table handling. This tutorial gives just a glimpse on capabilities of OpenDayLight controller. Behaviour of *l2switch* feature can be changed, e. g. by default switches are dropping unmatched packets but they can also flood them to all the ports or ingress interface. ARP packets can be sent to a controller or redirected. Ability to choose between *Reactive* and *Proactive* approach is one of the biggest advantages of advantages of SDN. OpenDayLight can use many protocols instead of OpenFlow, such as BGP, NETCONF and PCEP. They give full control of the network.