# ЕКСПЕРИМЕНТАЛНА ПОСТАНОВКА ЗА ТЕСТВАНЕ НА КОНТРОЛЕР ЗА СОФТУЕРНО ДЕФИНИРАНИ МРЕЖИ

**Цветан Райков, Делян Генков*, Мирослав Славов**

*Технически университет - Габрово, ул. Хаджи Димитър 4, Габрово, България*

**кореспондиращ автор: dgenkov@tugab.bg*

# EXPERIMENTAL SETUP FOR TESTING OF SOFTWARE DEFINED NETWORKS CONTROLLER

**Tsvetan Raykov, Delyan Genkov*, Miroslav Slavov**

*Technical University - Gabrovo, 4, Hadji Dimitar Str., Gabrovo, Bulgaria;*

**Corresponding author: dgenkov@tugab.bg*

**Abstract**

*Software defined networking is an important development of the traditional networks. In this concept, the networking devices communicate with a software, named controller, in order to receive their configuration and to report their status. There are plenty of free and commercial controllers. In this paper, we provide the results from a simulation tests between networking devices and controller.*

**Keywords:** Software Device Network; SDN; Controller; Mininet; pox.

## ВЪВЕДЕНИЕ

Софтуерно дефинираните мрежи са важно развитие на традиционните компютърни мрежи. Те намират все по-широко разпространение в съвременния свят на информационни и комуникационни технологии. Въпреки различните разбирания и реализации на много производители, принципът на работата им се основава на разделянето на управляващата логика, наречена „контролна равнина“ (control plane) и логиката за препращане на данни, наричан „равнина за данни“ (data plane).

Равнината за данни включва хардуерните и софтуерните компоненти, които извършват комутация, маршрутизация, филтриране и понякога криптиране на пакети. Равнината на данните работи според указанията от контролната равнина и се занимава с предаването на трафик в мрежата на ниво пакет.

Контролната равнина е "мозъка" на мрежовите устройства. Тя отговаря за вземането на решения относно управлението на трафика. Тя генерира маршрутизиращи таблици, таблици на съседство и правила за препращане, които равнината за данни ще следва. При софтуерно дефинираните мрежи, управлението на мрежовите устройства е отделено от тях и е поставена в централизирано устройство наречено - SDN контролер. Така, вместо всяко устройство да е управлявано от своята управляваща равнина, те се управляват централизирано от контролера. Това позволява централизиране на управлението, централизиране на политиките и улеснява менажирането на устройствата [1].

На практика в софтуерно дефинираните мрежи обикновено се реализира и трети слой - управляваща равнина (Management plane), която взаимодейства както с контролната, така и с равнината за данни. Тя се фокусира върху предоставянето на интерфейси за ръчно или автоматизирано управление на устрой-

ствата. На практика тя предоставя на администраторите на мрежите инструментите, необходими да конфигурират устройствата, да следят производителността на мрежата, както и да отстраняват мрежови проблеми. Тази равнина не е стандартизирана и не е предмет на настоящото изследване.

## ИЗЛОЖЕНИЕ

Основните компоненти на софтуерно дефинираните мрежи са крайните устройства (комутатори, защитни стени, маршрутизатори), контролер и управляващо приложение. Конкретните изследвания на екипа за настоящия доклад са насочени към първите два компонента. Комуникацията между устройствата и контролерите се извършва чрез протокол, наричан южен програмен интерфейс (Southbound API). Примери за такива протоколи са OpenFlow [2], NETCONF [3] и други.

От гледна точка на съвместимост и наличие на отворен код, който може да бъде модифициран при нужда, нашият опит препоръчва използването на OpenFlow. Към момента на написване на настоящия документ са налични версии на OpenFlow 1.0, 1.1, 1.2, 1.3, 1.4 и 1.5. През 2016 година е разработен OpenFlow 1.6, но той е достъпен само за платени членове на Open Network Foundation. Към момента най-широко поддържани и използвани са версии 1.0 и 1.3.

В наше предишно изследване, описано в [4] са изложени проучванията за възможности за апаратни и програмни средства за крайни устройства. Поради високата цена на хардуерните реализации, нашият екип прие да използва софтуерни симулации на устройства. В настоящата разработка се използва симулаторът mininet [5]. С помощта му може да се създаде реалистична виртуална мрежа, симулираща комутатори и крайни устройства в различни конфигурации. В симулацията също може да се вкарва външен програмен код, симулиращ различни мрежови елементи, например контролер за софтуерно дефинирани мрежи.

