Zeszyty Naukowe
Wyższej Szkoły Ekonomii i Informatyki
Nr 20/2024, s. 191-208
DOI: XXXXXX

ISSN 1734-5391
e-ISSN 2957-2223

Marta Magdalena Szymczyk
mgr inż.
AGH Akademia Górniczo-Hutnicza
email: marta.m.szymczyk@gmail.com
ORCID: 0009-0007-0679-2407

ANALIZA METODY ELASTYCZNEGO STEROWANIA RUCHEM W SIECIACH SDN

ANALYSIS OF FLEXIBLE TRAFFIC CONTROL METHOD IN SDN

Słowa kluczowe: *sieci sterowane programowo, sieci neuronowe, uczenie przez wzmacnianie, głębokie uczenie*

Key words: *Software-Defined Networks, SDN, Neural Networks, Reinforcement Learning, Deep Learning*

JEL Classification: C 45

WSTĘP

Jednym z najbardziej zaawansowanych rozwiązań, które pojawiły się w odpowiedzi na wyzwania stawiane nowoczesnym sieciom komputerowym, są sieci definiowane programowo (SDN). Umożliwiają one efektywne zarządzanie zasobami oraz ruchem sieciowym, co jest niewątpliwą zaletą w obliczu coraz bardziej skomplikowanych sieci wymagających dynamicz-

nego zarządzania. Dzięki centralizacji kontroli i możliwości elastycznego zarządzania, SDN otwiera nowe możliwości w zakresie optymalizacji sieci. Niemniej jednak pełne wykorzystanie potencjału SDN wymaga opracowania zaawansowanych i adaptacyjnych metod sterowania.

Artykuł ten skupia się na analizie aktualnych metod elastycznego sterowania sieciami SDN oraz na prezentacji rozwiązania mającego na celu poprawę efektywności i adaptacyjności zarządzania sieciami. Przedstawione podejście opiera się na zastosowaniu uczenia maszynowego, a w szczególności na technice uczenia przez wzmacnianie (ang. Reinforcement Learning). Technika ta pozwala sieciom na autonomiczne podejmowanie decyzji, bazując na wcześniejszych doświadczeniach i dynamicznie zmieniających się warunkach, co jest zbliżone do sposobu, w jaki uczą się ludzie.

Celem proponowanego rozwiązania jest nie tylko zwiększenie wydajności sieci, ale także poprawa jej elastyczności oraz zdolności do adaptacji w czasie rzeczywistym. Wykorzystanie uczenia przez wzmacnianie umożliwia dynamiczne i elastyczne sterowanie ruchem sieciowym, co przekłada się na bardziej efektywne i responsywne zarządzanie zasobami. W artykule dokonano przeglądu istniejących rozwiązań oraz szczegółowo opisano opracowane własne, oryginalne podejście, wskazując na jego potencjalne korzyści i możliwości wdrożenia.

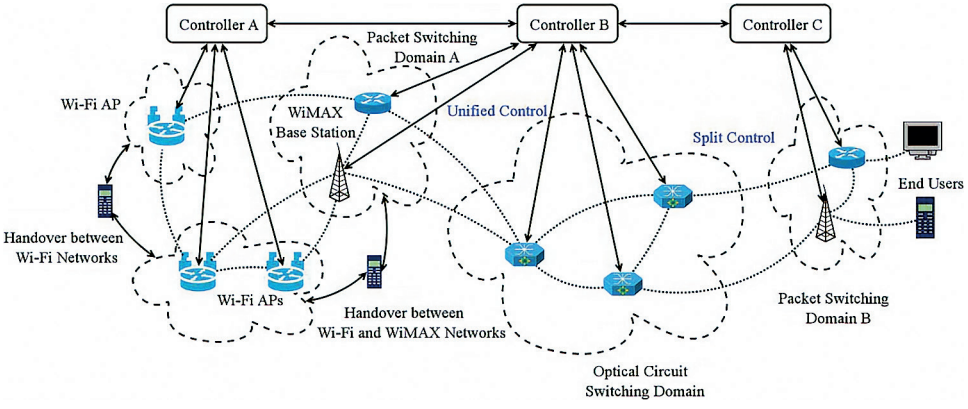
W rozdziale 2 i 3 krótko zaprezentowano wykorzystane technologie jako krótki wstęp teoretyczny. Celem rozdziału 4 jest przedstawienie przeglądu literatury dotyczącej wykorzystania RL w kontekście SDN. Analiza dostępnej literatury pozwoli na zidentyfikowanie głównych wyzwań oraz przyszłych kierunków badań w tej interdyscyplinarnej dziedzinie. Podsumowanie przeglądu literatury pozwoli na wyciągnięcie wniosków dotyczących skuteczności zastosowania RL w SDN oraz wskaże potencjalne kierunki dalszych badań, które mogą przyczynić się do jeszcze lepszego wykorzystania tej technologii w praktyce. W projektowej części artykułu zaprezentowano możliwość nauki sieci neuronowej na bazie uczenia ze wzmocnieniem, gdzie sieć neuronowa poszukuje optymalnej konfiguracji spełniającej wymagania postawione przed siecią SDN. Jako przykład takiego działania przedstawiono problem poszukiwania najkrótszej ścieżki w sieci.

