СОФТУЕРНО ДЕФИНИРАНИ МРЕЖИ – КОНЦЕПЦИИ, РЕАЛИЗАЦИИ И ИНСТРУМЕНТИ

Делян Генков[1], Цветан Райков[1]
[1]Технически университет - Габрово

SOFTWARE DEFINED NETWORKING – CONCEPTS, IMPLEMENTATIONS AND TOOLS

Delyan Genkov[1], Tsvetan Raykov[1]
[1] Technical University - Gabrovo

Abstract
Software defined networks become a very popular topic in the past years. There are many different approaches and implementations that can be named software defined networks. The present paper summarizes some of the different approaches and proposes a way to create and study a software defined network.

Keywords: Software Defined Networks, SDN, Controller, OpenFlow.

ВЪВЕДЕНИЕ

Традиционните мрежи се състоят от множество мрежови устройства (маршрутизатори, комутатори, защитни стени), които трябва да работят заедно, за да осигурят правилното функциониране на мрежата. Всяко от тези устройства се настройва отделно от останалите и взима самостоятелно решение какво да прави с конкретен мрежов пакет. За да се настрои такава мрежа трябва администратора да има цялостна визия за начина на работа на мрежата, да моделира правата на поведение за всяко отделно устройство и да ги изпълни, чрез прилагане на подходяща конфигурация на отделните устройства. При това е възможно да се допусне грешка при осмислянето на концепцията от гледна точка на конкретно устройство, при конфигурирането му или дори при начина, по който устройството прилага правилата.

При концепцията за софтуерно дефинирана мрежа (Software Defined Network, SDN) се предполага, че конфигурирането се извършва в централна софтуерна платформа, която изработва конфигурация за всяко устройство и промените се прилагат едновременно на всички устройства. По този начин се изработват правилата от гледна точка на цялата мрежа, а не от гледна точка на отделно устройство, намалява се възможността за грешки при конфигурациите и промените се прилагат едновременно, което намалява времето на неработоспособност на мрежата. Освен това централизираното управление предполага възможности за висока степен на автоматизация на процесите и преконфигуриране на мрежовите устройства и топологии при отказ или промяна на условията.

Много организации разработват концепция за това как трябва да изглежда софтуерно дефинирана мрежа. В момента на пазара има доста решения, които се различават по архитектура и по начин на функциониране.

АРХИТЕКТУРА НА SDN

При традиционните мрежи всяко устройство трябва да взима самостоятелно решение какво да прави с дадена порция данни – дали да я предаде или да я изхвърли, къде да я предаде, как да научи топологията на мрежата и т.н. Затова всяко устройство има три основни компонента, условно наречени „равнини“ (planes). Структурата е показана на фиг. 1.

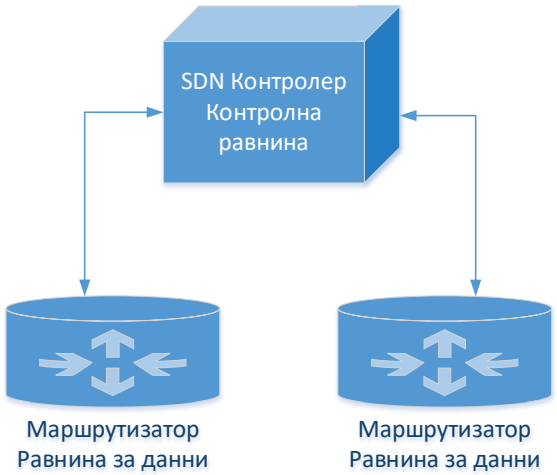
***Фиг. 1**. Структура на традиционно устройство*

Управляващата равнина предоставя графичен и/или текстов интерфейс, чрез който се конфигурира устройството.

В контролната равнина работят алгоритмите, които дават възможност на устройството да взима решения, да научава мрежовата топология, да си взаимодейства с другите устройства, както и таблиците с научената информация – например информация за съседите, маршрутизираща таблица и др.

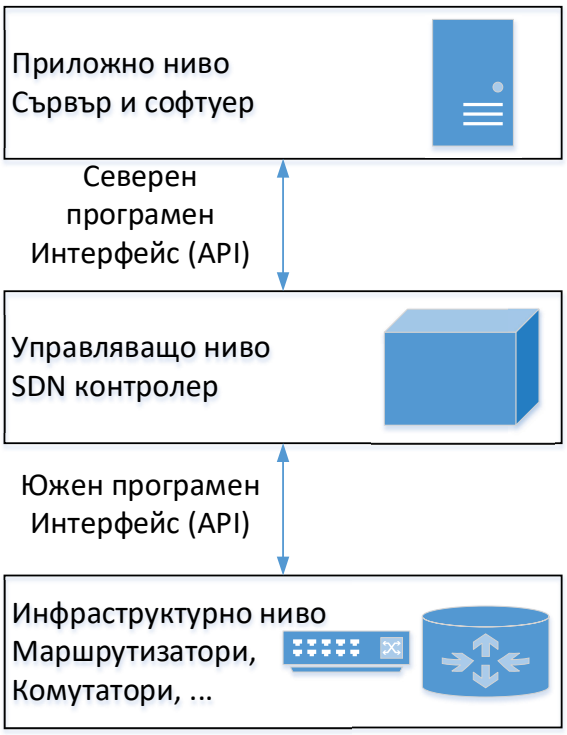
В равнината за данни се намира изчислената от горната равнина таблица за препращане (Forwarding Table), която прилага взетото решение и определя дали и къде ще бъде изпратен даден мрежов пакет, в зависимост от адресите, съдържанието или други параметри.

