PCM2.3. Algorithms and applications for utilization of SDN technology to IoT

Prof., Dr. Sc. I.S. Skarga-Bandurova, PhD student A.Yu. Velykzhanin, Dr. L.O. Shumova (V. Dahl EUNU)

Laboratory work 1

APPLICATION OF ONOS SDN CONTROLLER PLATFORM FOR IOT NETWORKS MANAGEMENT

Goal and objectives: This laboratory work is an introduction to the managing a software-defined networks (*SDN*). We'll discover the open network operating system (*ONOS*); set up a network OS, and practice in working with *ONOS*.

Learning objectives:

– to study the principles of the *ONOS*;

– to study the possibilities of managing a software-defined network using ONOS.

Practical tasks:

– acquire practical skills in working with *ONOS*.

Exploring tasks:

– discover *ONOS* communication tools to message exchange in the network;

– investigate how to perform basic management operations with *ONOS*.

Setting up.

In preparation for laboratory work it is necessary:

– to clear the goals and mission of the research;

– to study theoretical material contained in this manual, and in [1]-[3];

– to familiarize oneself with the main procedures and specify the exploration program according to defined task.

Recommended software and resources: *ONOS*, *Mininet*, OracleVM *VirtualBOX*, *SDNHub*.

1.1. Synopsis

In this laboratory work you will learn basic software-defined networking (SDN) concepts using the *ONOS* SDN controller, *ONOS* components, and the *Mininet* network simulator. Specifically you will use *ONOS* SDN controller. While you explored this tool using the Linux operating system, the same tool is available for Windows operating systems.

1.2. Brief theoretical information

IoT applications are fundamentally different from traditional ones. IoT intends to connect billions of devices from different manufacturers that can be deployed in an uncoordinated way. These problems can be solved by implementing a high level of automation of delivery and operation of IoT applications. IoT is the area where SDN can be extremely useful. Below are some issues in IoT and how SDN application can help with this:

– mass device connectivity: Adding routing/forwarding information related to the IoT devices will require the automatic detection of these devices and mechanisms for dynamically calculating the route over the network. The SDN controller can be used to interact with switches in the forwarding planes to configure traffic flows over the network for these devices.

– fast network changes: IoT devices are limited in terms of power consumption and processor utilization. They can often be removed from the network due to low battery or processor overload. They also work on a variety of wireless technologies, which can have a significant failure rate. IoT network infrastructure may need to handle fast changes, which requires changing routing/flow information in the network elements. The SDN can optimally handle such scenarios with dynamic topology maintenance.

– network scalability processing. Recent applications and services are developed based on the principles of NFV, when network objects are created or completed on the fly. In IoT, this can mean pruning and grafting IoT controllers (gateways) as needed. SDN can be used to intelligently locate these controllers and update the streams of connected devices accordingly.

– low power sensors: IoT sensors have the very low processing power and need frequent battery replacement. Therefore, they cannot implement complex routing or network management protocols.

One of the most common SDN controllers for this purpose is *ONOS*. *ONOS* is an SDN network operating system with powerful architectural features such as high availability, scalability, modularity, and complete separation of protocol-independent and protocol-specific device and channel representations.

The *ONOS* (Open Network Operating System) project is an open source community supported by The Linux Foundation. The aim of the project is to create SDN operating system for communications service providers designed to provide scalability, high performance and high availability [1].

In this work we will develop and test an IoT network using *ONOS* SDN Controller with the Mininet Network Emulator. The example of network is shown in Fig. 1.

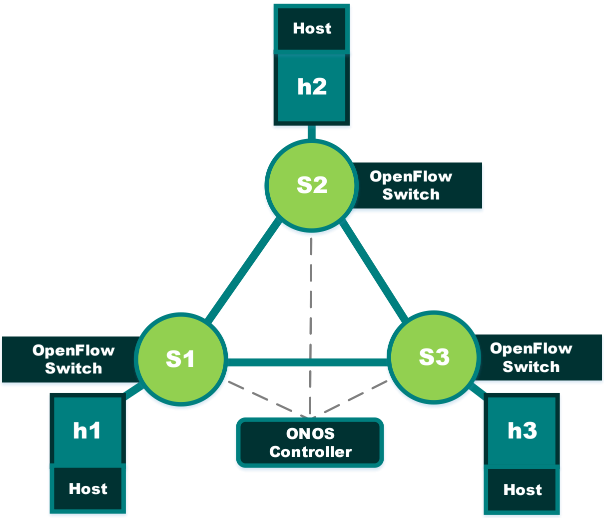


Fig. 1 – An example the network under the test

1.3. Execution order and discovery questions

Step 1. Install *OracleVM VirtualBOX*.

Use link [4] to install OracleVM.

Step 2. Deploy the *SDNHub* image [5].

Step 3. Create a network of switches and hosts.