CHARAKTERYSTYKA SIECI STEROWANYCH PROGRAMOWO

Sieci sterowane programowo (ang. *Software-Defined Networking*, w skrócie SDN) to koncepcja sterowania siecią, która opiera się na oddzieleniu

warstwy sterowania od warstwy transportowej. Urządzenia sieciowe przy użyciu takiej koncepcji mają za zadanie jedynie przesył danych. Wszystkie instrukcje płyną z głównego elementu w SDN, czyli inteligentnego kontrolera, który zarządza oraz steruje siecią. Za jego pomocą możliwe jest programowe zarządzanie zachowaniem sieci, a także monitorowanie i diagnozowanie jej stanu. Poprzez centralizację zarządzania SDN pozwala na kontrolę całego środowiska, co przekłada się na efektywność operacyjną, optymalizację wykorzystania zasobów oraz zwiększenie elastyczności i bezpieczeństwa sieci. Komunikacja między kontrolerem oraz urządzeniami jest możliwa za pomocą protokołu, którego najbardziej rozpowszechnionym przykładem jest protokół OpenFlow [1].

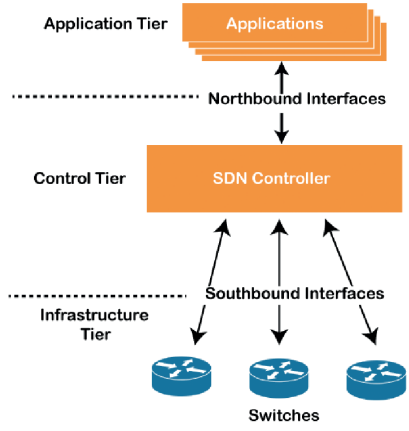
Rysunek 1. Przykład infrastruktury SDN ilustrujący jego ideę



Źródło: opracowano na podstawie [1], data dostępu: 18.08.2024.

Architektura SDN składa się z trzech głównych warstw: warstwy aplikacji, płaszczyzny sterowania oraz płaszczyzny danych. Każda z tych warstw pełni określone funkcje, które współpracują ze sobą w celu zapewnienia elastycznego i zcentralizowanego zarządzania siecią. Zobrazowaną architekturę SDN można zobaczyć na Rys. 2.

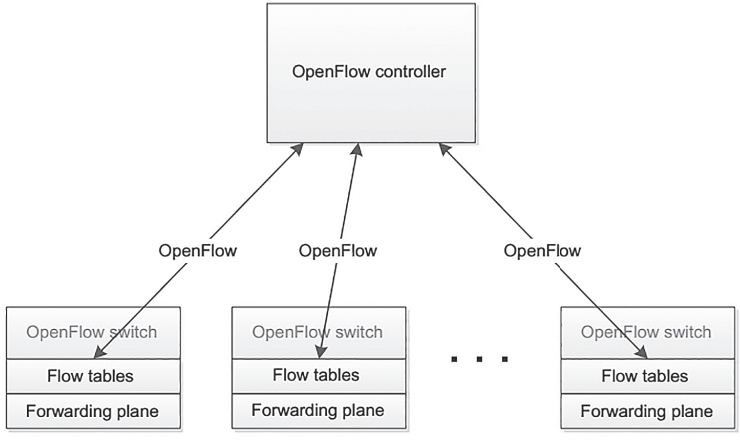
Rysunek 2. Architektura SDN



Źródło: opracowano na podstawie [2], data dostępu: 18.08.2024.

OpenFlow to protokół komunikacyjny wykorzystywany między kontrolerem SDN a przełącznikami sieciowymi. OpenFlow definiuje sposób, w jaki kontroler może zdalnie programować tablice przekazywania danych w przełącznikach kontrolując przy tym przepływ pakietów w sieci [1]. Rys. 3 ilustruje ogólną architekturę protokołu OpenFlow.

Rysunek 3. Ogólny schemat działania protokołu OpenFlow



Źródło: opracowano na podstawie [3], data dostępu: 18.08.2024.

CHARAKTERYSTYKA UCZENIA PRZEZ WZMACNIANIE

Uczenie przez wzmacnianie (ang. *Reinforcement learning*, w skrócie RL) czerpie inspirację z psychologii, gdzie proces uczenia się człowieka oparty jest na doświadczeniach i popełnianiu błędów. Leży ono pomiędzy uczeniem nadzorowanym a nienadzorowanym, jest to definitywnie ambitniejszy i najnowocześniejszy sposób uczenia niż wcześniej wymienione propozycje. W tego rodzaju podejściu wykorzystywany jest agent, który podejmuje decyzje w środowisku i w zależności od wyniku tych działań, jest nagradzany lub karany (Rys. 4). Kara zdefiniowana jest jako negatywna nagroda. Nagroda jest najważniejszym elementem w uczeniu przez wzmacnianie, pozwala ona wzmacniać lub zmieniać zachowanie agenta. Jest ona odzwierciedleniem niedawno wykonanych akcji agenta. Środowisko, w którym działa agent, jest kluczowym elementem tego procesu i zazwyczaj definiowane jest zarówno dla treningu sieci, jak i dla testowania jej skuteczności. Agent i środowisko komunikują się za pomocą nagrody oraz obserwacji środowiska.