При софтуерно дефинираните мрежи контролната равнина се изважда от мрежовото устройство и се поставя в специализирано устройство – контролер, което централизирано изработва таблиците за препращане на всички устройства и им ги изпраща, а те работят според изчисленията на контролера. Концепцията е показана на фиг. 2 [1].



***Фиг. 2**. Структура на SDN устройство*

Архитектурата на SDN мрежата е структура на три нива, както е показано на фиг. 3 [2].



***Фиг. 3**. Архитектура на SDN мрежа*

На инфраструктурното ниво работят устройствата, които изграждат мрежова-

та инфраструктура – маршрутизатори, комутатори и др. Те получават таблиците си със знания за препращането на пакетите и сигнализират за промяна в мрежовата топология към управляващото ниво чрез протокол, наричан южен програмен интерфейс (Southbound API). Примери за такива протоколи са OpenFlow [3], Cisco OpFlex [4], NETCONF [5] и други.

На управляващото ниво работи един или често повече от един контролер за отказоустойчивост и разпределяне на натоварването. Това е софтуерен модул, който получава информация за топологията на мрежата от устройствата, стартира протоколи, изчисляващи най-добрите пътища в мрежата, изработва таблиците с правила кои пакети къде да се препращат за всяко устройство и ги изпраща към устройствата.

Известни са много SDN контролери, сред които: NOX/POX [6], Ryu [7], Floodlight [8], OpenDaylight [9] и други.

NOX е оригиналният контролер на OpenFlow, разработен от Nicira Networks, в момента собственост на VMware. Текущата му версия е написана на C++ и поддържа само него като програмен език. POX е версията на NOX, написана и поддържаща програмен език Python. Ryu е контролер с отворен код на Python, който поддържа всички версии на OpenFlow, но има не особено добра производителност. Floodlight е контролер на Java с отворен код, поддържа всички версии на OpenFlow, има добра документация и притежава много добра производителност, достатъчна за продукционна среда. OpenDaylight е сложен и разширяем проект на Java, широко разпространен в индустрията, с интеграция за OpenStack и много облачни платформи.

Северният програмен интерфейс позволява на приложенията да управляват SDN мрежата. Тези интерфейси обикновено не са стандартни и с отворен код. Компаниите разработват собствени решения, за да използват собствени приложения, които на базата на програмируемата мрежа да реализират допълнителни функционалности, например мрежови защитни стени, системи за балансиране на натоварването или приложения за централизирано управление и наблюдение на мрежата. Понякога те публикуват програмен интерфейс (API), най-често базиран на REST протокол, който да направи възможна интеграцията на техните хардуерни и софтуерни решения в програмите за мрежово наблюдение на други производители.

ИЗГРАЖДАНЕ НА ЛАБОРАТОРНА ПОСТАНОВКА ЗА SDN

За изграждане на лабораторна постановка за симулиране и изследване на софтуерно дефинирани мрежи и разработка на приложения за наблюдение и управление са необходими следните компоненти:

1) Крайни устройства.

Хардуерните устройства (комутатори), които поддържат стандартен южен програмен интерфейс са доста скъпи. Устройства, произведени в Китай от непопулярни марки се предлагат на цени по-високи от 2000 лева и стигат до 5-цифрени суми, а тези на маркови производители са непосилни за изследователски лаборатории.

Преди години беше стартиран проект за евтин OpenFlow съвместим комутатор – Zodiac-FX с цена около 75 щатски долара, но проектът беше изоставен през 2017 г.

Има някои публикации, които представят моделирането на хардуерен комутатор, използвайки Raspberry Pi мини компютър. Той има само един вграден мрежов интерфейс и още четири USB порта, към които могат да се свържат USB-Ethernet конвертори и така на цена малко над 100 евро може да се симулира 5-портов комутатор, съвместим с OpenFlow или друг стандартен интерфейс.

Съществуват виртуални комутатори, които могат да работят във виртуализирана среда. Един популярен такъв е Open vSwitch – виртуален комутатор с отворен код и производителност достатъчна за продукционна среда. С такова решение може да се изгради изцяло виртуализи-

рана софтуерно дефинирана мрежа, което е сравнително евтино и доста мащабируемо решение при наличие на виртуална среда за изследвания.

Има доста симулационни среди, които имат възможност да симулират SDN комутатор и да им се вгради контролер. Едно популярно решение за симулация на SDN мрежа, заедно с крайни устройства, свързани към виртуалния комутатор е симулаторът Mininet [10].

2) Южен интерфейс. В предишната точка бяха дадени примери за различни южни интерфейси. 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 Fondation. Към момента най-широко поддържани и използвани са версии 1.0 и 1.3.