1. Install *ONOS* [6] and *Mininet Network Emulator* [7].

In this laboratory work we assume you have already set up a *Mininet* VM in VirtualBox.

Start VirtualBox and then start the VM.

We will use the *Mininet* graphical user interface, to set up an emulated network made up of *OpenFlow* switches and Linux hosts. To start *Mininet*, run the following command on a terminal window connected to the *Mininet* VM:

```
> mininet@mininet-vm:~$ sudo ~/mininet/examples/miniedit.py
```

2. Double-click on the downloaded *ONOS* tutorial OVA file will open virtual box with an import dialog. Allocate 2-3 CPUs and 4-8GB of RAM for the VM.

3. Run the *ONOS*. Since we use a ready-made image of the system, almost all the necessary packages have been already installed but you have to conFig. them.

4. Run the command prompt. Then set up a network operating system (Fig. 2).

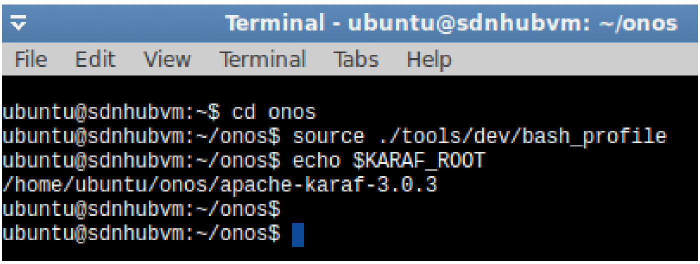


Fig. 2 – Setting the environment variables for *ONOS* and *karaf* execution

Optionally, you can compile the controller using the commands:

```
> mvn clean install -nsu -DskipIT -DskipTests
```

In the new terminal window, write a command:

```
> sudo mn --custom lab1.py --topo mytopo –mac –controller remote
```

Set up a network using standard *Mininet* topology commands. As an alternative for standard command you can use MiniEdit to create the network topology. MiniEdit provides a visual representation of the network. However, while MiniEdit is a good tool for creating topologies for the *Mininet* network simulator, in this lab we create a topology via standard *Mininet* commands.

```
> sudo mn --custom lab1.py --topo mytopo --mac --controller=remote
```

Step 4. Discover *ONOS* Basics.

Before start *ONOS* SDN controller, we need to determine which components we want to run when we start the controller.