Agent posiada [4]:

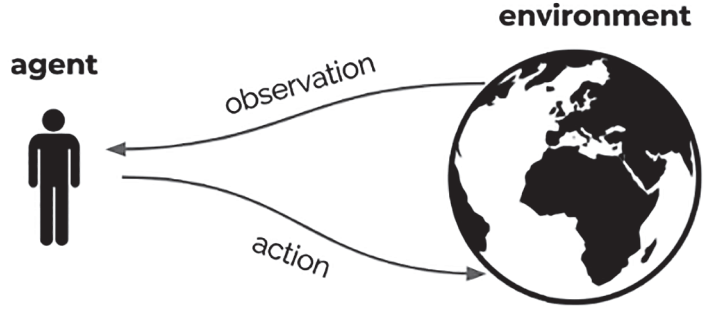
- Politykę czyli zbiór zasad, który ogranicza zachowania agenta,
- Hiperparametry, które mogą pomagać w regulacji procesu uczenia.

Środowisko posiada:

- Swój stan,
- Krok będący funkcją, która bazując na akcji agenta zmienia stan środowiska oraz zwraca nagrodę,
- Epizod składający się ze zbioru kroków, po których stan środowiska jest resetowany,
- Nagrodę jako zmienną zwracaną po wykonaniu akcji,
- Obserwację opisującą stan środowiska w postaci macierzy, wektora lub skalara.

Uczenie przez wzmacnianie to uczenie aktywne, w którym poprzednie akcje wpływają na przyszłe wybory agenta. Dodatkowym atutem tego rozwiązania jest fakt, że nie potrzebujemy przykładów poprawnych wyborów lub zachowań.

Rysunek 4. Pętla interakcji między agentem a środowiskiem



Źródło: opracowano na podstawie [5], data dostępu: 18.08.2024.

Procesy Markowa (ang. *Markov process*) wykorzystywane są w RL do modelowania problemów jak i w szeroko pojętej dziedzinie inżynierii. Głównym założeniem jest to, że istnieje system, na który my jako obserwatorzy nie możemy wpływać. System posiada stany, między którymi może przechodzić z pewną logiką. Przestrzenią stanów nazywamy wszystkie stany systemu, są one zbiorem skończonym. Obserwator jest w stanie stworzyć łańcuchy stanów. Bardzo ważną własnością, którą posiadają procesy Markowa jest to, że stany przyszłe muszą być definiowane tylko przez stan aktualny. Jest to własność Markowa [6].

PRZEGLĄD LITERATURY

Tabela 1. Wybrane artykuły

Źródło	Rok publikacji	Osiągnięcia i kierunki dalszych badań
[7]	2019	Kompleksowy przegląd algorytmów uczenia maszynowego w SDN, obejmujący klasyfikację ruchu, optymalizację routingu, przewidywanie QoS i zarządzanie zasobami.
[8]	2020	System obrony sieciowej oparty na RL, wykrywający i łagodzący ataki na IoT w czasie rzeczywistym, skutecznie minimalizujący wpływ ataków w małych sieciach prywatnych

[9]	2021	Wykrywanie i łagodzenie ataków DDoS oraz zarządzanie nagłymi wzrostami obciążenia w SDN za pomocą wieloagentowego RL, wykazując lepsze wyniki w opóźnieniu, jitterze i utracie pakietów.
[10]	2021	Analiza roli DRL w zarządzaniu SDN-NFV w Edge-IoT, podkreślająca znaczenie tych technologii w poprawie wydajności i efektywności zarządzania sieciami.
[11]	2021	Omawia i porównuje techniki uczenia maszynowego w kontekście routingu, podkreślając potrzebę dalszych badań w tym zakresie.
[12]	2023	Przedstawiają nowatorskie podejście do konsolidacji maszyn wirtualnych w SDN z NFV, wykorzystujące DRL oparte na transformerach, co zwiększa efektywność energetyczną i optymalizuje rozmieszczenie funkcji sieciowych jako VM. Metoda ta skraca czas wnioskowania i treningu, zachowując wysoką jakość usług
[13]	2023	Opis algorytmu trasowania wykorzystujący wieloagentowe, wielozadaniowe RL, zapewniający lepszą skalowalność i QoS dla różnych rodzajów ruchu. Implementacja w kontrolerze Ryu wykazała skuteczność w adaptacji do awarii łączy, zmian w topologii i wahań ruchu.
[14]	2023	Opis podejścia do inżynierii ruchu w hybrydowych sieciach SDN, łączące uczenie samonadzorowane i RL, co poprawia równoważenie obciążenia w porównaniu z nowoczesnymi metodami.
[15]	2023	Opis podejścia do kontroli zatorów w SDN za pomocą MARL, wykorzystującego algorytm Q-learning do adaptacji do zmieniających się warunków sieciowych.

Źródło: opracowanie własne.