Съществува голямо многообразие от контролери за софтуерно дефинирани мрежи, някои от които с отворен код, които могат да се вградят в симулатора. Някои от тях са:

- NOX - оригиналният OpenFlow контролер. Той служи като платформа за мрежово управление, която осигурява програмен интерфейс на високо ниво за управление и разработване на приложения за мрежов контрол. NOX първоначално е разработен от Nicira Networks, сега собственост на VMware, заедно с OpenFlow — беше представен за първи път на общността през 2009 г [6]. Съвременната версия на NOX поддържа само C++ и има по-малко приложения.
- POX - предоставя рамка за комуникация със SDN комутатори, използвайки протокола OpenFlow или OVSDB. Разработчиците могат да използват POX, за да създадат SDN контролер, използвайки езика за програмиране Python. Това е популярен инструмент за преподаване и изследване на софтуерно дефинирани мрежи и програмиране на мрежови приложения. POX може да се използва като основен SDN контролер, като се използват стандартните компоненти, които идват в комплект с него. Разработчиците могат да създадат по-сложен SDN контролер чрез създаване на нови POX компоненти или да пишат мрежови приложения, които адресират API на POX [7].
- Ryu е компонентно-базирана рамка за софтуерно дефинирани мрежи с отворен код на Python. Ryu предоставя софтуерни компоненти с добре дефиниран програмен интерфейс, който улеснява разработчиците да създават нови приложения за управление и контрол на мрежата. Ryu поддържа различни протоколи за управление на мрежови устройства, като OpenFlow, Netconf и др. За OpenFlow, Ryu поддържа напълно 1.0, 1.2, 1.3, 1.4, 1.5 и Nicira Extensions. Целият код е свободно достъпен под лицензa Apache 2.0 [8]. За съжаление има не особено добра производителност.

- OpenDaylight е сложен и разширяем проект на Java, широко разпро-странен в индустрията, с интеграция за OpenStack и много облачни платформи [9].

Сравнение на представените контролери е показано в таблица 1.

***Таблица 1***. *Сравнение на контролери*

| | NOX | POX | Ryu | ODL |
|---|---|---|---|---|
| Език | C++ | Python | Python | Java |
| Произв. | бърз | бавен | бавен | бърз |
| Разпре-делен | не | не | да | да |
| Open Flow | 1.0 | 1.0 | <=1.5 | 1.3 |
| учене | средно | лесно | средно | трудно |

За изграждане на настоящата платформа е избран контролерът POX, поради използвания от него програмен език python и възможностите за разширения.

## ОПИСАНИЕ НА РЕАЛИЗАЦИЯТА

За създаване на платформата за софтуерно дефинирани мрежи е необходимо да се изтегли и инсталира симулаторът mininet. Възможностите за инсталация са няколко: да се изтегли виртуална машина и да се стартира във виртуализационна платформа, да се инсталира чрез компилация на първичния код или да се използват готови прекомпилирани пакети, достъпни за някои версии на linux. В настоящата разработка е използван подходът с виртуална машина. Изтегленият файл е във формат Open Virtualization Format (.ovf), който се поддържа от много виртуализационни платформи. За настоящото изследване е избрана платформата Oracle Virtual Box – безплатен хипервайзор от тип 2 с отворен код, чрез който могат да се виртуализират различни операционни системи [10]. В него се импортира и стартира изтеглената виртуална машина.