1. Start *ONOS* controller with one of the following commands command:

```
> ok clean
> karaf clean
```

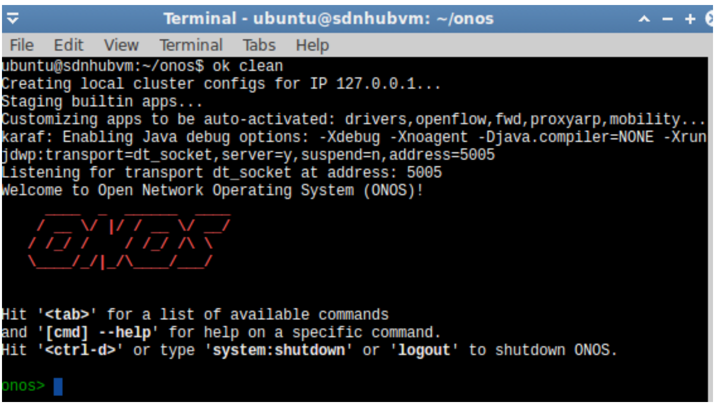


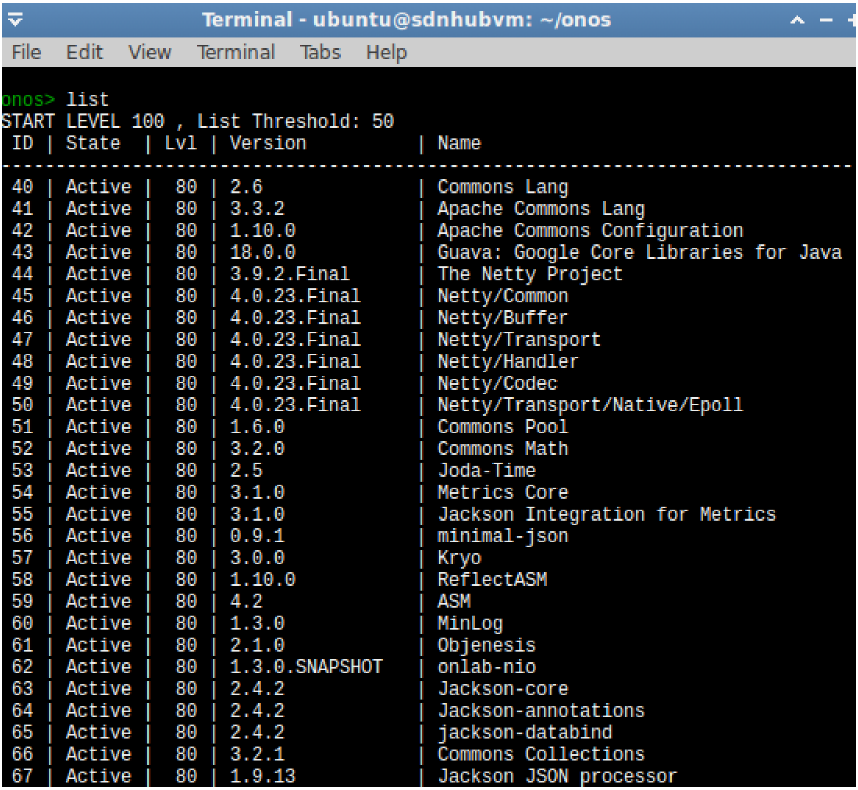
Fig. 3 – Start *ONOS* controller window

2. Look through the main commands. A full list of commands is

available at [8], or use basic *ONOS* tutorial for [9].

2.1. Links. Similarly, the links command is used to list the links detected by *ONOS*.

At the *ONOS* prompt run the command (Fig. 4).

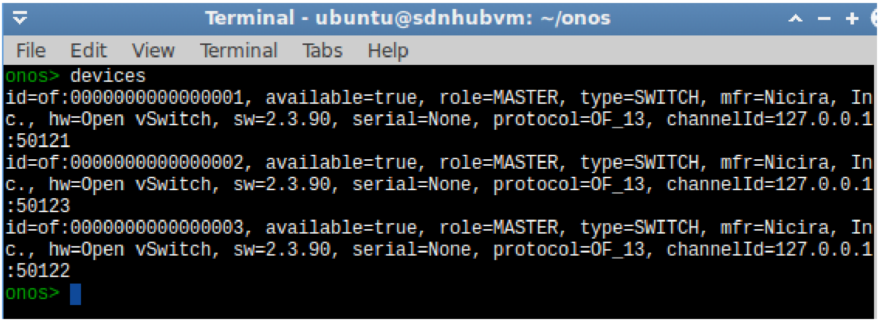


```
Terminal - ubuntu@sdnhubvm: ~/onos
File  Edit  View  Terminal  Tabs  Help

onos> list
START LEVEL 100 , List Threshold: 50
 ID | State    | Lvl | Version            | Name
-----------------------------------------------------------------------------------
 40 | Active  |  80 | 2.6                | Commons Lang
 41 | Active  |  80 | 3.3.2              | Apache Commons Lang
 42 | Active  |  80 | 1.10.0             | Apache Commons Configuration
 43 | Active  |  80 | 18.0.0             | Guava: Google Core Libraries for Java
 44 | Active  |  80 | 3.9.2.Final        | The Netty Project
 45 | Active  |  80 | 4.0.23.Final       | Netty/Common
 46 | Active  |  80 | 4.0.23.Final       | Netty/Buffer
 47 | Active  |  80 | 4.0.23.Final       | Netty/Transport
 48 | Active  |  80 | 4.0.23.Final       | Netty/Handler
 49 | Active  |  80 | 4.0.23.Final       | Netty/Codec
 50 | Active  |  80 | 4.0.23.Final       | Netty/Transport/Native/Epoll
 51 | Active  |  80 | 1.6.0              | Commons Pool
 52 | Active  |  80 | 3.2.0              | Commons Math
 53 | Active  |  80 | 2.5                | Joda-Time
 54 | Active  |  80 | 3.1.0              | Metrics Core
 55 | Active  |  80 | 3.1.0              | Jackson Integration for Metrics
 56 | Active  |  80 | 0.9.1              | minimal-json
 57 | Active  |  80 | 3.0.0              | Kryo
 58 | Active  |  80 | 1.10.0             | ReflectASM
 59 | Active  |  80 | 4.2                | ASM
 60 | Active  |  80 | 1.3.0              | MinLog
 61 | Active  |  80 | 2.1.0              | Objenesis
 62 | Active  |  80 | 1.3.0.SNAPSHOT     | onlab-nio
 63 | Active  |  80 | 2.4.2              | Jackson-core
 64 | Active  |  80 | 2.4.2              | Jackson-annotations
 65 | Active  |  80 | 2.4.2              | jackson-databind
 66 | Active  |  80 | 3.2.1              | Commons Collections
 67 | Active  |  80 | 1.9.13             | Jackson JSON processor
```