Wybrane artykuły zostały zaprezentowanie w Tab. 1, uporządkowane względem roku publikacji. Podsumowanie przeglądu literatury wskazuje, że zastosowanie zaawansowanych technik RL w zarządzaniu sieciami SDN przynosi liczne korzyści. Techniki te zwiększają efektywność energetyczną i operacyjną oraz pozwalają na lepsze dostosowanie do dynamicznie zmieniających się warunków sieciowych. Inteligentne mechanizmy zarządzania zasobami redukują potrzebę manualnej interwencji, co obniża koszty operacyjne i zwiększa skalowalność oraz elastyczność sieci. Przyszłe badania powinny koncentrować się na dalszej optymalizacji tych technik, zwiększeniu ich skalowalności oraz ocenie ich efektywności w rzeczywistych środowiskach produkcyjnych. Potencjał uczenia wzmacniającego w kontekście SDN i IoT jest ogromny, a dalszy rozwój tych technologii może znacząco wpłynąć na przyszłość obszaru zarządzania sieciami.

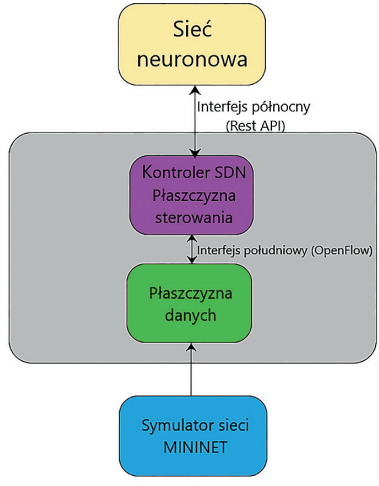
PROJEKT ZASTOSOWANIA UCZENIA MASZYNOWEGO W SDN

Ogólny schemat i topologia

Za pomocą REST API z Floodlight wczytywana jest struktura oraz parametry sieci. Na podstawie tych danych stworzono model do nauki sieci neuronowej, następnie sieć została poddana uczeniu, w wyniku czego znajduje optymalną ścieżkę w zadanej jej sieci. Ostateczny wynik działania sieci neuronowej zostaje następnie przekształcony w konfigurację, która jest przesyłana z powrotem za pomocą REST API do Floodlight.

Proponowany system (Rys. 5) zawiera sieć neuronową. Ma ona za zadanie wyznaczyć najkrótszą ścieżkę, którą przesłany będzie ruch. Komunikuje się ona z kontrolerem SDN za pomocą interfejsu północnego, a dokładnie REST API. Komunikacja między kontrolerem SDN, a przełącznikami jest możliwa za pomocą interfejsu południowego. Wybrany do tego został standardowy protokół komunikacyjny OpenFlow. W trakcie konfiguracji początkowej projektu za pomocą narzędzia Mininet przesyłana jest wcześniej zdefiniowana topologia sieci do kontrolera SDN.

Rysunek 5. Architektura systemu



Źródło: opracowanie własne.

Realizacja projektu sieci SDN z siecią neuronową uczoną przez wzmocnienie

Emulacja sieci – konfiguracja sieci w Mininet

Testowaną topologię sieci uruchamiano za pomocą narzędzia Mininet oraz specjalnej komendy, w której jako parametry określamy, że kontroler SDN nie działa lokalnie, jego adres IP, port na którym kontroler będzie się komunikował, ścieżkę pod którą można znaleźć niestandardową topologię, użycie niestandardowej topologii „mytopo", użycie narzędzia „Traffic Control", przepustowość łączy oraz opóźnienie na łączu.

Wczytanie danych z SDN oraz przygotowanie danych

Za pomocą własnego skryptu wczytującego i przygotowującego oraz REST API dane z sieci SDN są pobierane do środowiska sieci neuronowej. Skrypt wykonuje szereg operacji, aby pobrać i przetworzyć dane dotyczące sieci zarządzanej przez kontroler SDN. Kod efektywnie przetwarza dane dotyczące topologii sieci SDN, tworzy reprezentację macierzową połączeń i identyfikuje kluczowe węzły końcowe w sieci w ten sposób przygotowując dane do pracy sieci neuronowej.

Projekt i implementacja głębokiej sieci DQN

W projekcie zdefiniowane jest środowisko dla uczenia się maszynowego z użyciem biblioteki GYM. Na początku sprawdzana jest poprawność danych wejściowych otrzymanych z SDN, a potrzebnych do nauki sieci neuronowej i przygotowywane całe środowisko. W trakcie nauki będzie tworzony zapis tego procesu w postaci zapamiętania prawdopodobieństw akcji modelu. W tym celu tworzony jest jednowymiarowy wektor dla węzła startowego, a następnie jest zapisywane przewidywane prawdopodobieństwo dla każdej akcji do pliku CSV po każdej epoce nauki.

