PCM2.1. Software defined networks basics

Assoc. Prof., Dr. V.V. Shkarupylo, Assoc. Prof., Dr. R.K. Kudermetov, MSc student D.S. Mazur (ZNTU)

Laboratory work 1

INSTALLATION AND CONFIGURATION OF MININET ENVIRONMENT

Goal: get familiar with technical aspects of Software defined network functioning and usage; obtain practical skills in sphere of Mininet emulator installation, configuration and utilization.

Laboratory work participants: lecturers, scientists, technical staff, students and post-graduate students of the department (faculty, institute) of the university; developers, engineers, trainees.

1.1. Theoretical information

Current level of global networking evolution can be characterized as follows: there is a vital need for such system control, monitoring and configuring aspects improvement. The solution can be found with Software Defined Networking (SDN) principles in mind. Here is the brief list [1]: differentiation between control and date planes [2], unified switches usage to maintain data forwarding, utilization of controller – to coordinate such switches in a centralized manner, stick to OpenFlow protocol [3, 4]. OpenFlow specification provides open interface for SDN-network components communication. The key components are controller, switches and hosts.

Because of the fact that SDN technology is relatively new, it is commonly relatively difficult to work with such network directly. The solution can be in different emulators usage. The emulator typically is a set of software and hardware means to represent SDN network within virtual environment. SDN software is based on Linux platform. Here are some examples of such emulators: Mininet [5, 6], EstiNet [7], OpenNet [8], ns-3 [9]. Each of these solutions has its advantages and drawbacks. The Mininet emulator though is being frequently considered to be an exemplar to be compared to. That's why this solution will be used in our laboratory works.

Mininet environment is devoted to be the mean for SDN-network emulation, particularly by creating virtual hosts, switches, controllers and connections between these components. Named components and connections between them form the topology of network.

The architecture of SDN is given in Fig. 1 [10].

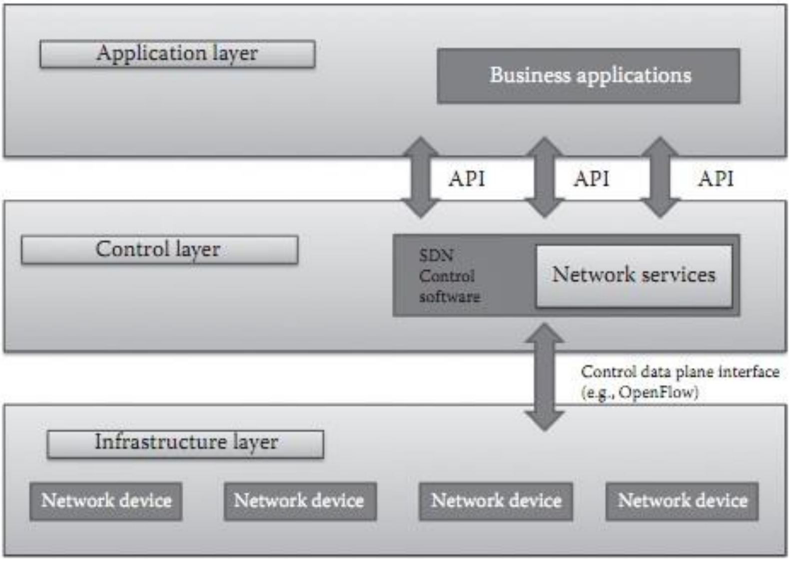

Fig. 1 – Layered architecture of SDN

Mininet environment provides the means to conduct the development, investigation, testing and software configuring of SDN systems, etc.

Mininet provides in particular the following abilities:

– can be used as testbed for SDN applications development;

– brings to the table the ability of different developers to jointly work on network topology;

– includes the means of complex topology testing;

– provides specialized Application Programming Interface (API), oriented on Python programming language usage;

Comparing to typical approaches to virtualization, Mininet provides the following advantages:

– easiness of installation;

– quick boot time;

– easiness of system reconfiguration.

As a drawback the difficulties during the work with graphical environment of Mininet on Windows platform and also the limitation of network configuration by hardware resources available for virtual machine can be pointed out [6].

The tasks to be accomplished during the laboratory work:

– Mininet Linux-environment installation on Windows platform by way of VirtualBox usage;

– virtual machine network interfaces configuration;

– get in touch with basic console commands of Mininet emulator, particularly to create the networks with different topologies.

