



A G H

AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
Wydział Informatyki, Elektroniki i Telekomunikacji

Instytut Telekomunikacji

Praca dyplomowa

Porównanie parametrów rozproszonej sieci komputerowej z
parametrami sieci SDN przy wystąpieniu zdarzeń
zaplanowanych lub losowych.

Comparison of the parameters of a distributed computer network to
the SDN network in cases of scheduled or random events.

Autor: Kacper Handzlik
Kierunek studiów: Teleinformatyka
Opiekun pracy: dr hab. inż. Jerzy Domążał

Kraków, 2024

Spis treści

1 Teoria	4
1.1 Wprowadzenie	4
1.2 SDN	4
1.3 Multipathing	5
1.4 Mininet	6
1.5 Sterownik Ryu	6
2 Badania	9
2.1 Środowisko badawcze	9
2.1.1 Sieć rozproszona	9
2.1.2 Sieć SDN	9
2.2 Przeprowadzenie badań	10
2.2.1 Sieć rozproszona	10
2.2.2 Sieć SDN	13
3 Wyniki	19
3.1 Wyniki - sieć rozproszona	19
3.1.1 Scenariusz 1	19
3.1.2 Scenariusz 2	21
3.1.3 Scenariusz 3	23
3.2 Wyniki - SDN	25
3.2.1 Scenariusz 1	26
3.2.2 Scenariusz 2	28
3.2.3 Scenariusz 3	30
4 Wnioski	33

Słowniczek pojęć

pozycje przepływów Instrukcje mówiące co ma zrobić przełącznik z pakietem. 4

przepływ Sekwencja pakietów o takich samych atrybutach. Krotka atrybutów: (adres źródłowy, adres docelowy, port źródłowy, port docelowy, protokół. 4

RL Reinforcement Learning - algorytm uczenia maszynowego ze wzmocnieniem. 5, 6

sieci SDN (Software-Defined Networks) Programowalne Sieci Komputerowe, w tym dokumente dla uproszczenia będą określane sieciami SDN. 4–6

sterownik sieci Program zawierający logikę zachowania się sieci. 4

tablice przepływów Tablice, w których umieszczane się pozycje przepływów. 4

Rozdział 1

Teoria

1.1 Wprowadzenie

Na przestrzeni ostatnich lat wiele wydarzyło się w kwestii technologii SDN. Z uwagi na swoją uniwersalność, znajduje ona swoje zastosowanie w wielu dziedzinach związanych z tradycyjnymi sieciami. Sieci SDN z uwagi na scentralizowany sposób zarządzania, posiadają większą świadomość o wyglądzie sieci. Pozwala to na większą automatyzację sieci, dzięki czemu ogranicza się wpływ człowieka oraz związane z nim potencjalne błędy. Sieci SDN znalazły swoje zastosowanie w sieciach WAN [1], Wi-Fi [2], sieciach komórkowych [3] czy systemach IoT [4] oraz ze względu na swoją elastyczność mogą być odpowiednio dostosowywane np. pod względem optymalizacji ruchu w sieci poprzez wyznaczanie wielu tras dla pary komputerów [5] (źródło-cel), czy np. wydajności energetycznej [6]. Przez to, że zarządzanie siecią może odbywać się w niezależnym środowisku, możliwa jest łatwiejsza integracja sterownika sieci z modelami uczenia maszynowego [7]. Zaletami korzystania z takich sieci jest mnogość doboru sterowników napisanych w różnych technologiach oraz łatwość ich aktualizacji, bez konieczności oczekiwania na najnowszą wersję oprogramowania od producenta. Na rynku dostępne są rozwiązania zarówno otwartoźródłowe dla przełączników np. Open vSwitch [8] jak i dla sterowników np. Ryu [9], które wykorzystano w tej pracy. Dostępna jest również duża oferta komercyjnych rozwiązań od międzynarodowych producentów: HP Virtual Connect, Cisco Nexus 1000V, Juniper Contrail Networking.

Cel pracy: Celem pracy jest porównanie możliwości rozproszonych sieci komputerowych z sieciami SDN.

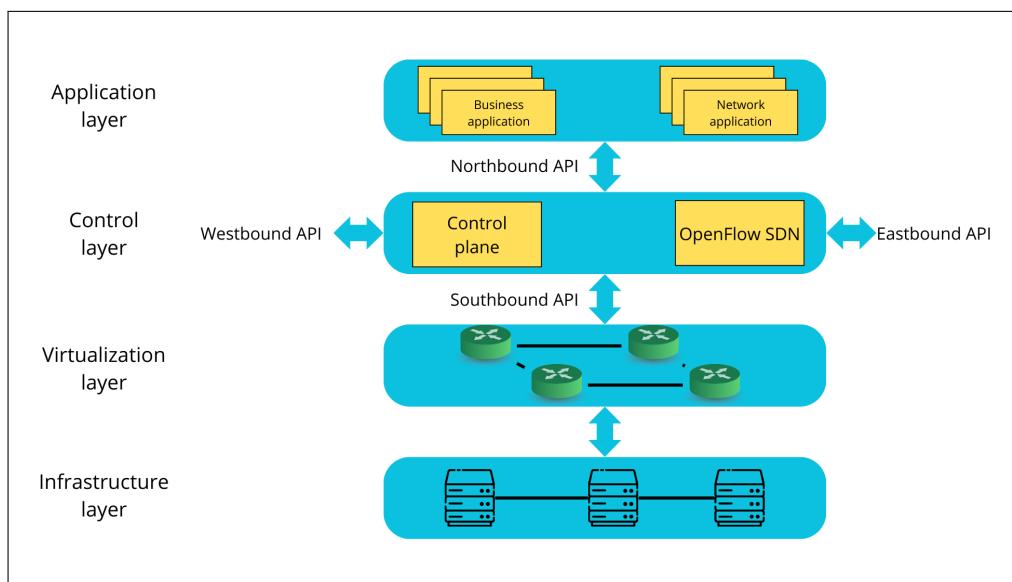
1.2 SDN

Sieci SDN buduje się przy pomocy przełączników SDN oraz sterowników sieci. Przełączniki SDN swoim działaniem przypominają przełączniki stosowane w rozproszonych sieciach komputerowych, natomiast wyróżniają się tym, że posiadają tablice przepływów działające podobnie do tablic rutingu w tradycyjnych ruterach. Tablice przepływów posiadają pozycje przepływów, na podstawie których przełącznik dokonuje wyboru na jaki interfejs przekazać przychodzący przepływ. Takie tablice aktualizowane są przez sterownik sieci, który w przeciwnieństwie do przełączników posiada logikę pozwalającą mu na zarządzanie całą siecią. Kroki instalacji pozycji przepływu:

1. do przełącznika przychodzi pakiet, którego przełącznik nie potrafi obsłużyć,
2. przełącznik wysyła wiadomość do sterownika o pojawienniu się przepływu,

3. sterownik ze świadomością całej topologii wysyła instrukcje (pozycja przepływów) na jaki port wysłać pakiet do przełącznika,
4. pozycja przepływów instalowana jest w tablicy przepływów,
5. pakiet wysyłany jest przez wskazany interfejs.

Dzięki temu, że przełączniki sprowadza się praktycznie do roli urządzeń przenoszących dane, zmniejsza się złożoność budowy urządzenia tym samym koszty jego zakupu i budowy sieci w przeciwieństwie do ruterów. Dzięki rozdzieleniu warstwy danych (przełączniki SDN) od warstwy sterującej (sterowniki sieci) można precyzyjnie rozdysponować zadania poszczególnych warstw pomiędzy wyspecjalizowane urządzenia [10].



Rysunek 1.1: Architektura SDN według Guy Pujolle [11].

W książce [11] Guy Pujolle rozszerza architekturę sieci SDN (Rys. 1.1) rozbijając warstwę danych na warstwy wirtualizacji i infrastruktury oraz dodaje warstwę aplikacji. Wirtualizacja często jest stosowana w sieciach tradycyjnych, gdzie tworzone są wirtualne routery a w tym przypadku przełączniki. W warstwie aplikacji zawierają się programy pomagające sterownikowi w podejmowaniu decyzji o zachowaniu sieci. W warstwie sterującej przez *Westbound API* i *Eastbound API* sterownik może połączyć się z innym sterownikiem.

