International Scientific Conference 20 UNITECH 26

INTERNATIONAL SCIENTIFIC CONFERENCE
19-21 November 2026
GABROVO

АРХИТЕКТУРА НА СИСТЕМА ЗА ИЗСЛЕДВАНЕ НА СОФТУЕРНО ДЕФИНИРАНИ МРЕЖИ

Цветан Райков[1], Делян Генков[1], Виктория Иванова[1]

[1]Технически университет - Габрово, ул. Хаджи Димитър 4, Габрово, България
**кореспондиращ автор: dgenkov@tugab.bg*

ARCHITECTURE OF A SYSTEM FOR SOFTWARE DEFINED NETWORK

Tsvetan Raykov[1], Delyan Genkov[1], Viktoria Ivanova[1]

[1]Technical University - Gabrovo, 4, Hadji Dimitar Str., Gabrovo, Bulgaria
**Corresponding author: dgenkov@tugab.bg*

Abstract

Software Defined Networking is a wide used approach in modern computer networks. Many vendors implement this approach in their commercial networking solutions. There are also many open-source implementations. Currently no open, easily deployable platform exists that combines SDN approach, using OpenFlow capable controller, and a web interface in a single environment. Present work describes architecture of an integrated management system for Software-Defined Networking (SDN) based on OpenFlow 1.3. The architecture has two layers with a clear separation: the management layer operates exclusively through the SDN controller REST API, while the structural layer uses the Mininet REST API, accessible only from the Data Plane page.

Keywords: software-defined networking, SDN, OpenFlow, Mininet, Ryu, Blazor Server, REST API.

ВЪВЕДЕНИЕ

Глобалният IP трафик е нараснал от под 1 EB годишно в началото на 2000 година до над 7000 EB към 2025 г. [1], като облачните услуги, поточните видео услуги и Интернет на нещата (IoT) поддържат ръста. Центровете за данни и телекомуникационните оператори търсят мрежови архитектури с по-висока гъвкавост и автоматизирано управление.

Традиционната мрежова архитектура вгражда управляващата логика (control plane) директно в хардуера на всяко мрежово устройство. Конфигурацията се осъществява устройство по устройство чрез командния ред или специализирани протоколи за мрежово управление — подходи, непригодни за динамична преконфигурация. Добавянето на нова политика за маршрутизация изисква промени на множество устройства, всяко с различни конфигурационни интерфейси. Тази разпокъсаност пречи на автоматизацията и увеличава риска от конфигурационни грешки.

Идеята зад софтуерно дефинираните мрежи (Software-Defined Networking, SDN) е да се раздели физически управляващия слой от слоя за препращане на данни. Мрежовите устройства изпълняват само пренос на пакети, докато логиката на маршрутизация е централизирана в SDN контролер — програмен компонент, комуникиращ с устройствата чрез стандартизиран протокол. Протоколът OpenFlow [2], стандартизиран от Open

Print ISSN: 1313-230X, Online ISSN: 3033-1404

Networking Foundation (ONF), е де факто стандарт за тази комуникация и се използва в академични изследвания, центрове за данни и телекомуникационни мрежи.

Въпреки широкото внимание към SDN, изследователите и преподавателите срещат практическа пречка: не съществува интегрирана среда, съчетаваща SDN емулация, програмируем контролер и уеб интерфейс за управление.

Съществуващите решения са фрагментирани. Mininet [3] — водещата платформа за SDN емулация — предоставя пълноценна виртуална мрежа с OVS (Open vSwitch) комутатори, но се управлява изцяло от командния ред или Python скриптове, без интерактивен графичен интерфейс. Ryu [4] е утвърден SDN контролер с богат REST API, но собствените му управляващи инструменти са ограничени до сурови JSON заявки.

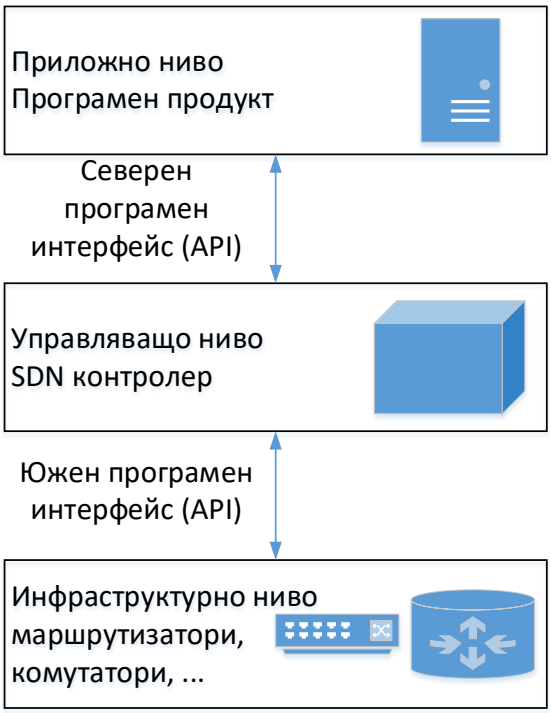
Комерсиалните платформи като ONOS и OpenDaylight предлагат уеб интерфейси, но изискват значителни хардуерни ресурси и не се разгръщат лесно в академичен контекст. Академичните симулатори (GNS3, EVENG) симулират пълни мрежови устройства, но не осигуряват програмируем OpenFlow control plane.