За отдалечен достъп до виртуалната машина трябва да се използва SSH клиент. В настоящата разработка е използван безплатния клиент PuTTY [11]. Стартирането на симулацията и изграждането на мрежовата топология е показано на фигура 1.

```
mininet@mininet-vm:~$ sudo mn --topo single,3 --mac \
> --switch ovsk --controller remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> _
```

***Фиг. 1***. *Стартиране на топологията*

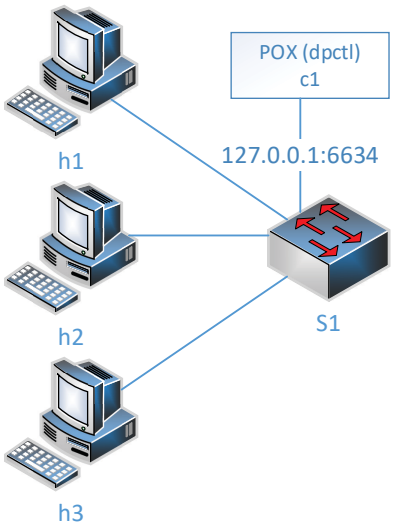
Командата за стартиране е:

```
sudo mn --topo single,3 --mac
--switch ovsk -controller remote
```

Параметрите означават:

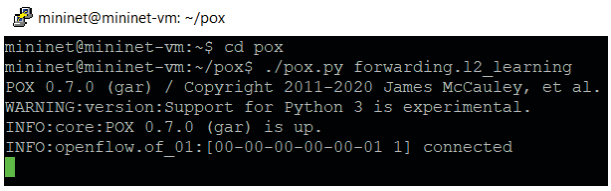
- sudo – изпълни командата с администраторски права;
- mn – команда mininet;
- topo single,3 – изгради топология с един комутатор и три компютъра;
- mac – използва малки числа за MAC адреси за лесно помнене;
- switch ovsk – да се използва комутатор Open vSwitch (OVS);
- controller remote – да се използва външен контролер, вместо вградения в комутатора.

От изхода на командата се вижда, че се създава контролер, стартиран на порт 6633, добавят се трите компютъра h1, h2 и h3, както и комутатора s1. Сега симулирана мрежа вече е стартирана. Тя има топологията, показана на фигура 2.



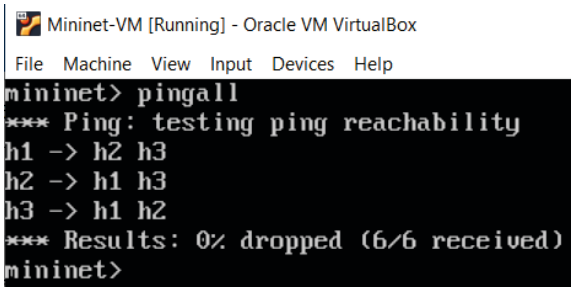

***Фиг. 2***. *Схема на топологията*

Сега трябва да стартираме контролера и да му определим модел на поведение. За целта се свързване със SSH клиента към виртуалната машина, отиваме в директория pox и стартираме python кода на контролера с параметър forwarding.l2_learning, който представлява стандартното поведение на комутатор – да научава MAC адресите на компютрите, свързани към него, да проверява MAC адреса на получателя в таблицата с MAC адресите и да препраща данните на съответния порт, където е свързан получателя. На фигура 3 е показан резултатът от стартирането на контролера и успешното му свързване към комутатора.




***Фиг. 3**. Стартиране на контролера*

За да проверим функционалността на симулираната мрежа, можем от терминала на виртуалната машина да използваме командата pingall, както е показано на фигура 4.



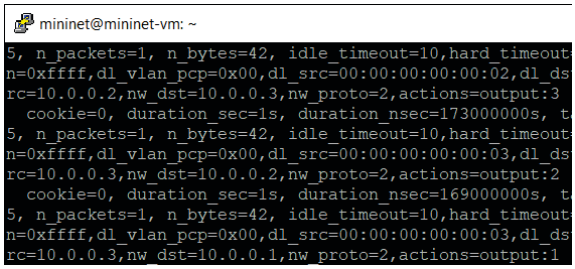

***Фиг. 4**. Проверка на работоспособността*

Командата изпраща тестови ICMP Echo Request пакети последователно от всеки компютър към всички други и ги поканва да върнат Echo Reply отговор. На фигурата се вижда, че всички 6 пакета са изпратени и получени успешно, т.е. мрежата е работоспособна.

С командата dpctl можем да видим създадените в паметта на комутатора потоци данни, в резултат от препратените пакети. Синтаксисът на командата е:

```
dpctl dump-flows tcp:127.0.0.1:6654
```

Където dump-flows означава да се покажат запаметените от комутатора пакети, адресът 127.0.0.1 е на локалната виртуална машина, а TCP порт 6654 е портът, на който се достъпва комутаторът s1. Ако в топологията има повече комутатори, те заемат следващите номера на портове. Резултатът от изпълнението на командата е показан на фигура 5.