1.3 Multipathing

Algorytmy multipathingu w sieciach SDN (Software-Defined Networking) stanowią kluczowy element optymalizacji przepływu danych, umożliwiając efektywne wykorzystanie różnych ścieżek transmisji. Tradycyjne podejścia do rutingu często opierają się na jednej głównej ścieżce, co może prowadzić do przeciążeń i utraty efektywności w przypadku awarii lub nadmiernego obciążenia sieci.

W kontekście Sieci SDN, algorytmy multipathingu pozwalają na dynamiczne dostosowywanie trasy danych w czasie rzeczywistym, reagując na zmienne warunki sieciowe np. aktualna przepustowość łącza [12]. Dodatkowo, wykorzystanie uczenia maszynowego ze wzmacnieniem (RL) staje się coraz bardziej obiecującym podejściem. Algorytmy RL mogą

uczyć się optymalnych strategii decyzyjnych na podstawie interakcji z otoczeniem, co jest szczególnie cenne w dynamicznych środowiskach sieciowych.

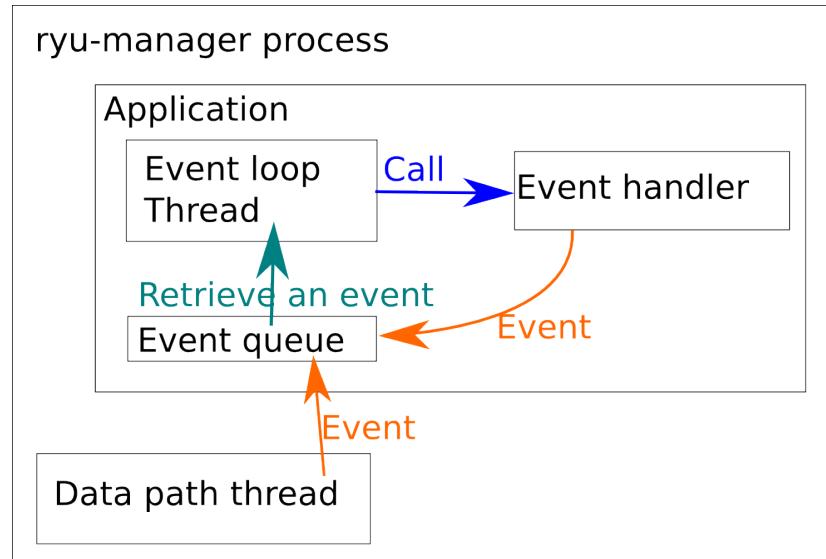
Uczenie maszynowe ze wzmacnieniem może być używane do dynamicznego dostosowywania parametrów algorytmów multipathingu, uwzględniając różne kryteria, takie jak opóźnienie, przepustowość czy koszty energetyczne. Dzięki temu Sieci SDN stają się bardziej elastyczne, efektywne i odporne na awarie, co przekłada się na zwiększoną wydajność i jakość usług świadczonych przez sieć. Współpraca algorytmów multipathingu i uczenia maszynowego stanowi innowacyjne podejście do zoptymalizowanej obsługi ruchu w nowoczesnych środowiskach sieciowych. Autorzy badania [13] wykazali, że algorytm obliczający polityki ruchu dla sieci z wykorzystaniem RL zmniejsza średnie opóźnienie o ok. 160 ms i zwiększa średnie wykorzystanie łącz o ok. 27 punktów procentowych względem algorytmu *Shortest Path*.

1.4 Mininet

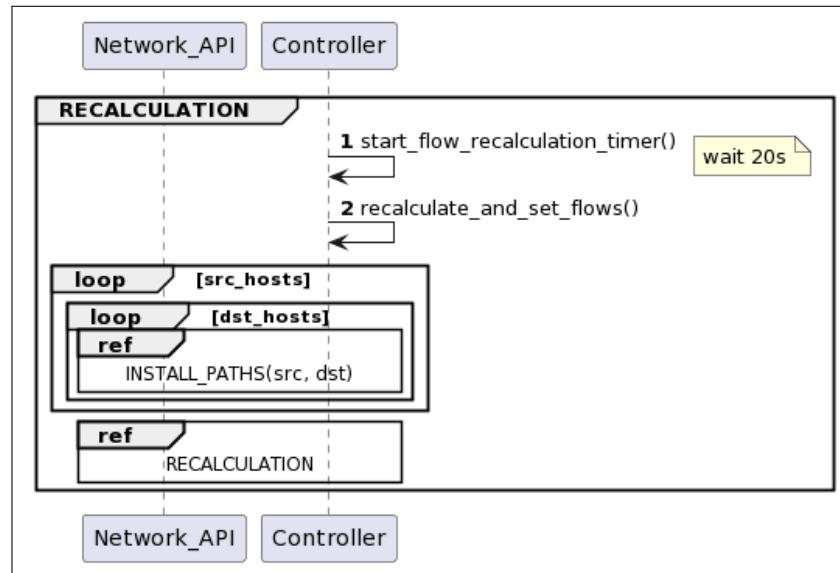
Mininet to otwartoźródłowe środowisko do emulacji sieci SDN w napisane w języku Python. Jego możliwości obejmują określenie m.in. rodzaju przełączników, parametrów połączeń między nimi, czy uruchamianie komend bezpośrednio z kodu. Mininet posiada również wiele predefiniowanych funkcjonalności, których użycie wymaga jedynie wprowadzenia komend. Po uruchomieniu sieci wyświetla się w terminalu wewnętrzny terminal mininetu, który pozwala na zarządzanie urządzeniami wewnętrz sieci. Przez to, że emulowane urządzenia współdzielą system operacyjny, możliwe jest uruchamianie na nich programów zainstalowanych w systemie środowiska. Ułatwia to w znacznym stopniu zarządzanie wieloma instancjami przełączników i hostów naraz. Mininet jest narzędziem, które służy do testowania funkcjonalności sterowników sieci w zamkniętym środowisku.[14]

1.5 Sterownik Ryu

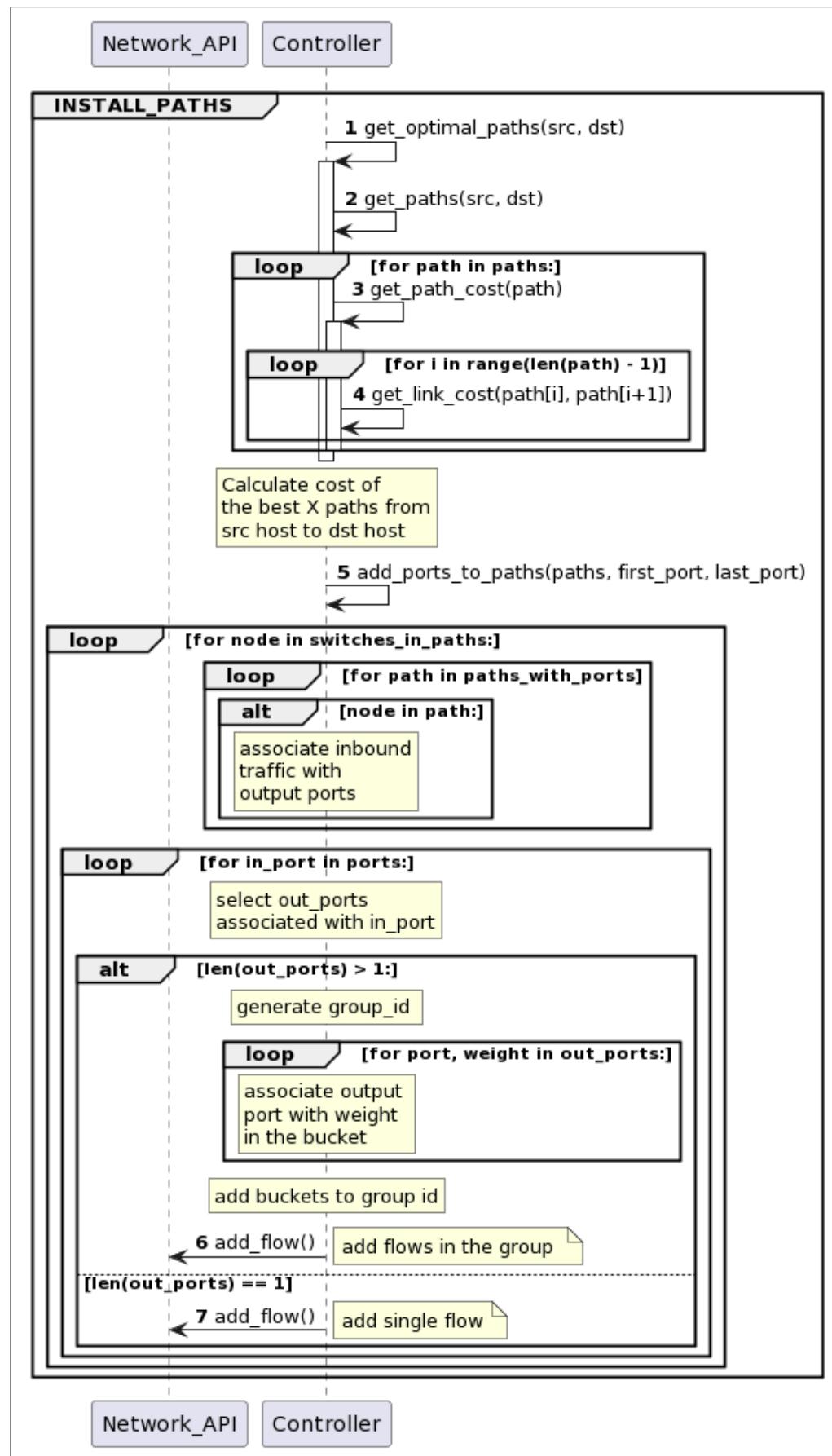
Do zarządzania siecią zestawioną w emulatorze Mininet, wykorzystano sterownik sieci w technologii Ryu. Został on wybrany ze względu na możliwość napisania, bądź edycji w prosty sposób aplikacji zarządzającą siecią. Sterownik buduje się poprzez napisanie skryptu w języku Python. Ważne jest, aby wersja Python'a na jakiej uruchamiany jest sterownik miała numer 3.8.0. Wspiera on protokół OpenFlow, dzięki któremu sterownik komunikuje się z urządzeniami dostępnymi w sieci. Podstawową zaletą sterownika Ryu jest mnogość bibliotek umożliwiających szeroki wachlarz zastosowań [15]. Działanie sterownika Ryu polega na dodawaniu do bufora tzw. event'ów przychodzących do sterownika i obsługiwanych zgodnie z kolejką FIFO (Rys. 1.2). W tej pracy wykorzystano sterownik [16], który realizuje wyznaczanie wielu tras ruchu (pierwotnie tylko jednorazowo). Został zmodyfikowany, aby wyznaczanie tras odbywało się cyklicznie co 20 s (Rys. 1.3), dzięki czemu było możliwe to dynamiczne dostosowywanie się do zmieniającej się sieci.