Fig. 4 – List of links detected by *ONOS*

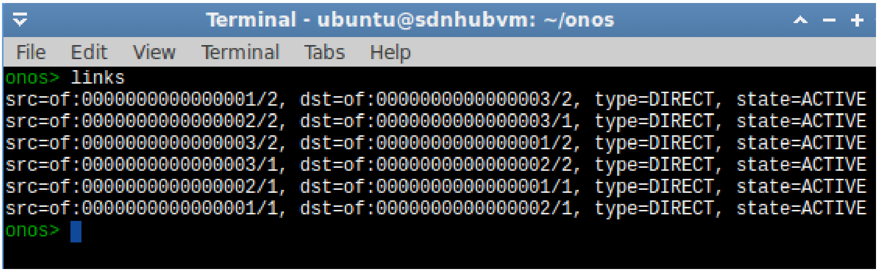
With the help of the "*devices*" command, find out the list of all infrastructure devices (Fig. 5).



```
Terminal - ubuntu@sdnhubvm: ~/onos
File Edit View Terminal Tabs Help
onos> devices
id=of:0000000000000001, available=true, role=MASTER, type=SWITCH, mfr=Nicira, In
c., hw=Open vSwitch, sw=2.3.90, serial=None, protocol=OF_13, channelId=127.0.0.1
:50121
id=of:0000000000000002, available=true, role=MASTER, type=SWITCH, mfr=Nicira, In
c., hw=Open vSwitch, sw=2.3.90, serial=None, protocol=OF_13, channelId=127.0.0.1
:50123
id=of:0000000000000003, available=true, role=MASTER, type=SWITCH, mfr=Nicira, In
c., hw=Open vSwitch, sw=2.3.90, serial=None, protocol=OF_13, channelId=127.0.0.1
:50122
onos>
```

Fig. 5 – The list of infrastructure devices

With the help of the "*links*" command, we can find out the list of all infrastructure links (Fig. 6).



```
Terminal - ubuntu@sdnhubvm: ~/onos
File Edit View Terminal Tabs Help
onos> links
src=of:0000000000000001/2, dst=of:0000000000000003/2, type=DIRECT, state=ACTIVE
src=of:0000000000000002/2, dst=of:0000000000000003/1, type=DIRECT, state=ACTIVE
src=of:0000000000000003/2, dst=of:0000000000000001/2, type=DIRECT, state=ACTIVE
src=of:0000000000000003/1, dst=of:0000000000000002/2, type=DIRECT, state=ACTIVE
src=of:0000000000000002/1, dst=of:0000000000000001/1, type=DIRECT, state=ACTIVE
src=of:0000000000000001/1, dst=of:0000000000000002/1, type=DIRECT, state=ACTIVE
onos>
```

Fig. 6 – The list of infrastructure links

2.2. Flows list all currently-known flows.

Now all switches have 5 routing rules. Here, deviceID is a device identifier (switch) and id is a routing rule (Fig. 7).

2.3. Run the *Mininet pingall* command.

This command runs ping tests between each host in the emulated network. This generates traffic to the controller every time a switch receives a packet that has a destination MAC address that is not already in its flow table.

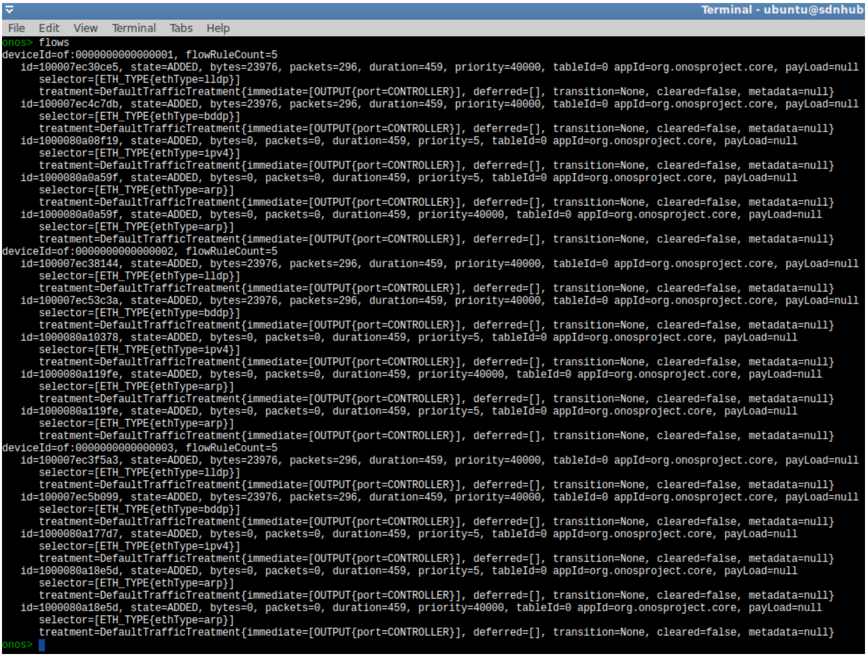


Fig. 7 – The list of flows

From the host h3 send the command "*ping*" to the host h1 in the *Mininet* window (Fig. 8).