3) SDN контролер. Както беше описано по-горе има голямо многообразие от SDN контролери с отворен код. Изборът на контролер зависи от целите, които си поставяме за постановката – дали да е с висока производителност, какъв програмен език да използва, коя версия на OpenFlow или друг протокол поддържа и други критерии. В таблица 1 е представено кратко сравнение на описаните контролери.

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

	NOX	POX	Ryu	Fllod-light	ODL
Език	C++	Python	Python	Java	Java
Произв.	бърз	бавен	бавен	бърз	бърз
Разпределен	не	не	да	да	да
Open Flow	1.0	1.0	<=1.5	1.0	1.3
учене	Средно	лесно	средно	Трудно	Трудно

4) Северен програмен интерфейс. Както беше описано по-горе, този интерфейс не е стандартен и няма много такива с достатъчна функционалност за директно използване. Това е компонент, който трябва да се разработи от изследователския екип с възможност за комуникация с приложението за управление и наблюдение на мрежата и възможност за външна комуникация с подходящ REST интерфейс. Като пример за такъв интерфейс, може да се използва описанието на интерфейса на Aruba Networks [11]. За този компонент могат да бъдат използвани и някои езици от високо ниво, като:

- Frenetic – програмен език, подобен на SQL, с който могат да се правят запитвания за състоянието на мрежата [12].
- Pyretic – език и модул за софтуерно дефинирани мрежи, позволяващ на програмиста да дефинира политики за пакетите, базирани на местоположението им в мрежата [13].
- Kinetic – език за динамично управление на поведението на мрежата, на базата на възникнали събития в нея [14].

5) Приложение за мрежово наблюдение и/или управление. То е предмета на разработка за изследователския екип и може да бъде с разнообразни функции и предназначение – от система за наблюдение на мрежата до сложни системи за мрежова защита, за динамично балансиране на натоварването, за разделяне на мрежовия трафик на много клиенти в една обща мрежова среда, за динамично прилагане на политики за достъп и много други.

ЗАКЛЮЧЕНИЕ

В настоящата работа бяха описани разликите между традиционна и софтуерно дефинирана мрежа. Беше дефинирана архитектурата и основните компоненти на софтуерно дефинираната мрежа. Бяха разгледани и сравнени основните компоненти, чрез които може да се създаде лабораторна тестова среда за из-

следване и развитие на софтуерно дефинирани мрежи, както и възможните цели за изследователските екипи, за които тази област е предмет на бъдещи разработки.

Пред работния екип предстои тестване на различните предложени възможности за опитни постановки за изследвания, избор на най-подходящи компоненти за създаване на такава и изследователска работа по създаване на конкретна система за изследване, тестване и създаване на приложения и системи за софтуерно дефинирани мрежи.

БЛАГОДАРНОСТИ

Настоящият документ е изготвен с финансовата помощ на договор № 2304Е за провеждане на научни изследвания по проект на тема: „Повишаване на ефективността на образованието чрез използване на информационни и комуникационни технологии“ към Технически университет – Габрово.

REFERENCE

[1] NetworkLessons.com. Introduction to SDN (Software Defined Networking). https://networklessons.com/cisco/ccna-routing-switching-icnd2-200-105/introduction-to-sdn-software-defined-networking#SDN_Software_Defined_Networking, Date of usage 05.10.2023.

[2] study-ccna.com. Cisco SDN – Software Defined Networking Explained. https://study-ccna.com/cisco-sdn-software-defined-networking/. Date of usage 05.10.2023.

[3] Open Networking Foundation. OpenFlow. https://opennetworking.org/sdn-resources/customer-case-studies/openflow/ Date of usage 05.10.2023.

[4] Shashi Kiran. Introducing OpFlex – A new standards-based protocol for Application Centric Infrastructure. https://blogs.cisco.com/datacenter/introducing-opflex-a-new-standards-based-protocol-for-application-centric-infrastructure. Date of usage 05.10.2023.

[5] 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 05.10.2023.

[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] Ryu SDN Framework Community, Build SDN Agilely. https://ryu-sdn.org/. Date of usage 05.10.2023.

[8] Atlassian, Floodlight Controller, https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview, Date of usage 05.10.2023.

[9] OpenDaylight Project, OpenDaylight, https://www.opendaylight.org/, Date of usage, 05.10.2023.

[10] Minimet, Mininet - An Instant Virtual Network on your Laptop (or other PC). http://mininet.org/, Date of usage 08.10.2023.

[11] HPE, Aruba Networks Northbound API, https://www.arubanetworks.com/techdocs/ArubaOS_85_Web_Help/Content/arubaos-solutions/sdn-ctrl/nbap.htm, date of usage 08.10.2023.

[12] Frenetic. Frenetic a family of network programming languages. http://frenetic-lang.org/, date of usage 08.10.2023.

[13] Frenetic, Pyretic: embedded and implemented in Python. http://frenetic-lang.org/pyretic/. Dat of usage 08.10.2023.

[14] Frenetic, Kinetic: a state-machine based network control platform, http://frenetic-lang.org/, date of usage 08.10.2023.