Rysunek 1.2: Architektura sterownika Ryu [9].



Rysunek 1.3: Diagram sekwencji cyklicznej inicjalizacji przeliczania trasy.



Rysunek 1.4: Diagram sekwencji instalowania tras.

Rozdział 2

Badania

2.1 Środowisko badawcze

2.1.1 Sieć rozproszona

Badania przeprowadzane w sali 1.11 budynku Teleinformatyki. Do przeprowadzenia badań na rozproszonej sieci komputerowej wykorzystano następujące urządzenia:

- 4x komputer z systemem Windows 10
- 8x ruter Cisco ISR4321-V/K9
- 4x przełącznik Cisco

Oprogramowanie jakim badano rozproszoną sieć to:

- iperf3
- Wireshark
- ping
- tracert

2.1.2 Sieć SDN

Badania sieci SDN dokonano na maszynie wirtualnej. Środowisko wirtualne, na którym było przeprowadzane badanie sieci SDN (Mininet):

- Hypervisor: VMware Workstation 17 Player
- System operacyjny: Ubuntu 22.04.3 LTS
- Pamięć RAM: 16 GB
- Procesor: 13th Gen Intel® Core™ i7-13700K, 8 rdzeni
- Miejsce na dysku: 50 GB
- Mininet 2.3.1b4
- Python 3.10.2

Środowisko wirtualne, na którym działał sterownik sieci:

- Hypervisor: VMware Workstation 17 Player
- System operacyjny: Ubuntu 20.04.6 LTS
- Pamięć RAM: 4 GB
- Procesor: 13th Gen Intel® Core™ i7-13700K, 2 rdzenie
- Miejsce na dysku: 30 GB
- Ryu 4.32
- Python 3.8.0

Oprogramowanie jakiego użyto do przeprowadzenia badania:

- Wireshark
- Ping
- iperf3

2.2 Przeprowadzenie badań

Badania przeprowadzono w 3 scenariuszach. Scenariusz 1 testuje zachowanie sieci bez interwencji w jej działaniu. Scenariusz 2 ma na celu zasymulować zaplanowaną wymianę potencjalnie wadliwych urządzeń sieciowych (R/S2, R/S4) na nowe o takiej samej konfiguracji. W scenariuszu 3 zbadano pracę sieci w przypadkach, gdy niektóre połączenia między urządzeniami (R/S2-R/S8, R/S4-R/S5, R/S2-R/S3) dezaktywują się z powodów losowych. Do wygenerowania ruchu wykorzystano program *iperf3*. Na komputerze WWW/h4 utworzono 9 serwerów *iperf3*, po 3 dla każdego komputera.

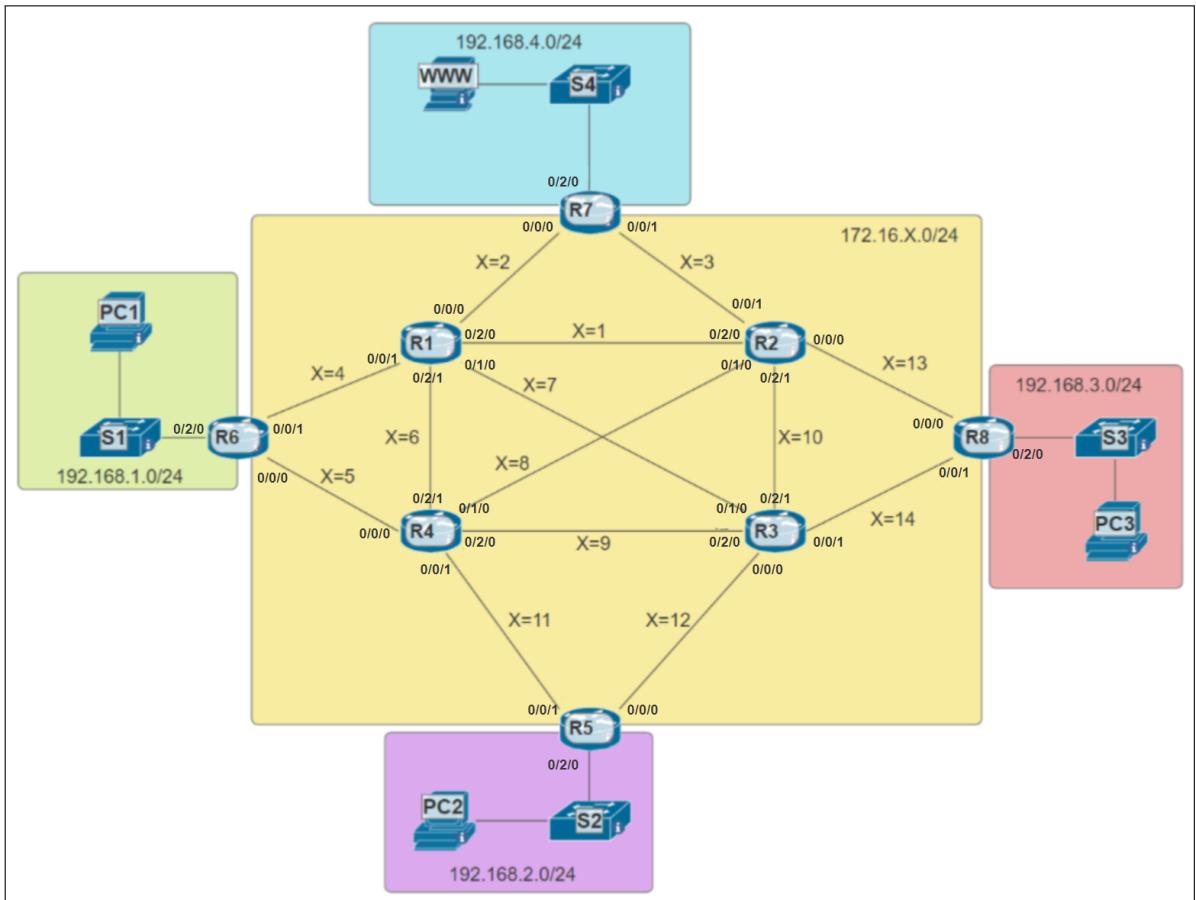
2.2.1 Sieć rozproszona

Badaną sieć zestawiono ręcznie w sposób przedstawiony na rys. 2.1. Następnie interfejsy ruterów oraz komputerów zostały zaadresowane zgodnie z rys. 2.2. Aby urządzenia mogły się komunikować włączono na routeraх protokół OSPF [17] oraz rozgłoszono sieci. Pliki konfiguracyjne można zobaczyć w repozytorium pracy [18]. Sprawdzono czy między komputerami jest komunikacja za pomocą programu *ping* i jakimi drogami ona zachodzi za pomocą programu *tracert*. Do zaobserwowania zdarzeń w sieci rozpoczęto przechwytywanie pakietów programem *Wireshark* na komputerze oznaczonym 'WWW'. Do każdego scenariusza włączono na każdym PC trzech klientów *iperf3* następującymi komendami:

```
$ iperf3.exe -c 192.168.9.4 -p 30[Nr PC][1..3] -u -b 5M -l 1468 -t [T] -i 1  
T - czas trwania badania
```