To see the contents of the flow tables on all switches, execute the *Mininet* command:

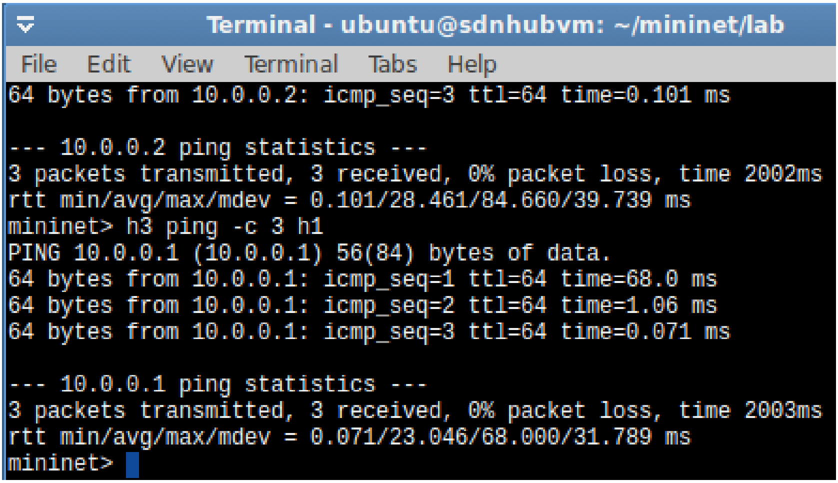
```
mininet> dpctl dump-flows
```

To check ARP tables on each host, execute the *Mininet* "*arp*" command. For instance, to show the ARP table for host h1, enter the following command:

```
mininet> h1 arp
```

To clear all flow tables on all switches, enter the *Mininet* command:

```
mininet> dpctl del-flows
```



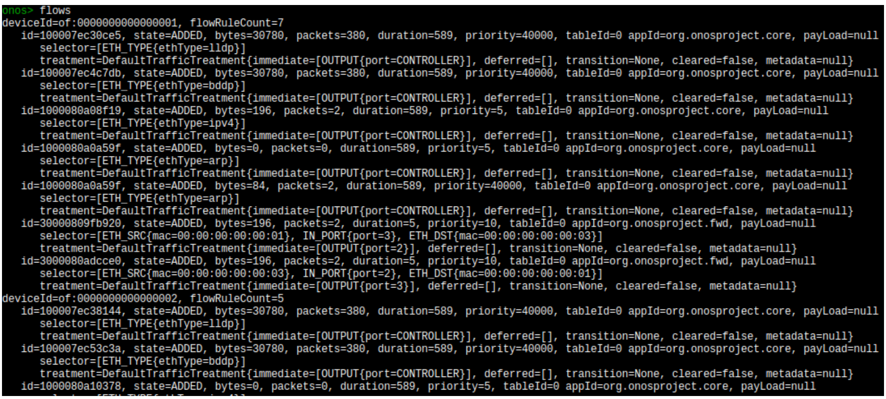
```
Terminal - ubuntu@sdnhubvm: ~/mininet/lab
File  Edit  View  Terminal  Tabs  Help
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.101 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 0.101/28.461/84.660/39.739 ms
mininet> h3 ping -c 3 h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=68.0 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=1.06 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.071 ms

--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 0.071/23.046/68.000/31.789 ms
mininet>
```