Istotnym elementem implementacji jest zdefiniowana w projekcie funkcja „create_model" pokazana na Rys. 6, która tworzy model sieci neuronowej. W modelu zdefiniowane są trzy warstwy, w tym dwie ukryte z aktywacją ReLU, a trzecia wyjściowa z aktywacją softmax, która konwertuje wyjście na prawdopodobieństwa sumujące się do 1. Używany jest optymalizator Adam oraz funkcja straty „categorical_crossentropy", standardowej funkcji strat dla problemów klasyfikacji wieloklasowej. Argumenty funkcji to liczba wejściowych cech, czyli wymiary wejściowe, liczba wyjściowych klas, czyli wymiary wyjścia oraz współczynnik uczenia dla optymalizatora Adam, domyślnie ustawiony na wartość 0.01. Tworzony

jest model sekwencyjny, który pozwala na dodawanie warstw w sposób liniowy. Funkcja zwraca skonfigurowany model.

Rysunek 6. Tworzenie modelu sieci neuronowej

```
def create model(input dim, output dim, learning rate=0.01):
    model = keras.Sequential(
        [
            layers.Dense(64, input_dim=input_dim, activation='relu'),
            layers.Dense(64, activation='relu'),
            layers.Dense(output_dim, activation='softmax')
        ])
    model.compile(optimizer=Adam(learning_rate=learning_rate),
                  loss='categorical crossentropy')
    return model
```

Źródło: opracowanie własne.

Bardzo ważnym krokiem w projekcie jest trenowanie modelu sieci neuronowej w środowisku (Rys. 7). Argumentem funkcji jest środowisko oraz liczba epizodów treningowych, domyślnie ustawiona na wartość 1000. Na początku inicjalizowane są zmienne odpowiadające liczbie węzłów w sieci oraz wymiarom wejścia i wyjścia, które są równe właśnie liczbie węzłów. Następnie tworzony jest model sieci neuronowej za pomocą funkcji „create_model". Za pomocą pętli funkcja przechodzi przez każdy epizod treningowy, na początku pętli środowisko jest resetowane do stanu początkowego. Kolejnym etapem jest pętla przez stany w jednym epizodzie. Pętla trwa do momentu aż bieżący węzeł nie będzie węzłem końcowym. Model przewiduje prawdopodobieństwa dla każdej akcji bieżącego stanu i zapisuje je. Następnie prawdopodobieństwa dla niepodłączonych węzłów są zerowane, a zmodyfikowane prawdopodobieństwa są sumowane. Jeśli ta suma jest różna od zera to prawdopodobieństwa są normalizowane. Kolejny węzeł jest wybierany na podstawie prawdopodobieństw akcji. Następnie wyliczana jest nagroda oraz aktualizowany jest stan.

Model trenowany jest następnie przez jedną epokę, po czym aktualizowany jest bieżący stan oraz węzeł. Na końcu funkcja zwraca wytrenowany model.

Rysunek 7. Trenowanie modelu sieci neuronowej

```
def learning model(env, num episodes=1000):
    num nodes = env.num nodes
    input dim = num nodes
    output_dim = num_nodes
    model = create_model(input_dim, output_dim)

    for ep in range(num_episodes):
        print('.............. Episode: ', ep, ' ..............')
        state = env.reset()

        while env.current_node != env.end_node:
            action_probs = model.predict(state, verbose=0, workers=4,
               use_multiprocessing=True)[0]
            save_action_probs(model, env, ep)

            # It cannot stay at the same node or if there is no
            # connection between nodes
            tmp action probs = action probs.copy()
            for i in range(num_nodes):
                if env.net[env.current node, i] == 0:
                    tmp_action_probs[i] = 0
            sum_row = np.sum(tmp_action_probs)
            if sum_row != 0:
                tmp_action_probs /= sum_row
            else:
                print('Continue')
                env.current_node = env.end_node
                continue
            action_probs = tmp_action_probs.copy()

            # Action
            next_node = np.random.choice(range(num_nodes),
                                         p=action probs)

            # Calculation of the reward
            reward = env.net[env.current_node, next_node]

            # Update of the state
            next_state = np.zeros((1, num_nodes))
            next_state[0, next_node] = 1

            # Training a model
            target = np.zeros((1, num nodes))
            target[0, next_node] = reward

            print('FIT: current node: ', env.current_node,
                  ' next node: ', next_node)

            model.fit(state, target, epochs=1, verbose=0, workers=4,
               use multiprocessing=True)

            state = next_state
            env.current_node = next_node

    return model
```

Źródło: opracowanie własne.

Po zakończeniu tego procesu znajdowana jest ścieżka od węzła początkowego do węzła końcowego w środowisku symulacyjnym za pomocą wytrenowanego modelu sieci neuronowej. Taki wynik jest z kolei przygotowywany do powrotnego wysłania do sieci SDN.

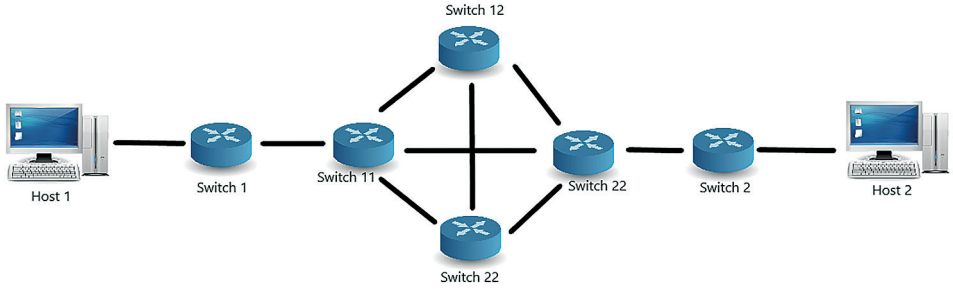
Transfer wyników do SDN

Aby odesłać wyniki działania sieci neuronowej używany jest skrypt wyjściowy, który dodaje statyczne przepływy do sieci SDN na podstawie najkrótszej ścieżki między węzłami. Dane uzyskane z sieci neuronowej są tłumaczone na postać potrzebną do zaprogramowania (modyfikacji konfiguracji) sieci komputerowej i wysyłane do kontrolera SDN.