oraz 9 serwerów na komputerze 'WWW':

```
$ iperf3.exe -s -p 30[1..3][1..3]
```



Rysunek 2.1: Topologia sieci rozproszonej zestawionej w laboratorium - interfejsy.

Wyniki przebiegu działania sieci zapisywano w plikach **Scenariusz[Nr].LAB.pcapng**. Każdy scenariusz rozpoczynano od powyższej konfiguracji.

Scenariusz 1

T = 600 [s]

W tym scenariuszu nie ingerowano w sieć.

Scenariusz 2

T = 900 [s]

W tym scenariuszu zasymulowano wymianę ruterów R4 i R2. W tym celu wykonano następujące kroki:

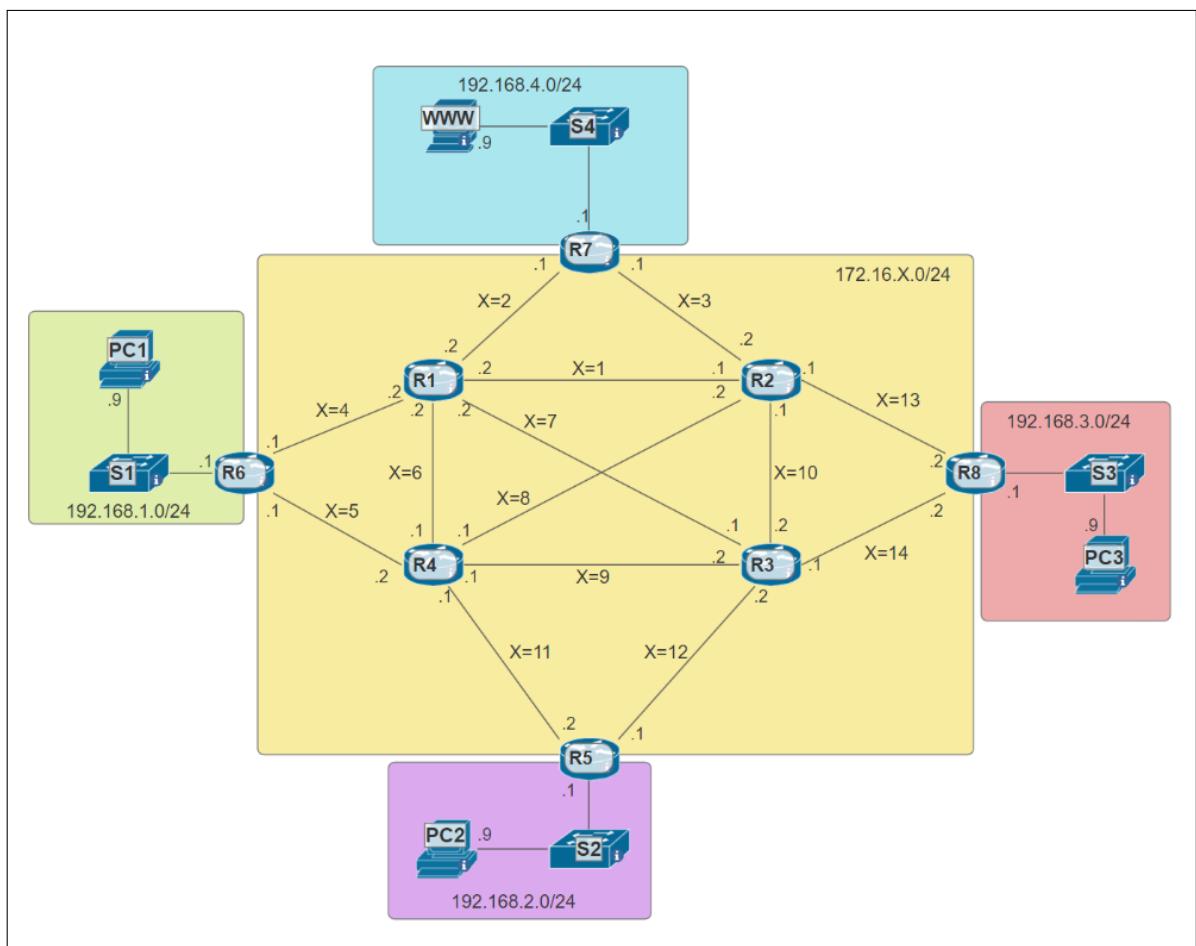
1. Po 1 minucie od rozpoczęcia obserwacji wyłączono R4
2. Po 2 minutach od rozpoczęcia obserwacji wyłączono R2
3. Po 4 minutach od rozpoczęcia obserwacji włączono R4
4. Po 5 minutach od rozpoczęcia obserwacji włączono R2

Scenariusz 3

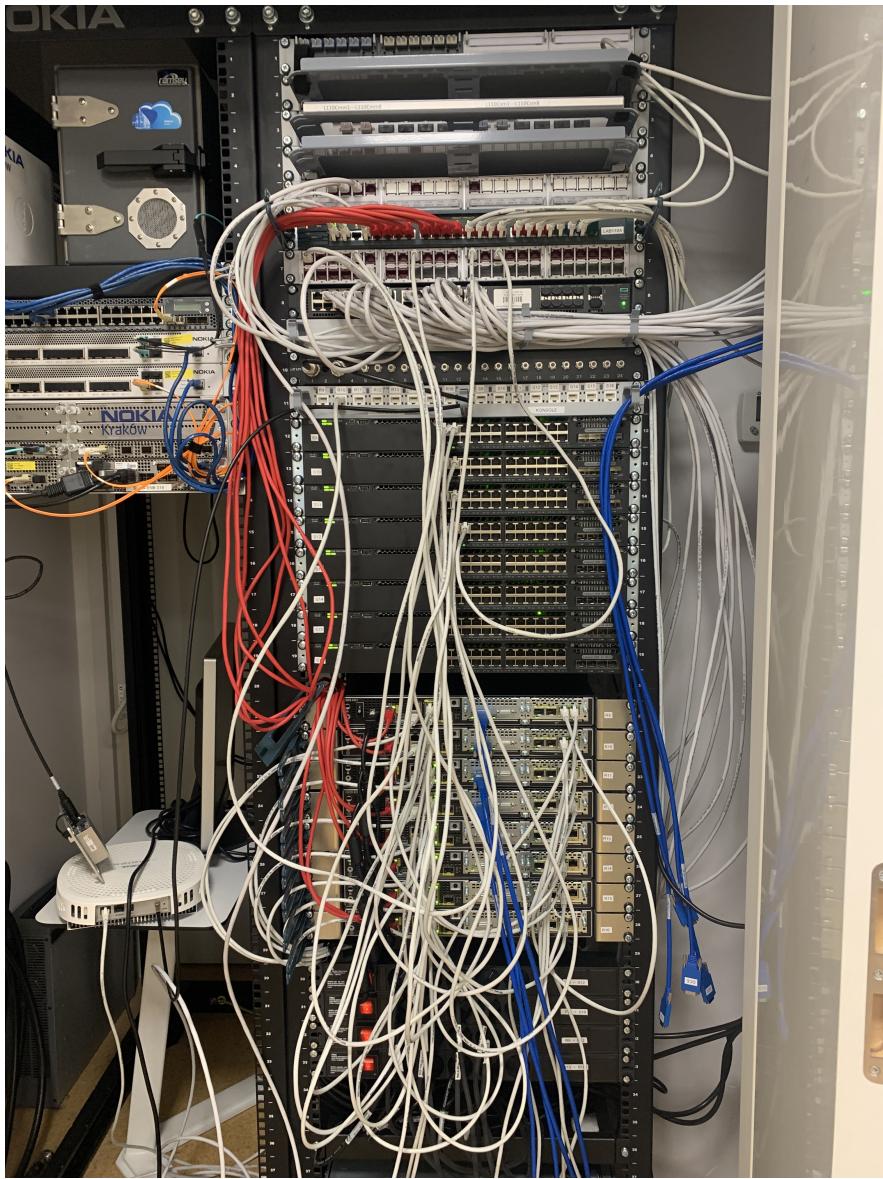
T = 300 [s]