Fig. 8 – *Mininet* "*ping*" command

Analyze how the rules change. On switches 1 and 3, two new routing rules should appear (Fig. 9).



```
onos> flows
deviceId=of:0000000000000001, flowRuleCount=7
   id=100007ec30ce5, state=ADDED, bytes=30780, packets=380, duration=589, priority=40000, tableId=0 appId=org.onosproject.core, payLoad=null
      selector=[ETH_TYPE{ethType=lldp}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=CONTROLLER}], deferred=[], transition=None, cleared=false, metadata=null}
   id=100007ec4c7db, state=ADDED, bytes=30780, packets=380, duration=589, priority=40000, tableId=0 appId=org.onosproject.core, payLoad=null
      selector=[ETH_TYPE{ethType=bddp}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=CONTROLLER}], deferred=[], transition=None, cleared=false, metadata=null}
   id=1000080a08f19, state=ADDED, bytes=196, packets=2, duration=589, priority=5, tableId=0 appId=org.onosproject.core, payLoad=null
      selector=[ETH_TYPE{ethType=ipv4}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=CONTROLLER}], deferred=[], transition=None, cleared=false, metadata=null}
   id=1000080a0a59f, state=ADDED, bytes=0, packets=0, duration=589, priority=5, tableId=0 appId=org.onosproject.core, payLoad=null
      selector=[ETH_TYPE{ethType=arp}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=CONTROLLER}], deferred=[], transition=None, cleared=false, metadata=null}
   id=1000080a0a59f, state=ADDED, bytes=84, packets=2, duration=589, priority=40000, tableId=0 appId=org.onosproject.core, payLoad=null
      selector=[ETH_TYPE{ethType=arp}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=CONTROLLER}], deferred=[], transition=None, cleared=false, metadata=null}
   id=30000809fb920, state=ADDED, bytes=196, packets=2, duration=5, priority=10, tableId=0 appId=org.onosproject.fwd, payLoad=null
      selector=[ETH_SRC{mac=00:00:00:00:00:01}, IN_PORT{port=3}, ETH_DST{mac=00:00:00:00:00:03}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=2}], deferred=[], transition=None, cleared=false, metadata=null}
   id=3000080adcce0, state=ADDED, bytes=196, packets=2, duration=5, priority=10, tableId=0 appId=org.onosproject.fwd, payLoad=null
      selector=[ETH_SRC{mac=00:00:00:00:00:03}, IN_PORT{port=2}, ETH_DST{mac=00:00:00:00:00:01}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=3}], deferred=[], transition=None, cleared=false, metadata=null}
deviceId=of:0000000000000002, flowRuleCount=5
   id=100007ec38144, state=ADDED, bytes=30780, packets=380, duration=589, priority=40000, tableId=0 appId=org.onosproject.core, payLoad=null
      selector=[ETH_TYPE{ethType=lldp}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=CONTROLLER}], deferred=[], transition=None, cleared=false, metadata=null}
   id=100007ec53c3a, state=ADDED, bytes=30780, packets=380, duration=589, priority=40000, tableId=0 appId=org.onosproject.core, payLoad=null
      selector=[ETH_TYPE{ethType=bddp}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=CONTROLLER}], deferred=[], transition=None, cleared=false, metadata=null}
   id=1000080a10378, state=ADDED, bytes=0, packets=0, duration=589, priority=5, tableId=0 appId=org.onosproject.core, payLoad=null
```

Fig. 9 – The list of routing rules

Step 5. Build a network in *Mininet* in accordance with your personal task and discover basic network management operations with *ONOS*.

An example below creates a 3-switch topology connected in a loop. A host is connected to each switch.

```
#!/usr/bin/python
from mininet.topo import Topo
class triangleTopo( Topo ):
  "Create a custom network and add nodes to it."
  def __init__( self ):
    #setLogLevel( 'info' )
    # Initialize topology
    Topo.__init__(self)
    #info( '*** Adding hosts\n' )
    h1 = self.addHost( 'h1' )
    h2 = self.addHost( 'h2' )
    h3 = self.addHost( 'h3' )
    #info( '*** Adding switches\n' )
    nodeA = self.addSwitch('s1')
    nodeB = self.addSwitch('s2')
    nodeC = self.addSwitch('s3')
    #info( '*** Creating links\n' )
    self.addLink( nodeA, nodeB )
    self.addLink( nodeB, nodeC )
    self.addLink( nodeC, nodeA )
    self.addLink( h1, nodeA )
    self.addLink( h2, nodeB )
    self.addLink( h3, nodeC )
topos = {'mytopo': (lambda: triangleTopo() ) }
```

Explore *OpenFlow* control messages and how flow tables are updated on the switches.

Explore how the other stock *ONOS* components work individually and in combination with other components or applications.

1. *ONOS* has a web-based GUI. To launch it you should click on the provided *ONOS* GUI icon (Fig. 10).

Fig. 10 – *ONOS* GUI icon

2. Login as user `onos` with password `rocks`;
3. Open the browser and go to the following link:

```
http://localhost:8181/onos/ui/index.html
```