Topologia testowa

W pierwszym przypadku wykorzystano topologię sieci przedstawioną na Rys. 8. Topologia ta została indywidualnie wygenerowana za pomocą skryptu napisanego w języku Python. Sieć składa się z dwóch hostów oraz sześciu przełączników. Jest to topologia testowa wybrana aby w sposób czytelny zaprezentować działanie proponowanego rozwiązania. Przełączniki środkowe są ze sobą połączone na zasadzie „każdy z każdym", a dwa przełączniki zewnętrzne tworzą połączenie z hostami.

Rysunek 8. Topologia sieci



Źródło: opracowanie własne.

Topologia badanej sieci SDN:

- Liczba hostów: 2
- Liczba przełączników: 6

- Liczba połączeń sieciowych: 11

Parametry sieci neuronowej:

- Liczba warstw sieci: 3
- Warstwy sieci:
  - Gęsta o wymiarze 64 i funkcją aktywacji „relu”
  - Gęsta o wymiarze 64 i funkcją aktywacji „relu”
  - Gęsta o wymiarze 16 i funkcją aktywacji „softmax”
  - Parametr „learnin rate” = 0,01
  - Funkcja „loss” = „ategorical_crossentropy”

Wyniki testów

Z wykresów pokazanych na Rys. 9 i 10 oraz Tab. 2 wynika, że liczba epok po których sieć się nauczyła wynosi 20.

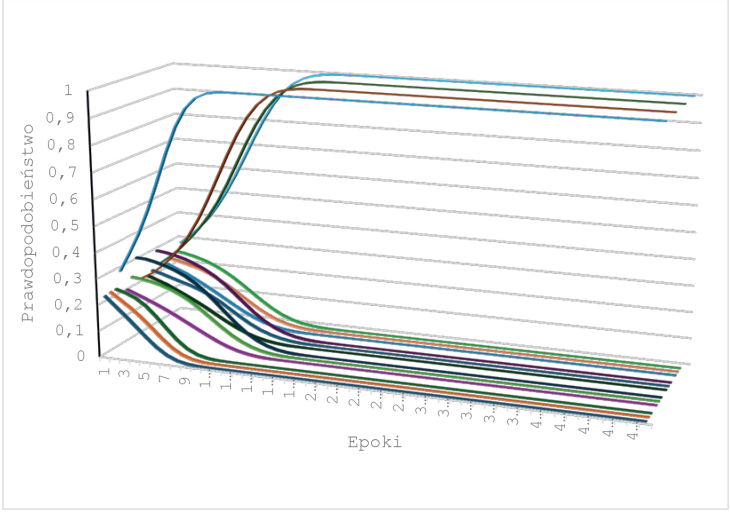
Tabela 2. Wartości „action probes” po każdej epoce

E	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16
0	0,233	0,236	0,236	0,295	0,212	0,249	0,315	0,223	0,223	0,232	0,301	0,244	0,226	0,235	0,256	0,282
1	0,198	0,209	0,225	0,368	0,197	0,245	0,313	0,245	0,209	0,222	0,296	0,273	0,210	0,226	0,250	0,314
2	0,164	0,182	0,209	0,445	0,181	0,238	0,305	0,275	0,195	0,209	0,287	0,308	0,194	0,216	0,241	0,349
3	0,127	0,151	0,182	0,540	0,165	0,229	0,293	0,312	0,177	0,198	0,277	0,349	0,174	0,206	0,229	0,391
4	0,090	0,115	0,146	0,649	0,147	0,216	0,280	0,358	0,158	0,186	0,262	0,394	0,153	0,189	0,214	0,444
5	0,056	0,078	0,104	0,762	0,128	0,198	0,261	0,413	0,139	0,172	0,243	0,446	0,131	0,169	0,194	0,506
6	0,030	0,046	0,064	0,861	0,108	0,176	0,237	0,479	0,118	0,154	0,221	0,507	0,107	0,146	0,171	0,576
7	0,013	0,023	0,034	0,931	0,087	0,148	0,206	0,559	0,095	0,133	0,192	0,580	0,084	0,119	0,143	0,653
8	0,005	0,009	0,015	0,971	0,067	0,118	0,170	0,645	0,073	0,107	0,158	0,662	0,062	0,093	0,114	0,731
9	0,001	0,003	0,006	0,989	0,047	0,089	0,132	0,732	0,052	0,081	0,123	0,743	0,043	0,068	0,086	0,804
10	0,000	0,001	0,002	0,997	0,031	0,062	0,095	0,812	0,035	0,058	0,090	0,818	0,027	0,046	0,060	0,867
11	0,000	0,000	0,001	0,999	0,019	0,040	0,063	0,878	0,022	0,038	0,061	0,879	0,016	0,029	0,038	0,917
12	0,000	0,000	0,000	1,000	0,010	0,023	0,039	0,928	0,013	0,024	0,039	0,925	0,009	0,016	0,023	0,952
13	0,000	0,000	0,000	1,000	0,005	0,012	0,021	0,961	0,007	0,014	0,023	0,957	0,004	0,009	0,013	0,974
14	0,000	0,000	0,000	1,000	0,002	0,006	0,011	0,981	0,003	0,007	0,012	0,977	0,002	0,004	0,006	0,987
15	0,000	0,000	0,000	1,000	0,001	0,003	0,005	0,992	0,001	0,003	0,006	0,989	0,001	0,002	0,003	0,994
16	0,000	0,000	0,000	1,000	0,000	0,001	0,002	0,997	0,001	0,002	0,003	0,995	0,000	0,001	0,001	0,998
17	0,000	0,000	0,000	1,000	0,000	0,000	0,001	0,999	0,000	0,001	0,001	0,998	0,000	0,000	0,001	0,999
18	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	0,999	0,000	0,000	0,000	1,000
19	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
20	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
21	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
22	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
23	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
24	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
25	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
26	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
27	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
28	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
29	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
30	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000