W tym scenariuszu zasymulowano awarię łączy: R2-R8, R4-R5, R2-R3. W tym celu wykono następujące kroki:

1. Po 1 minucie od rozpoczęcia obserwacji odłączono kabel z interfejsu g0/0/0 na R8
2. Po 2 minutach od rozpoczęcia obserwacji odłączono kabel z interfejsu g0/0/1 na R5
3. Po 3 minutach od rozpoczęcia obserwacji odłączono kabel z interfejsu g0/2/1 na R3



Rysunek 2.2: Topologia sieci rozproszonej zestawionej w laboratorium - adresy.



Rysunek 2.3: Sieć zestawiona w pracowni laboratoryjnej.

2.2.2 Sieć SDN

Topologię przedstawioną na Rys 2.4 zestawiono w emulatorze sieci *Mininet*. Skrypt odpowiadający za zbudowanie sieci znajduje się na listingu 2.1. Do pełnienia funkcji zarządzającej siecią zastosowano sterownik w technologii Ryu. Kod do sterownika i sieci Mininet można znaleźć w repozytorium pracy [18]. Skrypty uruchomiono na dwóch różnych maszynach wirtualnych, najpierw skrypt sterownika, później zestawiający topologię. Ważne aby maszyny posiadały aktywne połączenie sieciowe ze sobą nawzajem. Na listingu 2.2 pokazano w jaki sposób skonfigurować hosty oraz przełączniki. Wartość pasma jaką zastosowano odpowiada wartościom na interfejsach ruterów sieci rozproszonej.

```
1 class CustomMininetTopo(Topo):
2
3     def build( self ):
4
5         NUMBER_OF_SWITCHES = 8
6
```

```

7      # Make nodes' names
8      switchesNames = [f"s{num+1}" for num in range(NUMBER_OF_SWITCHES)]
9      hostNames = ["h1", "h2", "h3", "h4"]
10
11      # Make nodes' objects
12      hosts = [self.addHost(hName, mac=f'00:00:00:0{hName[1]}:0{hName[1]}:0{hName[1]}') for hName in hostNames]
13      switches = [self.addSwitch(name) for name in switchesNames]
14      switch = lambda name: switches[switchesNames.index(name)]
15      host = lambda name: hosts[hostNames.index(name)]
16
17      # Make links between nodes
18      self.addLink(switch("s1"), switch("s2"), port1=1, port2=1, bw=50)
19      self.addLink(switch("s2"), switch("s3"), port1=2, port2=2, bw=50)
20      self.addLink(switch("s3"), switch("s4"), port1=1, port2=1, bw=50)
21      self.addLink(switch("s4"), switch("s1"), port1=2, port2=2, bw=50)
22
23      self.addLink(switch("s4"), switch("s6"), port1=3, port2=1, bw=50)
24      self.addLink(switch("s1"), switch("s6"), port1=4, port2=2, bw=50)
25      self.addLink(switch("s1"), switch("s7"), port1=3, port2=1, bw=50)
26      self.addLink(switch("s2"), switch("s7"), port1=4, port2=2, bw=50)
27      self.addLink(switch("s2"), switch("s8"), port1=3, port2=1, bw=50)
28      self.addLink(switch("s3"), switch("s8"), port1=4, port2=2, bw=50)
29      self.addLink(switch("s3"), switch("s5"), port1=3, port2=1, bw=50)
30      self.addLink(switch("s4"), switch("s5"), port1=4, port2=2, bw=50)
31
32      self.addLink(switch("s1"), switch("s3"), port1=5, port2=5, bw=2)
33      self.addLink(switch("s4"), switch("s2"), port1=5, port2=5, bw=2)
34
35      self.addLink(switch("s6"), host("h1"), port1=3, port2=1, bw=50)
36      self.addLink(switch("s7"), host("h4"), port1=3, port2=1, bw=50)
37      self.addLink(switch("s8"), host("h3"), port1=3, port2=1, bw=50)
38      self.addLink(switch("s5"), host("h2"), port1=3, port2=1, bw=50)

```

Listing 2.1: Skrypt zestawiający topologię sieci mininet.

```

1 def networkSetup():
2     # Create network
3     topo = CustomMininetTopo()
4     net = Mininet(topo=topo,
5                   switch=OVSKernelSwitch,
6                   controller=RemoteController(name='c0', ip='ADRES', port=6633),
7                   autoSetMacs=False,
8                   autoStaticArp=False,
9                   xterms=False,
10                  host=CPULimitedHost, link=TCLink)
11
12     # Configure hosts
13     for h in net.hosts:
14         h.cmd("sysctl -w net.ipv6.conf.all.disable_ipv6=1")
15         h.cmd("sysctl -w net.ipv6.conf.default.disable_ipv6=1")
16         h.cmd("sysctl -w net.ipv6.conf.lo.disable_ipv6=1")
17         h.cmd('sysctl -w net.ipv4.neigh.default.base_reachable_time_ms=10000')
18         h.cmd(f'xterm -T "{h.name}" &')
19         h.cmd(f'ifconfig')
20         h.cmd(f'xterm -T "{h.name}" &')
21         h.cmd(f'ifconfig')
22         h.cmd(f'xterm -T "{h.name}" &')
23         h.cmd(f'ifconfig')
24
25     # Configure switches to disable ipv6
26     for sw in net.switches:
27         sw.cmd("sysctl -w net.ipv6.conf.all.disable_ipv6=1")
28         sw.cmd("sysctl -w net.ipv6.conf.default.disable_ipv6=1")
29         sw.cmd("sysctl -w net.ipv6.conf.lo.disable_ipv6=1")
30
31     # Configure interfaces' MAC addresses
32     for switch in net.switches:
33         switch_nr = int(switch.name.replace('s', ''))
34         for port in switch.intfList():
35             if 'eth' in port.name:
36                 port_nr = int(port.name.replace(switch.name + '-eth', ''))


```

```

37     mac_address = '88:0{switch_nr:1d}:00:00:00:{port_nr:02d}'.format(
38         switch_nr=switch_nr, port_nr=port_nr)
39         port.setMAC(mac_address)
40
41     net.start()
42     dumpNodeConnections(net.hosts)
43     net.pingAll
44
45     # input()
46     # networkTest[test number](net)
47
48     CLI(net)
49     net.stop()

```

Listing 2.2: Skrypt konfigurujący urządzenia w sieci Mininet.

W konstruktorze *RemoteController* w argumencie *ip* (linia 6, List.2.2) należy podmienić *ADRES* na adres maszyny wirtualnej, na której uruchomiony został sterownik sieci. Po uruchomieniu obu skryptów należy poczekać, aż hosty przestaną się pingować, aby sterownik uzupełnił tablice przepływu w przełącznikach. Do zaobserwowania zdarzeń w sieci rozpoczęto przechwytywanie pakietów programem *Wireshark* na różnych interfejsach przełączników. Abyłączyć terminale hostów, należy w terminalu *mininet* wpisać:

\$ xtrem h1 h2 h3 h4

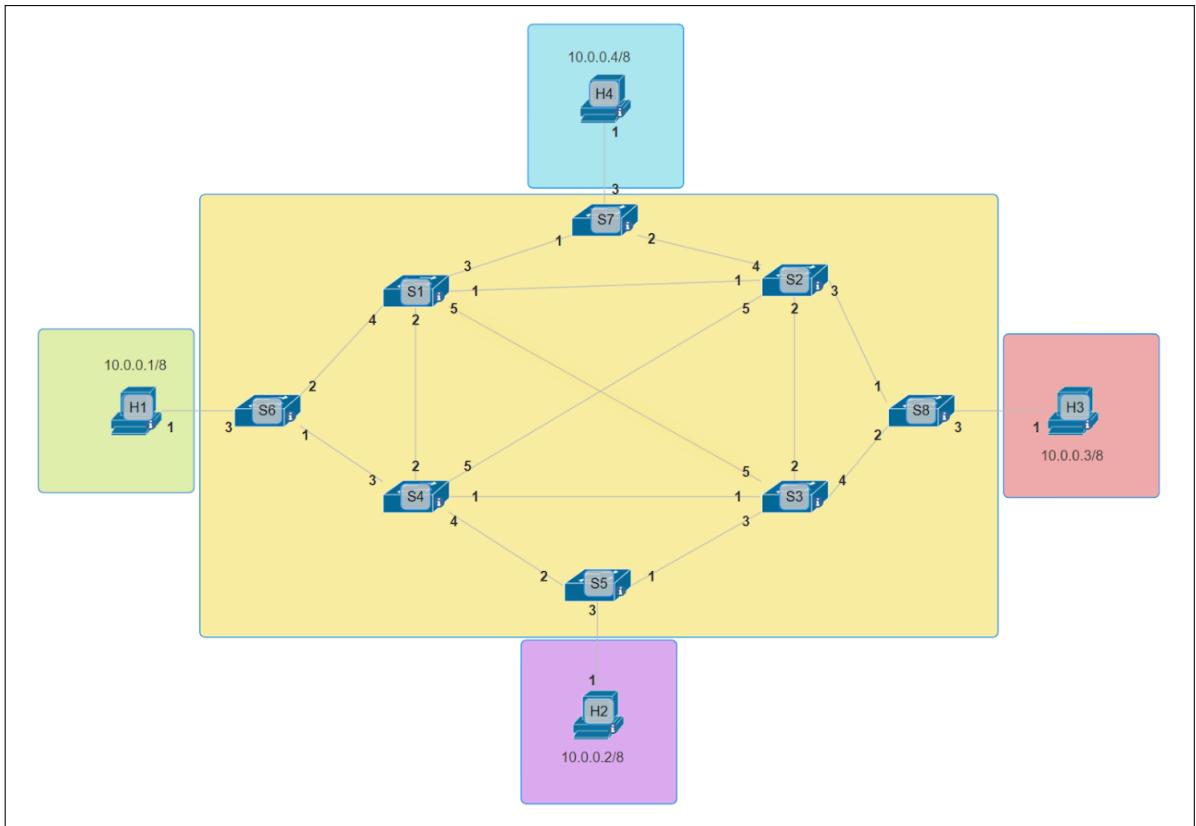
Do każdego scenariusza włączono na każdym hoście *h[1-3]* trzech klientów *iperf3* następującymi komendami:

\$ iperf3.exe -c 10.0.0.4 -p 30[Nr hosta][1..3] -u -b 5M -t [T] -i 1 &
T - czas trwania badania

oraz 9 serwerów na hoście *h4*:

\$ iperf3.exe -s -p 30[1..3][1..3] &

Przy uruchamianiu klientów nie podano długości pakietu UDP ponieważ domyślna długość nie wymagała fragmentacji.



Rysunek 2.4: Topologia sieci SDN zestawionej w emulatorze Mininet

Wyniki przebiegu działania sieci zapisywano w plikach **Scenariusz[Nr]_Mininet.pcapng**. Każdy scenariusz rozpoczynano od powyższej konfiguracji. Po każdej zmianie w sieci zebrano ze wszystkich przełączników tablice przepływu w celu analizy trasy jaką wybrał sterownik. Podobnie jak dla sieci rozproszonej przeprowadzono 3 scenariusze:

Scenariusz 1

$T = 600 \text{ [s]}$

W tym scenariuszu nie ingerowano w sieć. Programem *Wireshark* monitorowano następujące interfejsy: s5-eth1, s5-eth2, s6-eth1, s6-eth2, s7-eth1, s7-eth2, s8-eth1, s8-eth2.

Scenariusz 2

$T = 900 \text{ [s]}$

W tym scenariuszu zasymulowano wymianę przełączników S4 i S2. Aby przeprowadzić simulację należy zmienić skrypt następująco:

1. W linii 9 listingu 2.2 wartość argumentu *xterms* zmienić na *True*.
2. Odkomentować linie 45 i 46.
3. W linii 46 zmienić nazwę funkcji na tą z listingu 2.3.

Funkcja ma na celu przeprowadzenie następujących kroków:

1. Po 1 minucie od rozpoczęcia obserwacji zmieniono wszystkie połączenia do S4 w stan *down*

2. Po 2 minutach od rozpoczęcia obserwacji zmieniono wszystkie połączenia do S2 w stan *down*
3. Po 4 minutach od rozpoczęcia obserwacji zmieniono wszystkie połączenia do S4 w stan *up*
4. Po 5 minutach od rozpoczęcia obserwacji zmieniono wszystkie połączenia do S2 w stan *up*

W tym scenariuszu programem *Wireshark* monitorowano interfejsy: s3-eth3, s4-eth3, s4-eth2, s3-eth1, s7-eth1, s7-eth2. Po włączeniu monitoringu i uruchomieniu programów *iperf3* wciśnięto przycisk *Enter* w terminalu *Mininet*, aby włączyć skrypt symulacyjny.

```

1 def networkTest1(net: Mininet):
2     sleep(1*60)
3     net.configLinkStatus('s4', 's5', 'down')
4     net.configLinkStatus('s4', 's3', 'down')
5     net.configLinkStatus('s4', 's2', 'down')
6     net.configLinkStatus('s4', 's1', 'down')
7     net.configLinkStatus('s4', 's6', 'down')
8     net.configLinkStatus('s6', 's4', 'down')
9     net.configLinkStatus('s1', 's4', 'down')
10    net.configLinkStatus('s2', 's4', 'down')
11    net.configLinkStatus('s3', 's4', 'down')
12    net.configLinkStatus('s5', 's4', 'down')
13
14    sleep(1*60)
15    net.configLinkStatus('s2', 's8', 'down')
16    net.configLinkStatus('s2', 's3', 'down')
17    net.configLinkStatus('s2', 's4', 'down')
18    net.configLinkStatus('s2', 's1', 'down')
19    net.configLinkStatus('s2', 's7', 'down')
20    net.configLinkStatus('s8', 's2', 'down')
21    net.configLinkStatus('s3', 's2', 'down')
22    net.configLinkStatus('s4', 's2', 'down')
23    net.configLinkStatus('s1', 's2', 'down')
24    net.configLinkStatus('s7', 's2', 'down')
25
26    sleep(2*60)
27    net.configLinkStatus('s4', 's5', 'up')
28    net.configLinkStatus('s4', 's3', 'up')
29    net.configLinkStatus('s4', 's2', 'up')
30    net.configLinkStatus('s4', 's1', 'up')
31    net.configLinkStatus('s4', 's6', 'up')
32    net.configLinkStatus('s5', 's4', 'up')
33    net.configLinkStatus('s3', 's4', 'up')
34    net.configLinkStatus('s2', 's4', 'up')
35    net.configLinkStatus('s1', 's4', 'up')
36    net.configLinkStatus('s6', 's4', 'up')
37
38    sleep(1*60)
39    net.configLinkStatus('s2', 's8', 'up')
40    net.configLinkStatus('s2', 's3', 'up')
41    net.configLinkStatus('s2', 's4', 'up')
42    net.configLinkStatus('s2', 's1', 'up')
43    net.configLinkStatus('s2', 's7', 'up')
44    net.configLinkStatus('s8', 's2', 'up')
45    net.configLinkStatus('s3', 's2', 'up')
46    net.configLinkStatus('s4', 's2', 'up')
47    net.configLinkStatus('s1', 's2', 'up')
48    net.configLinkStatus('s7', 's2', 'up')
```

Listing 2.3: Skrypt symulujący wyłączanie się przełączników.

Scenariusz 3

$T = 300 \text{ [s]}$ W tym scenariuszu zasymulowano wyłączanie się połączeń kolejno: S2-S8, S4-S5, S2-S3. W celu przeprowadzenia tego testu dokonano następujący zmian względem scenariusza 1:

1. W linii 9 listingu 2.2 wartość argumentu `xterms` zmienić na `True`.
2. Odkomentować linie 45 i 46.
3. W linii 46 zmienić nazwę funkcji na tą z listingu 2.4.