4. Click “h”. We will see our hosts. As you will see, host two are invisible to us due to we did not ping it.

5. Ping the h2 host in the *Mininet* window (Fig. 11).

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

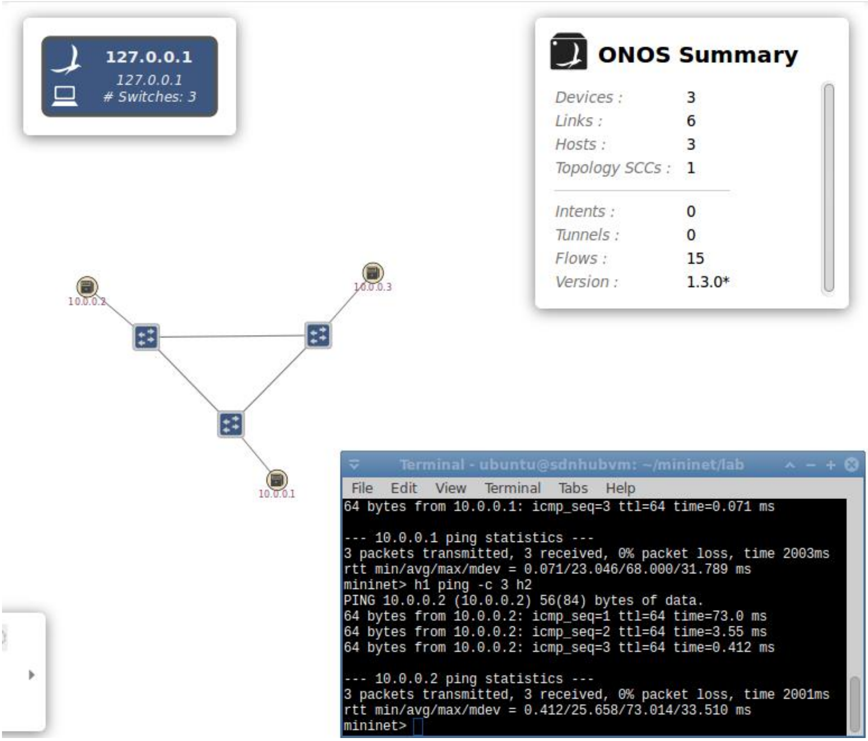


Fig. 11 – Simple tree with three switches and three hosts

For quick help press “\”.

6. On *ONOS* GUI press “F” and click to “s1” for seeing traffic flow (yellow line in Fig. 12).

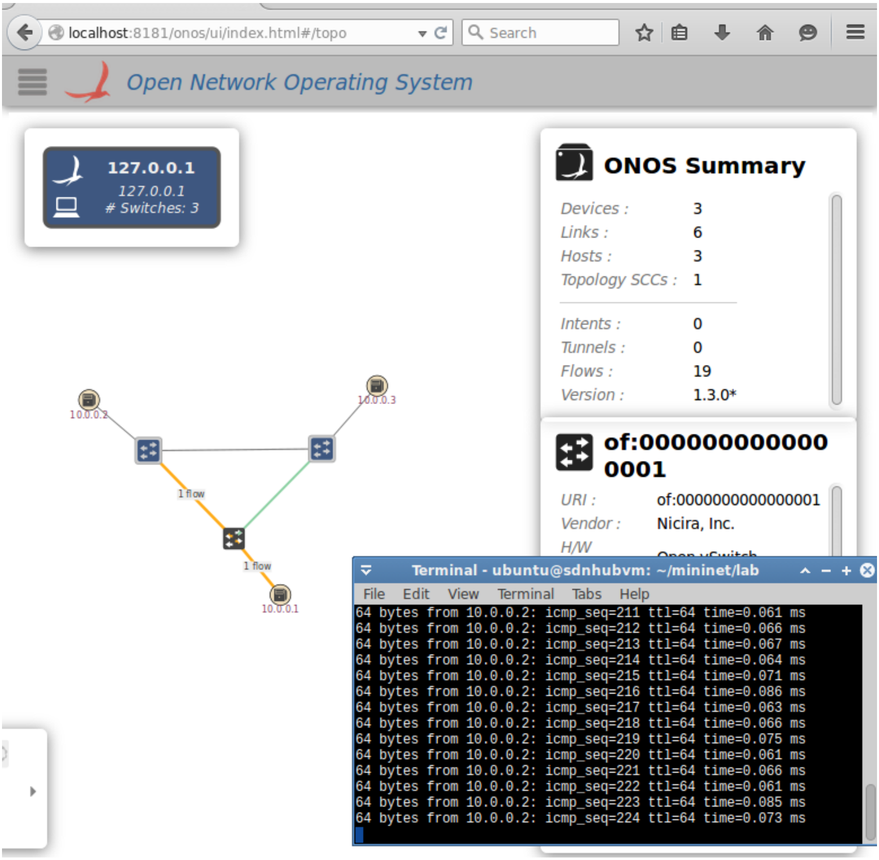


Fig. 12 – Traffic flow

In this lab, the *ONOS* controller is running on the same virtual machine that all the emulated switches and hosts created by *Mininet* are running on.

7. Run “h1 ping h3”.

Ctrl-c is an interrupt hotkey.

On *ONOS* command line type “stop onos-app-fwd”.

As it can be seen from Fig. 13, the packet forwarding has been stopped.