Целта на разработката е да се предложи архитектура на интегрирана система за управление на SDN мрежа, базирана на OpenFlow 1.3 протокол, която да осигурява управление на мрежата чрез SDN контролера и да осъществява измеримо поведение при динамични топологии.

ИЗЛОЖЕНИЕ

1. Архитектурен модел на SDN.

Референтният архитектурен модел на SDN, дефиниран от Open Networking Foundation [2], разделя функциите на мрежата в три концептуални слоя, свързани чрез стандартизирани интерфейси, както е показано на фигура 1.



***Фиг. 1**. Архитектура на софтуерно дефинирана мрежа*

Инфраструктурното ниво се състои от физически или виртуални мрежови устройства — комутатори и маршрутизатори, чиято единствена отговорност е препращането на пакети съгласно правилата, инсталирани от управляващото ниво. Устройствата в това ниво нямат собствена управляваща логика и са изцяло зависими от контролера. В контекста на OpenFlow протокола устройствата изпълняват match-action операции върху входящите пакети спрямо съдържанието на своите flow таблици.

Управляващото ниво е реализирано от SDN контролера — централизиран софтуерен компонент, който поддържа глобален изглед на мрежовата топология и взема всички решения за препращане. Контролерът комуникира с инфраструктурния слой чрез южен интерфейс, при OpenFlow — чрез TCP/TLS връзка на порт 6653. Контролерът комуникира с приложното ниво чрез северен интерфейс, типично реализиран като REST API.

Приложното ниво слой съдържа мрежовите приложения — системи за управление на трафика, системи за сигурност, балансиращи натоварването решения, мониторингови платформи. Те изразяват своите мрежови намерения (intents) към контролера чрез северен API, без да се нуждаят от познания за физическата топология или за конкретния тип устройства.

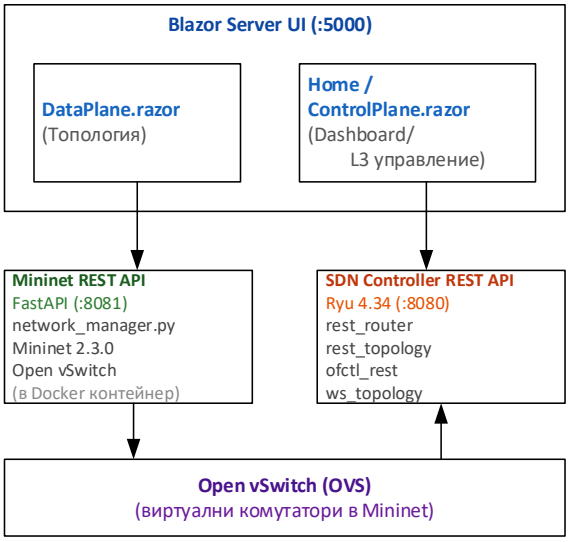
Ключовото предимство е програмируемостта: поведението на цялата мрежа може да се промени чрез актуализиране на приложенията или политиките в приложното ниво, без физическа намеса в инфраструктурата. Централизираният управляващ слой е и естествена платформа за интеграция на машинно обучение в мрежовото управление — направление, активно изследвано от 2022 г. насам.

2. Предложена архитектура

Централното архитектурно решение на разработваната система е разделено на двуслойна архитектура, като целта е явното разделяне между управлението на структурата на мрежата (data plane) и управлението на нейното поведение (control plane) на ниво потребителски интерфейс. Това разделяне ще бъде реализирано чрез два независими REST клиента в Blazor приложението, като всеки комуникира с различен backend компонент. Предложената архитектура е показана на фигура 2.

В реална физическа SDN мрежа операторът управлява мрежовото поведение изключително чрез SDN контролера — той никога не конфигурира директно физическите комутатори.

Mininet API в настоящата система изпълнява ролята на „физическата инфраструктура" — той създава и премахва виртуални устройства, но не взема решения за маршрутизиране. Тази архитектура прави системата реалистичен модел на SDN среда в продукционна употреба, а не просто Mininet wrapper.