Funkcja ma na celu przeprowadzenie następujących kroków:

1. Po 1 minucie od rozpoczęcia obserwacji zmieniono połączenie S2-S8 w stan `down`
2. Po 2 minucie od rozpoczęcia obserwacji zmieniono połączenie S4-S5 w stan `down`
3. Po 3 minucie od rozpoczęcia obserwacji zmieniono połączenie S2-S3 w stan `down`

Włączono monitoring programem *Wireshark* na interfejsach: s3-eth1, s3-eth2, s4-eth2, s4-eth3, s7-eth1, s7-eth2. Po włączeniu monitoringu i uruchomieniu programów *iperf3* wcisnięto przycisk *Enter* w terminalu *Mininet*, aby włączyć skrypt symulacyjny.

```

1 def networkTest2(net: Mininet):
2     sleep(1*60)
3     net.configLinkStatus('s8', 's2', 'down')
4     net.configLinkStatus('s2', 's8', 'down')
5
6     sleep(1*60)
7     net.configLinkStatus('s5', 's4', 'down')
8     net.configLinkStatus('s4', 's5', 'down')
9
10    sleep(1*60)
11    net.configLinkStatus('s3', 's2', 'down')
12    net.configLinkStatus('s2', 's3', 'down')
```

Listing 2.4: Skrypt symulujący wyłączanie się połączeń.

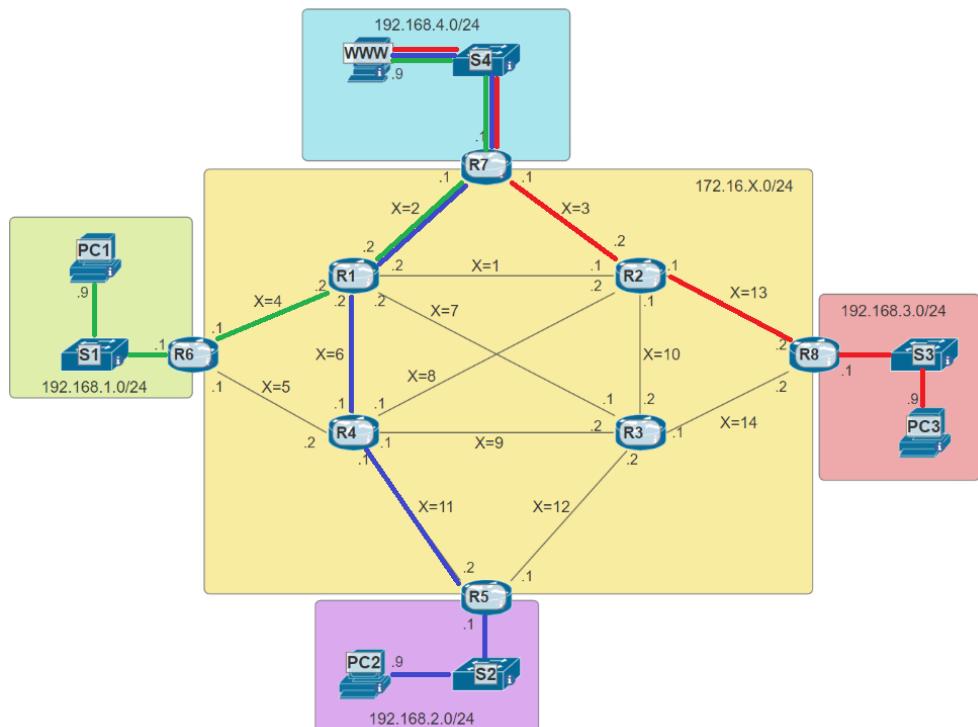
Rozdział 3

Wyniki

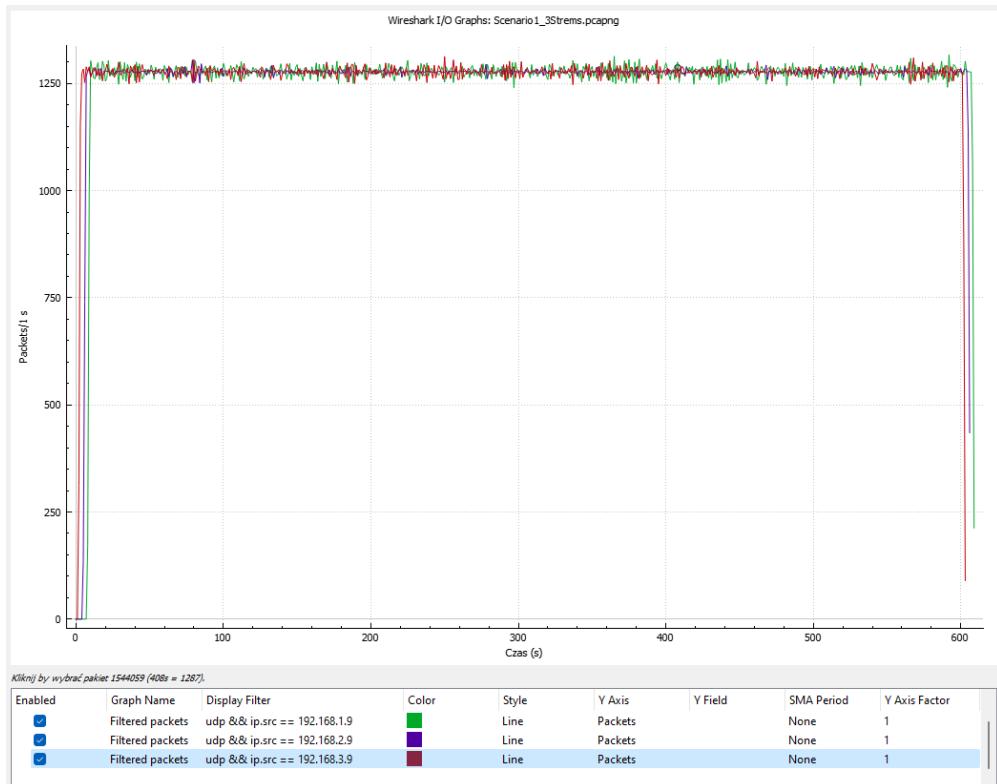
3.1 Wyniki - sieć rozproszona

Po zakończeniu badań, przeanalizowano wyniki programu *Tracert* i dla każdego scenariusza narysowano diagramy ze zmianami tras, jakie zachodziły podczas działania sieci. Diagramy pokazują zachowanie się protokołu OSPF [17], który został włączony na routeraх. Zebracono również wyniki z programu *Wireshark* i na ich podstawie narysowano wykresy liczby otrzymanych pakietów na sekundę w zależności od czasu trwania pomiaru.

3.1.1 Scenariusz 1



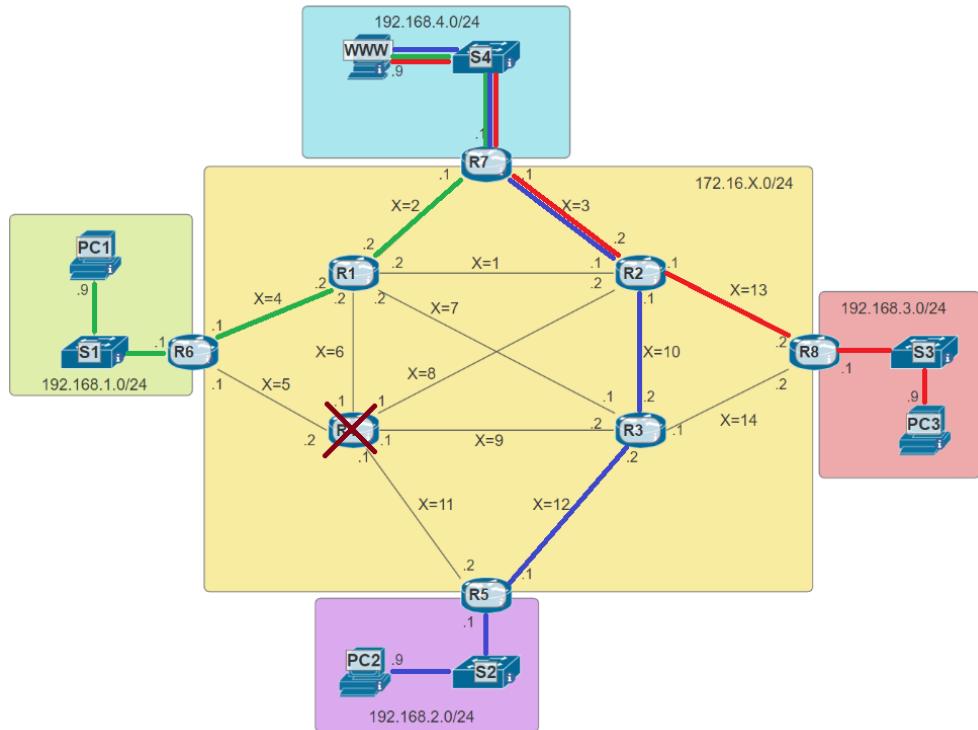
Rysunek 3.1: Trasy pakietów z komputerów źródłowych do komputera WWW - brak awarii.