The presentation of accomplished tasks has to be conducted by one of two ways:

– one-by-one;

– after all the tasks have been accomplished.

Obtained results have to be properly represented in the report to be defended.

1.2. Example of work execution

Step 1. Installation of Mininet environment.

To install Mininet emulator the following software has to be used:

– VirtualBox-4.3.10 has been chosen to be a tool for software virtualization. File VirtualBox-4.3.10-93012-Win.exe is intended to be installed on 32-bit Windows-platform;

– mininet-2.2.1-150420-ubuntu-14.04-server-i386.zip archive has been chosen as Mininet emulator build. This archive has to be unpacked. It can be seen from file name that the content of archive is the emulation of 32-bit ubuntu-14.04-server Linux environment.

After installation of VirtualBox-4.3.10 tool the content of archive – mininet-vm-i386.vmdk file – then has to be used as a hard drive of virtual system to be created.

The snapshot of successfully created virtual machine is given in Fig. 2.

It has to be noted that for the needs of virtual machine it has to be allocated not less than 512 MB of random access memory.

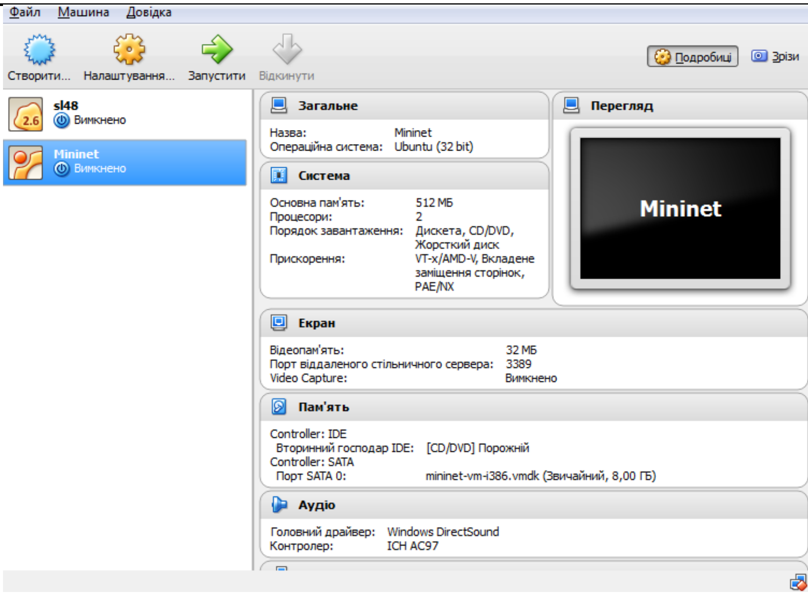

Fig. 2 – The information about virtual machine configuration

Step 2. Configuration of network interfaces.

To configure network interfaces of virtual machine the option "Settings" has to be chosen first, then "Adapter 1" and "Adapter 2" configurations have to be accessed via "Network" option (Fig. 3, Fig. 4).