***Фиг. 2**. Предложена архитектура на системата*

3. Компоненти на системата

3.1. SDN контролер

За SDN контролер е избран Ryu 4.34 [4]. Ryu е Python-базиран SDN контролер с компонентна архитектура, разработен от NTT Labs. Ключовата му характеристика е изключително лесната разработка: контролерните приложения се пишат като Python класове, наследяващи RyuApp, а OpenFlow съобщенията се обработват чрез декоратори (@set_ev_cls). Ryu поддържа OpenFlow 1.0–1.5 и предоставя богат набор от готови REST API модули.

Академичното изследване на Amin и съавтори от 2025/2026 г. [5] оценява Ryu според пропускателна способност и QoS метрики и потвърждава неговата производителност за изследователски сценарии. За настоящата система Ryu 4.34 е инсталиран от source в Ubuntu 20.04 Docker контейнер; това заобикаля ограниченията на Python версията.

3.2. Мрежов емулатор

За мрежов емулатор е избран Mininet 2.3.0 [3]. Mininet е специализирана платформа за емулация на SDN мрежи, разработена в Stanford University от Lantz, Heller и McKeown. За разлика от GNS3 и EVE-NG, Mininet не виртуализира пълни

операционни системи, а използва Linux namespaces — лека технология за изолация на мрежовите стекове в рамките на един Linux kernel. Всеки виртуален хост в Mininet е отделен network namespace, а комутаторите са реализирани чрез Open vSwitch (OVS) — пълноценен OpenFlow-съвместим софтуерен комутатор.

3.3. Южен протокол (Southbound API)

За комуникация между управляващото и инфраструктурното ниво е избран протокол OpenFlow 1.3 [6]. Протоколът дефинира комуникацията между контролера и OpenFlow-съвместимите устройства.

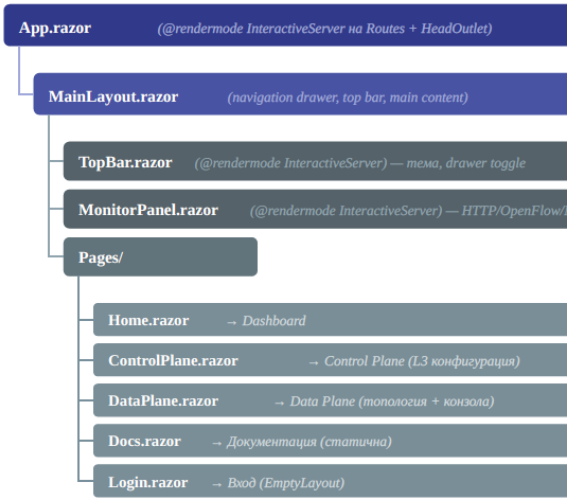
Въпреки зрелостта на OpenFlow 1.3, протоколът има фундаментални архитектурни ограничения. Те мотивират преминаването на индустрията и изследователите към P4. За настоящата разработка обаче OpenFlow 1.3 е напълно достатъчен: целта е управление на IP мрежа с L3 маршрутизиране — задача, за която протоколът е добре доказан.

3.4. Управляващ UI

За потребителския интерфейс на системата е избран .NET 9 Blazor Server [7] с компонентна библиотека MudBlazor 8.0. Blazor Server режимът изпълнява C# кода на сървъра и изпраща DOM промените към браузъра чрез SignalR (WebSocket). Това позволява използването на C# за целия стек — HTTP клиенти, бизнес логика и UI — без JavaScript framework. За настоящата система с малък брой едновременни потребители (изследователска среда).

4. Архитектура на UI компонентите

Архитектурата на компонентите следва стандартния Blazor Server шаблон и е показана на фигура 3.



***Фиг. 3**. Архитектура на UI*

Глобалното споделено състояние между компонентите е организирано в singleton services, регистрирани в Program.cs.

4.1. Data Plane страница

Страницата Data Plane (/data-plane) е единственият компонент в системата, комуникиращ с Mininet REST API. Тя предоставя графичен редактор на мрежовата топология.

4.2. Control Plane страница

Страницата Control Plane (/control-plane) е основният инструмент за конфигуриране на SDN мрежата. Тя комуникира единствено с контролера чрез IRyuApiClient и не прави никакви извиквания към Mininet API, отразявайки реалистичния модел на SDN управление.