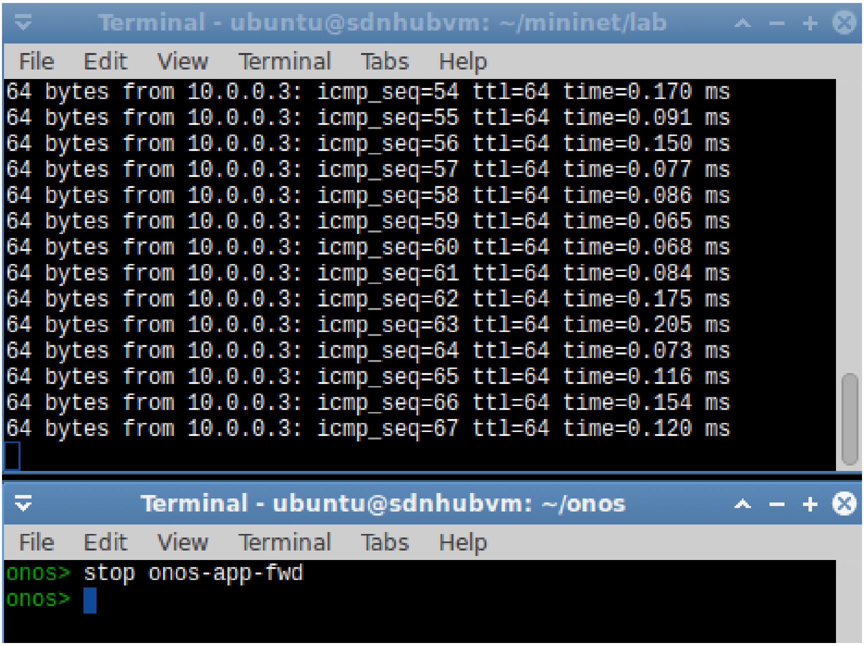


Fig. 13 – Stopping the packet forwarding

8. Restore packet flow (Fig. 14).

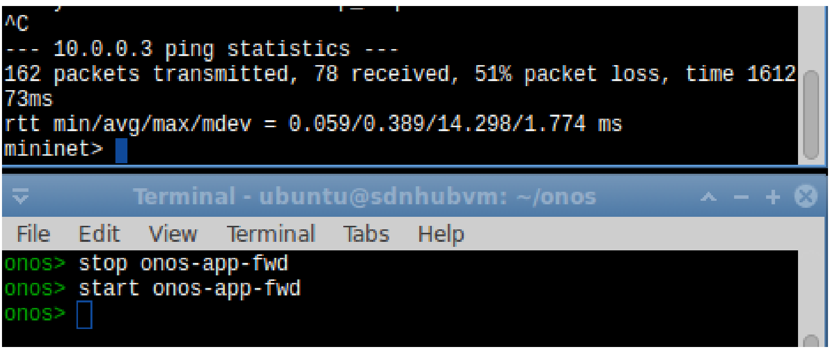


Fig. 14 – Command to restore packet flow

The command “logout” is used for *ONOS* stopping.

1.4. Requirements to the content of the report

Report should contain 5 sections: Introduction (I), Methods (M), Results (R), and Discussion (D)

– (I): background / theory, purpose and discovery questions;

– (M): complete description of the software, and procedures which was followed in the experiment, experiment overview, Fig. / scheme of testing environment, procedures;

– (R): narrate (like a story), tables, indicate final results;

– (D): answers on discovery questions, explanation of changes in traffic flow, conclusion / summary.

1.5. Control questions:

1. For what purpose *ONOS* is used?
2. List main command options available for *ONOS.*
3. How the rules changed when the host "h3" send the command "ping" to the host "h1"? Why?
4. What changes did you observe at your virtual network?
5. How to build a network in mininet?
6. What are the network management operations with *ONOS*?

1.6. Recommended literature:

1. Onosprojectorg. 2018. [Online]. Available: http://onosproject.org/wp-content/uploads/2014/11/Whitepaper-ONOS-final.pdf. [Accessed: 4 Feb. 2018].

2. “Basic ONOS Tutorial” *Wikionosprojectorg*. 2018. [Online]. Available: https://wiki.onosproject.org/display/ONOS/Basic+ONOS+Tutorial. [Accessed: 4 Feb. 2018].

3. Anadiotis, A.-C. G., Galluccio, L., Milardo, S., Morabito, G. and Palazzo, S., 2015, Towards a software-defined Network Operating System for the IoT. 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT). 2015. doi 10.1109/wf-iot.2015.7389118. IEEE.

4. Oracle VM VirtualBox. 2018. [Online]. Available: https://www.oracle.com/technetwork/server-storage/virtualbox/overview/index.html [Accessed: 4 Feb. 2018].

5. SDN Tutorial. 2018. [Online]. Available: http://yuba.stanford.edu/~srini/tutorial/SDN_tutorial_VM_32bit.ova. [Accessed: 4 Feb. 2018].

6. ONOS is the only open source controller providing: [Online]. Available: https://onosproject.org/. [Accessed: 4 Feb. 2018].

7. Mininet: An Instant Virtual Network on your Laptop (or other PC) [Online]. Available: http://mininet.org/. [Accessed: 4 Feb. 2018].

8. Appendix A: CLI commands [Online]. Available: https://wiki.onosproject.org/display/ONOS/Appendix+A+%3A+CLI+commands. [Accessed: 4 Feb. 2018].

9. Basic ONOS Tutorial [Online]. Available: https://wiki.onosproject.org/display/ONOS/Basic+ONOS+Tutorial. [Accessed: 4 Feb. 2018].