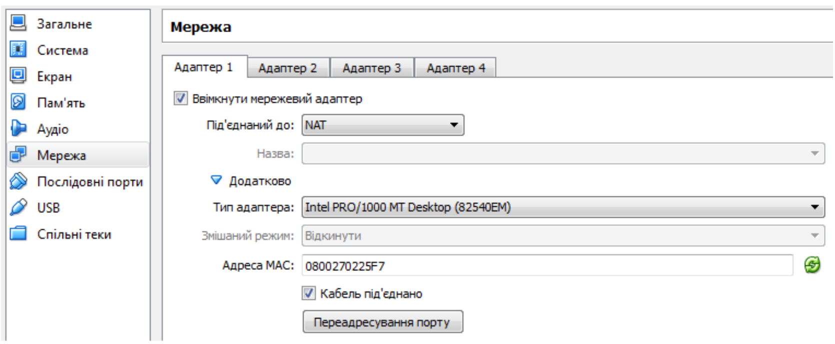

Fig. 3 – "Adapter 1" network configuration

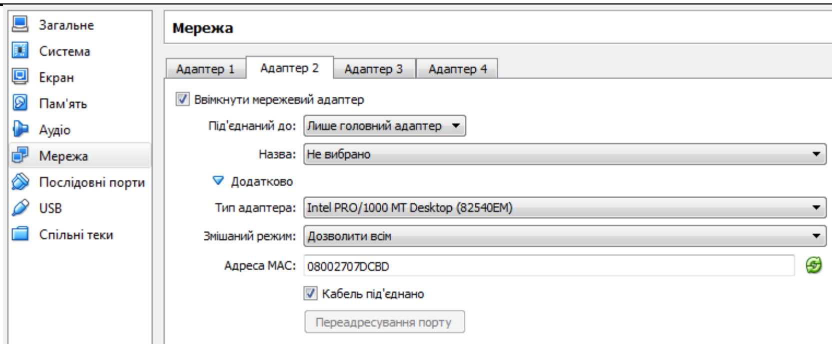

Fig. 4 – "Adapter 2" network configuration

In order to create your own information model, you first need to create a project in which you will work in the future.

Step 3. Mininet usage basics.

Launch virtual machine by pushing the "*Launch*" button.

After a while there will be proposed to enter the login and password in console:

– in the mininet-vm login: field the mininet login has to be entered;

– in the password: field the mininet password has to be entered.

As confirmation of success the following line is going to appear:

```
mininet@mininet-vm:~$
```

1. View the information about network interfaces configuration.

For this purpose the following command has to be executed:

```
> sudo ifconfig
```

As a result, we can check, for instance, the IP-address of *eth0* interface. It's going to be *10.0.2.15* or something like that.

2. Create network topology with minimal configuration – single controller (*c0*), single switch (*s1*) and pair of hosts (*h1*, *h2*):

```
> sudo mn
```

After command execution the information about newly created network with minimal topology will be given, and then the Mininet console will be provided.

3. View the information about Mininet commands:

```
> help
```

4. View the information about all network nodes (there are should be four nodes in total – controller (*c0*), switch (*s1*) and pair of hosts (*h1*, *h2*)):

```
> nodes
```

5. Check the connections between nodes:

```
> links
```

6. Get the information about the IP-address and corresponding subnetwork mask for a particular interface of certain host. For instance, for *eth0* interface of *h1* host, the command will be as follows:

```
> h1 ip addr show | grep eth0
```

7. Test the throughput of communication channel between the specified hosts. For instance, with respect to *h0* and *h1* hosts, the command should be as follows:

```
> iperf h1 h2
```

It takes a while to accomplish the command. Each new command execution will provide slightly different result.

8. View the information about nodes' interfaces:

```
> net
```

9. View the information about nodes configuration:

```
> dump
```

10. View the information about network interfaces of specified node. For instance, for *h1* node the following command should be executed:

```
> h1 ifconfig -a
```

11. Check the information about the processes executed on nodes. For instance, for *s1* node the following command should be executed:

```
> s1 ps -a
```

12. Check the connections between hosts.

Between specified hosts:

```
> h1 ping -c 1 h2
```

Between all pairs of hosts:

```
> pingall
```

13. Launch web server and appropriate client on *h1* and *h2* hosts respectively:

```
> h1 python -m SimpleHTTPServer 80 &
> h2 wget -O - h1
```

As a result of command execution, the HTML-code of web-page, obtained by client, will be shown in the console.

14. Finalize the functioning of web server:

```
> h1 kill %python
```

15. Exit from Mininet console environment back to Linux console:

```
> exit
```

16. Clear topology-related data:

```
> sudo mn -c
```

As a result, all topology-related configuration data will be removed.

1.3. Tasks for individual execution

1. Create Software-defined network with specified parameters.

Solve the task with respect to a given variant. The topology should be created with specified *bandwidth* and *delay* parameters.

Variant 1. Throughput – 10 Mb/s; communication delay: 20 ms:

```
> sudo mn --link tc,bw=10,delay=20ms
```

Variant 2. Throughput – 100 Mb/s; communication delay: 40 ms:

```
> sudo mn --link tc,bw=100,delay=40ms
```

For each variant the following should be done:

– measure the bandwidth of communication channel between hosts 5 times. For this purpose, the *iperf* command should be used. The average should be placed to report;

– find the minimal value of *rtt* (round trip time) parameter with *ping* command.

Remarks:

– *rtt* parameter encompass the time, spent on package transfer from source host to destination host, plus the time on package retrieval notification;

– for each variant, the student should be able to explain the obtained results, e.g., why measured values of bandwidth are lower than specified value of *bw* parameter, minimal value of *rtt* parameter is about 4 times above the specified *delay* value.