31	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
32	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
33	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
34	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
35	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
36	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
37	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
38	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
39	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
40	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
41	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
42	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
43	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
44	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
45	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
46	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
47	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
48	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000
49	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	1,000

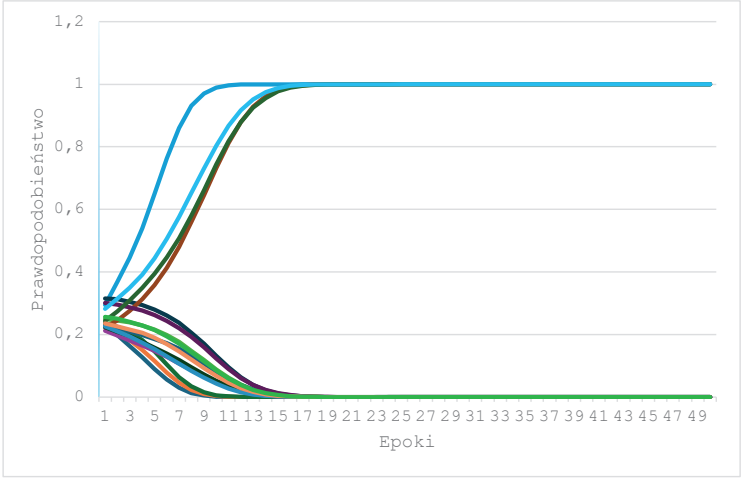
Źródło: opracowanie własne.

Rysunek 9 Wykres przestrzenny wartości „action probes” po każdej epoce



Źródło: opracowanie własne.

Rysunek 10 Wykres płaski wartości „action probes" po każdej epoce



Źródło: opracowanie własne.

Interpretacja wyników testów

W analizowanym przypadku sieć neuronowa wykazywała stabilne zachowanie oraz charakteryzowała się szybkim procesem uczenia, co można zaobserwować na podstawie faktu, że osiągała stan nasycenia po kilkudziesięciu epokach. Na tej podstawie można wywnioskować, że dobrane parametry sieci neuronowej były optymalne i nie wymagają korekt. Jednakże, w przypadku większych rozmiarów sieci SDN, może zaistnieć potrzeba wprowadzenia pewnych modyfikacji parametrów. Analiza wartości „action probes" pokazuje, że początkowe wartości były zróżnicowane, a w miarę kolejnych epok sieć neuronowa dążyła do jednolitych wyników bliskich wartości 1, co wskazuje na skuteczne uczenie się i minimalizację błędów przewidywania. Wyniki sugerują, że sieci neuronowe mogą być skutecznym narzędziem do przewidywania i zarządzania ruchem w sieciach SDN. Model może być zastosowany do monitorowania sieci, optymalizacji routingu i zapobiegania przeciążeniom. Ze względu na szybkie uczenie się modelu, sieci neuronowe mogą być stosowane w dynamicznych środowiskach, gdzie warunki sieciowe mogą się szybko zmieniać. Model może adaptować się do nowych wzorców ruchu, co jest kluczowe dla wydajnego zarządzania siecią. Choć badania przeprowa-

dzono na stosunkowo małej sieci, metody i wyniki sugerują, że technika ta może być skalowana do większych i bardziej złożonych sieci SDN, co otwiera drogę do dalszych badań i implementacji w realnych środowiskach produkcyjnych.

PODSUMOWANIE

Celem pracy była analiza metod elastycznego sterowania sieciami SDN oraz opracowanie własnego rozwiązania, które umożliwi inteligentne dostosowanie pracy kontrolera SDN, co zostało zrealizowane. W ramach projektu zastosowano uczenie przez wzmacnianie, które pozwala na autonomiczne podejmowanie decyzji przez sieć uczącą się na podstawie swoich wyborów w dynamicznie zmieniającym się środowisku, co umożliwia elastyczne sterowanie ruchem. Praca miała na celu poprawę wydajności, elastyczności oraz zdolności adaptacyjnych sieci w czasie rzeczywistym.