4.3. Dashboard страница

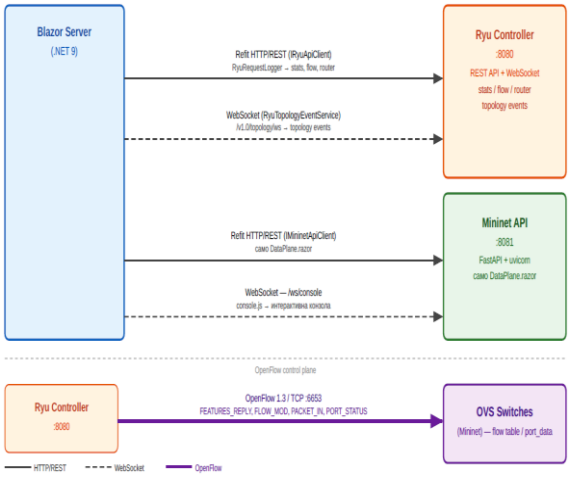
Dashboard страницата (/) предоставя обобщен изглед на мрежовото състояние в реално време. Страницата се обновява на всеки 5s чрез System.Threading.Timer.

4.4. Автентикация

Системата ще поддържа опционална cookie-базирана автентикация с access key. В продукционна среда операторът задава произволен access key чрез environment variable ACCESS_KEY; в среда на разработка (ключ = "") автентикацията може да бъде деактивирана.

5. Обмен на информация между компонентите

На фигура 4 е показано предвиденото взаимодействие между компонентите на системата.



***Фиг. 4**. Комуникационен модел*

5.1. Обмен на информация между Blazor Server и Ryu контролер

Обменът на информация между Blazor Server и контролера използва HTTP протокол. Всяка заявка преминава през опашка и достига до контролера. Комуникацията ще се извършва на TCP порт 8080, както е предвидено по подразбиране в Ryu.

5.2. Получаване на информация за събития в мрежата

Получаването на информация за събития в мрежата, например за промяна на топологията се извършва чрез установяването на три паралелни WebSocket канала.

Модулът Topology events установява WebSocket връзка към Ryu контролера за получаване на информация за събития в реално време.

Конзолата на Mininet емулатора комуникира с виртуалната машина за прихващане на грешки и събития.

Модулът OpenFlow monitor прихваща съобщения от протокола и ги обработва.

Комуникацията ще се извършва на TCP порт 8081, който е настроен по подразбиране на Mininet API.

5.3. Обмен на информация между контролера и мрежовите устройства

За комуникация между контролера Ryu и OVS комутаторите се използва протокол OpenFlow 1.3. Обменът на информация ще се извършва на TCP порт 6653, който се използва по подразбиране от OVS комутаторите.

Всички използвани TCP портове за обмен на информация са избрани по подразбиране, но могат да бъдат променяни при нужда. Тъй като целта на разработката е платформа за научни изследвания, а не реална продуктивна среда, за да не се натоварва системата и за по-лесно използване в текущата разработка трафикът не е криптиран. За бъдещи приложения се предвижда да се използва криптиране на информацията с помощта на TLS 1.3 протокол, с цел повишаване на сигурността на информацията.

ЗАКЛЮЧЕНИЕ

Настоящата разработка предлага архитектура на интегрирана система за управление на SDN мрежа, базирана на протокола OpenFlow 1.3. Системата обединява три самостоятелни компонента — Mininet REST API, SDN Controller REST клиент и Blazor Server уеб интерфейс — в единна full-stack платформа, разгръщаема в Docker среда с TLS-защитена публична достъпност.

Бъдещите планове включват разработка на всички модули на описаната среда, детайлно тестване и интеграция на системата, както и провеждане на изследвания в различни топологии и за разнообразни приложения на софтуерно дефинирани мрежи.

БЛАГОДАРНОСТИ: *Настоящият документ е изготвен с финансовата помощ на договор № НИП2026-6 за провеждане на научни изследвания по проект на тема: „Изследване и изграждане на система за откриване на аномалии в компютърни мрежи” към Технически университет – Габрово.*

ЛИТЕРАТУРА

[1] Tristan Gaudiaut. Internet Traffic Is Surging Worldwide. May 26, 2026. Available online: https://www.statista.com/chart/36248/volume-of-global-internet-traffic-in-2020-and-2025/, Date of Usage 20.09.2026.

[2] N. McKeown et al., OpenFlow: Enabling Innovation in Campus Networks, ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, pp. 69–74, 2008.

[3] B. Lantz, B. Heller, and N. McKeown, A Network in a Laptop: Rapid Prototyping for Software-Defined Networks, ACM HotNets, 2010.

[4] Ryu SDN Framework Community, Ryu SDN Framework, Available online at https://ryu-sdn.org/, Date of usage: 20.09.2026.

[5] P. Amin, I. Rojas et al., "A Survey on Machine Learning Techniques for Routing Optimization in SDN," IEEE Access, 2021. DOI: 10.1109/ACCESS.2021.3100560

[6] Open Networking Foundation. OpenFlow. https://opennetworking.org/sdn-resources/customer-case-studies/openflow/ Date of usage 05.10.2025.

[7]Microsoft, Launch your idea to the web fast with Blazor, Available online: https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor, Date of usage: 10.09.2026.