2. Create the network with linear topology, encompassing 3 hosts:

```
> sudo mn --test pingall --topo single,3
```

3. Redo step 3 with respect to newly created topology.

4. Create linear topology with four switches and four hosts:

```
> sudo mn --test pingall --topo linear,4
```

This topology will include seven connections.

5. Create a tree-topology network with minimal configuration (one controller, one switch and pair of hosts) and test it with *pingall* command:

```
> sudo mn --topo tree,depth=1,fanout=2 --test pingall
```

The *--topo tree* parameter sets tree topology itself. The depth attribute sets the amount of switch layers (one layer in our case, represented with single element (top) of switches tree): on the potential second layer there will be a pair of switches, on the third – four, and so on. The *fanout* attribute defines the number of connections to each switch. In our case *fanout=2*. This means that, taking into consideration that *depth=1* (there are no other layers with switches and there are no other switches at all), both connections are the direct connections to hosts.

For instance, if we had *depth=2*, there would be one switch from the first layer connected to a pair of switches from the second layer, and those switches from the second layer would be directly connected to a pair of hosts each. That means that there would be three switches and four hosts in total.

The *--test pingall* parameter means that, after creation of network with specified topology, each host should ping all other hosts to test network consistency.

The procedure of such network creation and testing is a time consuming process which will take place about 5 sec and will be shown in console log.

6. Experiment on networks creation and testing with different values of depth and fanout parameters.

<u>1.4. Report content</u>

The report should contain:

– title page with the name of the laboratory work;
– aim of the work; problem statement according to the task;
– work progress and the results of tasks execution;
– analysis of the results and conclusions;
– brief answers to the control questions.

1.5. Control questions:

1. Software Defined Networking. The aim, advantages and drawbacks.
2. Give the definition of 'emulation' notion. Name the examples of SDN emulators.
3. Mininet emulator. Usage and peculiarities.
4. Mininet installation and configuration. Brief description of steps performed.
5. Stages of SDN network with minimal topology creation (by default) – one switch and pair of hosts.
6. Commands to communicate with hosts and switches.
7. Commands to check connections between hosts.
8. Commands to launch web server and appropriate client.
9. Commands to set the delays on communicational channels.
10. Commands to change network configuration.
11. The use of depth and fanout parameters during the creation of network with tree topology. Characterize the impact of these parameters values on total number of network nodes.

1.6. Recommended literature:

1. N. Feamster, J. Rexford, and E. Zegura, “The road to SDN: an intellectual history of programmable networks,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014.
2. T. D. Nadeau and K. Gray, *SDN: Software Defined Networks*. Sebastopol, CA : O’Reilly Media, 2013, 384 p.
3. OpenFlow Switch Specification, March 26, 2015 [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf. [Accessed: 8 Jun. 2019].
4. P. Goransson, C. Black, and T. Culver, *Software Defined Networks: A Comprehensive Approach*, 2nd ed. Waltham, MA: Elsevier, 2016, 436 p.
5. Mininet: An Instant Virtual Network on your Laptop (or other PC) [Online]. Available: http://mininet.org. [Accessed: 8 Jun. 2019].

6. F. Keti and S. Askar, “Emulation of Software Defined Networks Using Mininet in Different Simulation Environments,” in *2015 6th International Conference on Intelligent Systems, Modelling and Simulation*, Kuala Lumpur, Malaysia, 9-12 February 2015, pp. 205-210.

7. S-Y. Wang, “Comparison of SDN OpenFlow network simulator and emulators: EstiNet vs. Mininet,” in *2014 IEEE Symposium on Computers and Communication*, Funchal, Portugal, 23-26 Jun. 2014.

8. M-C. Chan et al., “A simulator for software-defined wireless local area network,” in *2014 IEEE Wireless Communications and Networking Conference*, Istanbul, Turkey, 6-9 April 2014.

9. J. Ivey, H. Yang, C. Zhang and G. Riley, “Comparing a Scalable SDN Simulation Framework Built on ns-3 and DCE with Existing SDN Simulators and Emulators,” in *2016 annual ACM Conference on SIGSIM Principles of Advanced Discrete Simulation*, Banff, Alberta, Canada, 15-18 May 2016, pp. 153-164.

10. F. Hu, *Network Innovation through OpenFlow and SDN: Principles and Design*. Boca Raton, FL : CRC Press, 2014, 520 p.