***Фиг. 5**. Запаметени потоци от комутатора*

Част от екрана е изрязан, но изходът от командата показва всички изпратени в мрежата пакети от всеки компютър към останалите три и обратните отговори. На екрана се виждат лесните за проследяване IP адреси на трите компютъра: 10.0.0.1, 10.0.0.2 и 10.0.0.3, както и лесните MAC адреси 00:00:00:00:00:01, 00:00:00:00:00:02 и 00:00:00:00:00:03, в резултат на използвания при стартирането на топологията параметър --mac.

За да се симулират по-сложни топологии, може да се използват tree (дървовидна топология) и torus (тороид). Командата за дървовидна топология е:

```
topo=tree,depth=2,fanout=4
```

където topo=tree – създай дървовидна топология, depth=2 – дълбочина (брой нива) на дървото, fanout=4 – брой комутатори на всяко следващо ниво.

На фигура 6 е показано създаване на дървовидна топология.

```
mininet@mininet-vm:~$ sudo mn --topo tree,depth=2,fanout=4
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
*** Adding switches:
s1 s2 s3 s4 s5
*** Adding links:
(s1, s2) (s1, s3) (s1, s4) (s1, s5) (s2, h1) (s2, h2) (s2, h
s3, h8) (s4, h9) (s4, h10) (s4, h11) (s4, h12) (s5, h13) (s5
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
*** Starting controller
c0
*** Starting 5 switches
s1 s2 s3 s4 s5 ...
*** Starting CLI:
mininet> _
```

***Фиг. 6**. Създаване на дървовидна топология*

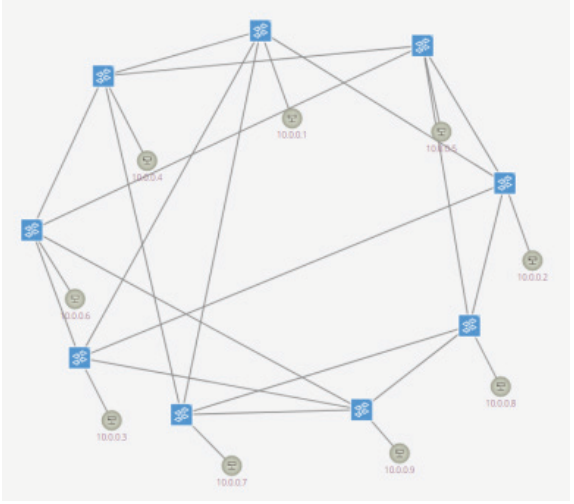
На фигурата се вижда, че се създават пет комутатора – s1 на първо ниво, към който е свързан контролера и четирите комутатора от второ ниво – s2, s3, s4 и s5, към които са свързани по четири компютъра – h1 до h16.

Създаването на тороидната топология е показано на фигура 7.

```
mininet@mininet-vm:~$ sudo mn --topo torus,3,3 --switch ovsk,stp=1
*** Creating network
*** Adding controller
*** Adding hosts:
h1x1 h1x2 h1x3 h2x1 h2x2 h2x3 h3x1 h3x2 h3x3
*** Adding switches:
s1x1 s1x2 s1x3 s2x1 s2x2 s2x3 s3x1 s3x2 s3x3
*** Adding links:
(h1x1, s1x1) (h1x2, s1x2) (h1x3, s1x3) (h2x1, s2x1) (h2x2, s2x2) (h
x2) (h3x3, s3x3) (s1x1, s1x2) (s1x1, s2x1) (s1x2, s1x3) (s1x2, s2x2
, s2x2) (s2x1, s3x1) (s2x2, s2x3) (s2x2, s3x2) (s2x3, s2x1) (s2x3,
s3x2, s1x2) (s3x2, s3x3) (s3x3, s1x3) (s3x3, s3x1)
*** Configuring hosts
h1x1 h1x2 h1x3 h2x1 h2x2 h2x3 h3x1 h3x2 h3x3
*** Starting controller
c0
*** Starting 9 switches
s1x1 s1x2 s1x3 s2x1 s2x2 s2x3 s3x1 s3x2 s3x3 ...
*** Starting CLI:
mininet> _
```

***Фиг. 7**. Създаване на тороидна топология*

Графичното представяне на тороидната топология е представено на фигура 8.