W analizowanym przypadku sieć neuronowa wykazywała stabilne zachowanie oraz charakteryzowała się szybkim procesem uczenia, co można zaobserwować na podstawie faktu, że osiągała stan nasycenia po kilkudziesięciu epokach. Na tej podstawie można wywnioskować, że dobrane parametry sieci neuronowej były optymalne i nie wymagają korekt. Jednakże, w przypadku większych rozmiarów sieci SDN, może zaistnieć potrzeba wprowadzenia pewnych modyfikacji parametrów.

Zastosowanie uczenia maszynowego w SDN umożliwiło uzyskanie nowych możliwości elastycznego sterowania ruchem. Sieć neuronowa sprawnie i skutecznie odszukiwała optymalne rozwiązanie problemów zadanych topologii. Uczenie maszynowe znajduje także zastosowanie w SDN w rozwiązywaniu nowych, dotychczas nie rozważanych problemów w sieciach SDN, jeśli tylko użytkownik jest w stanie zdefiniować problem i zapisać go w postaci wytycznych do nauki sieci. Sieć neuronowa jest w stanie sprawnie dokonywać modyfikacji SDN analizując i wyszukując najlepsze rozwiązania przy zmiennych wymaganiach i dynamicznie zmieniającym się stanie sieci SDN.

Przyszłe prace mogą skupić się na dalszym dostosowywaniu parametrów sieci neuronowych dla większych i bardziej złożonych sieci SDN, aby jeszcze bardziej zwiększyć ich wydajność i adaptacyjność. Istnieje więc możliwość dzięki temu uzyskania jeszcze większej elastyczności sieci. Można również zbadać integrację innych technik uczenia maszynowego z proponowanym rozwiązaniem, aby poprawić jego efektywność.

Dodatkowo, warto przeprowadzić testy w rzeczywistych środowiskach sieciowych, aby zweryfikować praktyczną użyteczność i skalowalność opracowanego systemu.

LITERATURA

1. Xia W., Wen Y., Foh C.H., Niyato D., Xie H. *„A Survey on Software-Defined Networking”,* IEEE Communications Surveys & Tutorials, 2015.
2. „javatpoint”, 18 08 2024. [Online]. Available: https://www.javatpoint.com/software-defined-networking-sdn-benefits-and-challenges--of-network-virtualization.
3. Göransson P., Black C., Culver T. *Software Defined Networks A Comprehensive Approach,* Cambridge: Todd Green, 2014.
4. R.S. Sutton, Barto A.G. *Reinforcement Learning,* Cambridge, Massachusets: Westchester Publishing Services, 2020.
5. Hasselt H.V. *„Reinforcement Learning, Lecture 1: Introduction,”* 2021.
6. Lapan M., *Deep Reinforcement Learning Hands-On: Apply modern RL methods to practical problem of chatbots, robotics, discrete optimization, web automation and more, 2nd Edition*, Packt Publishing, 2020.
7. Xie J., Yu F.R., Huang T., Xie R., Liu J., Wang C., Liu Y. *„A Survey of Machine Learning Techniques Applied to Software Defined Networking (SDN): Research Issues and Challenges”,* IEEE COMMUNICATIONS SURVEYS & TUTORIALS, tom 21, nr 1, 2019.
8. Zolotukhin M., Kumar S., Hämäläinen T. *„Reinforcement Learning for Attack Mitigation in SDN-enabled Networks”,* IEEE, Ghent, 2020.
9. Dake D.K., Gadze J. D., Klogo G. S. *„DDoS and Flash Event Detection in Higher Bandwidth SDN-IoT using Multiagent Reinforcement Learning”,* IEEE, 2021.
10. Alonso R.S., Prieto J., La Prieta F. de, Rodríguez-González S., Corchado J. M. *„Edge-IoT, A Review on Deep Reinforcement Learning for the management of SDN and NFV”,* IEEE, Madrid, 2021.
11. AminR. , RojasE. , Aqdus A., Ramzan S., Casillas-Perez D., Arco J.M. *„A Survey on Machine Learning Techniques for Routing Optimization in SDN”,* IEEE Access, 2021.
12. Jeong E.-D., Yoo J.-H., Hong J.W.-K. *„VM Consolidation for SDN using Transformer-Based Deep Reinforcement Learning”,* IEEE, 2023.
13. Tong W., LijunW. *„Intelligent Online Routing in SDN via Multi-Task Multi-Agent Reinforcement Learning”,* IEEE, 2023.

14. TangQ., Yang R., Guo Y. *„Reinforcement Learning with Contrastive Unsupervised Representations for Traffic Engineering in Hybrid SDN”*, IEEE, 2023.
15. Boussaoud K., Ayache M., En-Nouaary A. *„A Multi-Agent Reinforcement Learning Approach for Congestion Control in network based-SDN”*, IEEE, 2023.