***Фиг. 8**. Тороидна топология*

Топологията се състои от 9 комутатора с по един компютър, свързан към всеки комутатор. Тъй като между комутаторите има излишни връзки, тя няма да работи, ако не се стартира протокол, който да блокира излишните връзки и да премахва затворените цикли. В случая е използван протокол STP (Spanning Tree Protocol) [12].

Възможно е да се създаде всякаква топология, като се създаде скрипт на python и се наследи вградения клас Topo. В него трябва да се създадат комутаторите, компютрите и връзките по следния начин:

```
from mininet.topo import Topo
class topo(Topo):
    def __init__( self ):
    Topo.__init__( self )
    s1 = self.addSwitch('s1')
    s2 = self.addSwitch('s2')
    #...
    h1 = self.addHost('h1')
    h2 = self.addHost('h2')
    #...
    self.addLink(h1,s1)
    self.addLink(h2,s2)
    #...
topos={ 'topo': (lambda: topo () )}
```

Записва се файла с произволно име, например topo.py, след което топологията се стартира по следния начин:

```
sudo mn --custom ./topo.py
```

## ЗАКЛЮЧЕНИЕ

В настоящия доклад е разгледана опитна постановка за тестване на комуникация с контролер за софтуерно дефинирани мрежи. Текущата постановка използва един контролер – POX, написан на python. Бъдещата работа предполага изследване на всички описани в увода контролери, модификация на съществуващия код с цел добавяне на нови функционалности и модификация на съществуващи. Крайната цел е избор на най-подходящия контролер, конфигуриране на отказоустойчивост и създаване на приложение за мрежово наблюдение и управление, което да дава възможност на потребителя чрез графичен интерфейс да

дефинира настройките за мрежовите устройства. Приложението ще комуникира с контролера, чрез северен интерфейс (Northbound Interface), който ще бъде създаден специално за целта, под формата на REST API.

*Източник на финансиране: Настоящият документ е изготвен с финансовата помощ на договор № 2405Е за провеждане на научни изследвания по проект на тема: „Усъвършенстване на обучението чрез информационни и комуникационни технологии“ към Технически университет – Габрово.*

## ЛИТЕРАТУРА

[1] Jerzy Kaczmarski, Management vs. Control vs. Data Planes in a Network Device - https://codilime.com/blog/management-plane-vs-control-plane-vs-data-plane/ - Date of usage: 06.10.2024.

[2] Open Networking Foundation. OpenFlow. https://opennetworking.org/sdn-resources/customer-case-studies/openflow/ Date of usage 05.10.2024.

[3] Cisco Systems. NETCONF Protocol. https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/168/b_168_programmability_cg/NETCONG_YANG.pdf, Date of usage 05.10.2024.

[4] Genkov, D., Ts. Raykov, Software Defined Networking – Concepts, Implementations and Tools.

[5] Minimet, Mininet - An Instant Virtual Network on your Laptop (or other PC). http://mininet.org/, Date of usage 08.10.2024.

[6] Sridhar Rao. SDN Series Part Three: NOX, the Original OpenFlow Controller, https://thenewstack.io/sdn-series-part-iii-nox-the-original-openflow-controller/, Date of usage 05.10.2023.

[7] Brian Linkletter, Using the POX SDN controller, https://brianlinkletter.com/2015/04/using-the-pox-sdn-controller/, date of usage: 02.10.2024.

[8] Ryu SDN Framework Community, Build SDN Agilely. https://ryu-sdn.org/. Date of usage 05.10.2024.

[9] OpenDaylight Project, OpenDaylight, https://www.opendaylight.org/, Date of usage, 05.10.2024.

[10] Oracle, VirtualBox - Powerful open source virtualization, https://www.virtualbox.org/, Date of usage – 10.10.2024.

[11] Simon Tatham, PuTTY - an SSH and telnet client, https://www.putty.org/, Date of usage 10.10.2024.

[12] Cisco Systems, Spanning Tree Protocol, https://www.cisco.com/c/en/us/tech/lan-switching/spanning-tree-protocol/ index.html, Date of usage: 07.11.24.