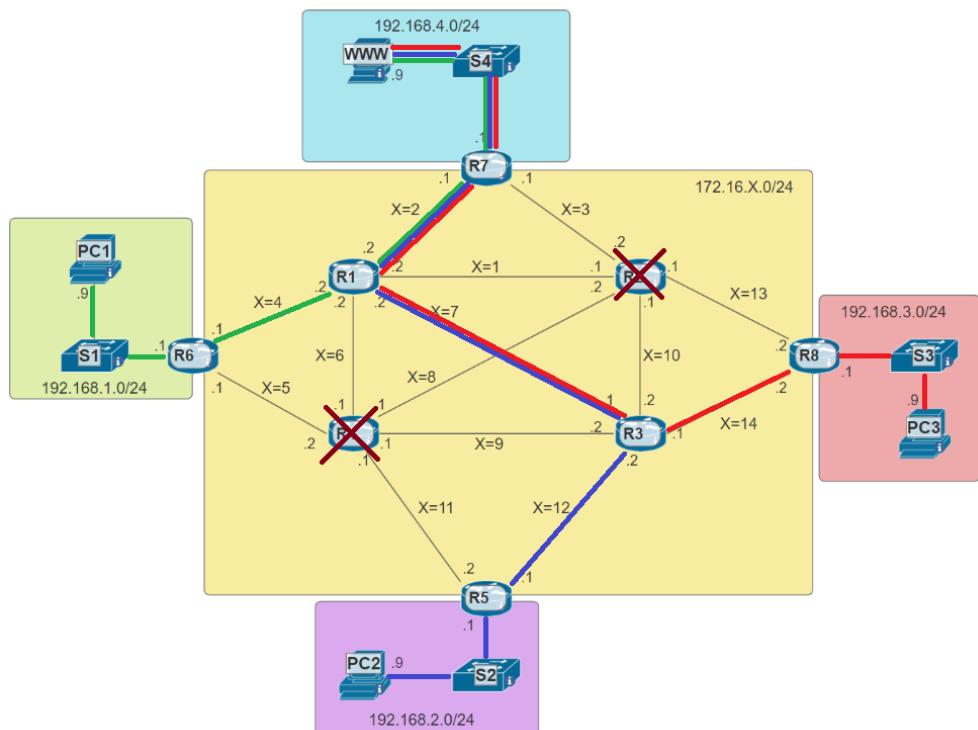
Rysunek 3.2: Wykres zależności pakietów na sekundę od czasu (Scen. 1).

W powyższym scenariuszu można zauważyc, że protokół OSPF wyznaczył jedną trasę z komputerów *PC1-3* do komputera *WWW*. Protokół włączyłby balansowanie ruchem na ruterze *R5*, jeżeli ruch nadawany byłby do dwóch komputerów za ruterem *R7*.

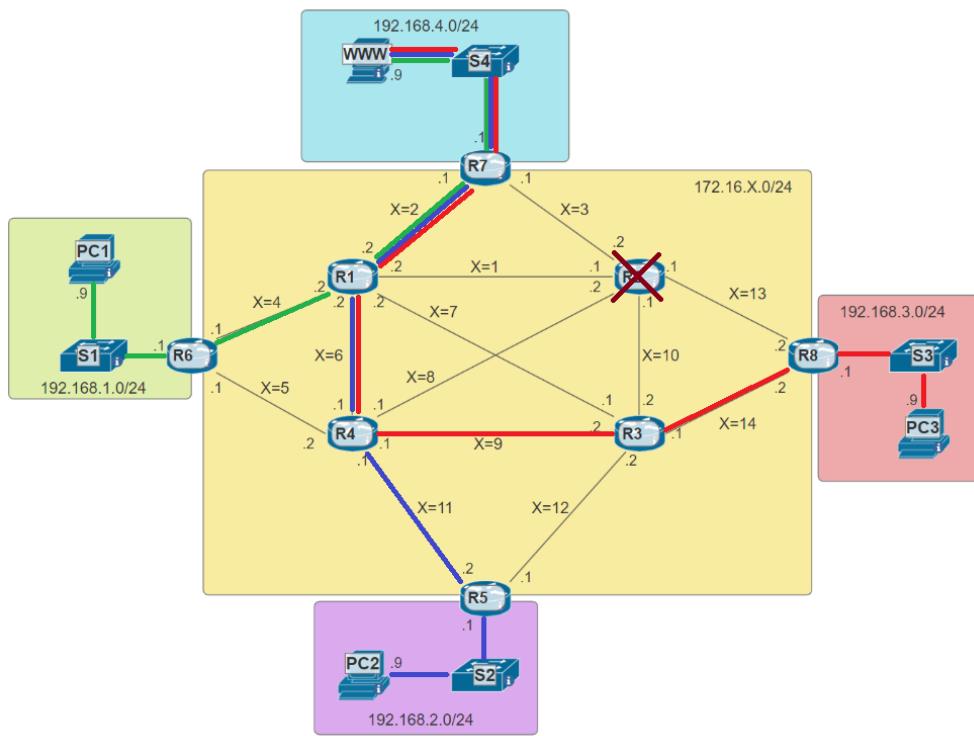
3.1.2 Scenariusz 2



Rysunek 3.3: Trasy pakietów z komputerów źródłowych do komputera WWW - R4 jest wyłączony.



Rysunek 3.4: Trasy pakietów z komputerów źródłowych do komputera WWW - R4 i R2 są wyłączone.



Rysunek 3.5: Trasy pakietów z komputerów źródłowych do komputera *WWW* - R2 jest wyłączony.

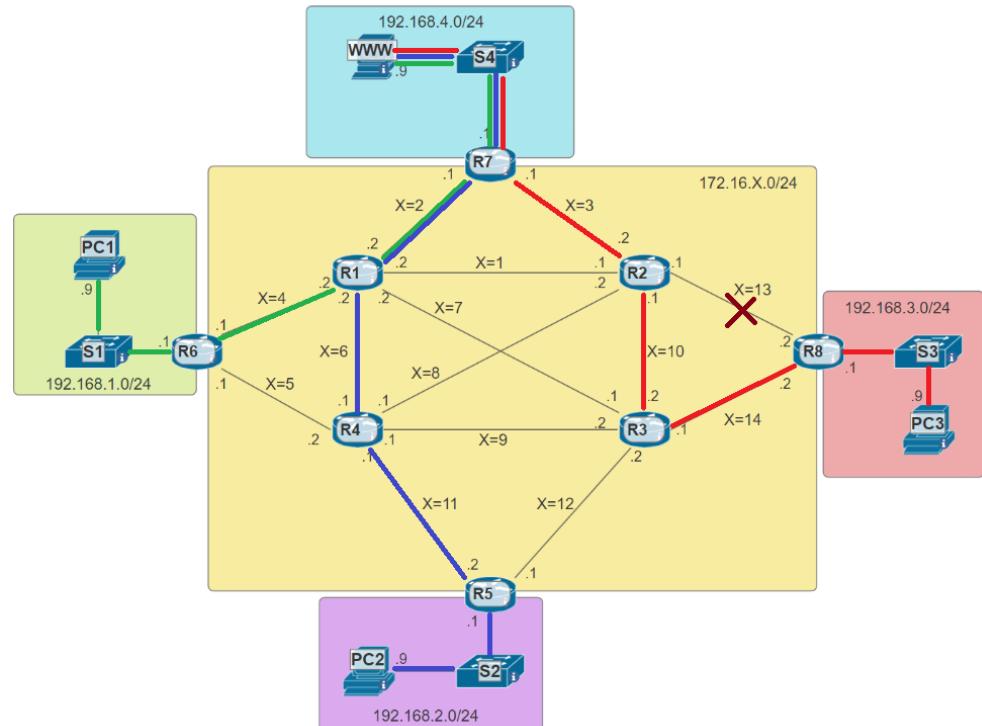


Rysunek 3.6: Wykres zależności liczby wysłanych pakietów na sekundę od czasu (Scen. 2).

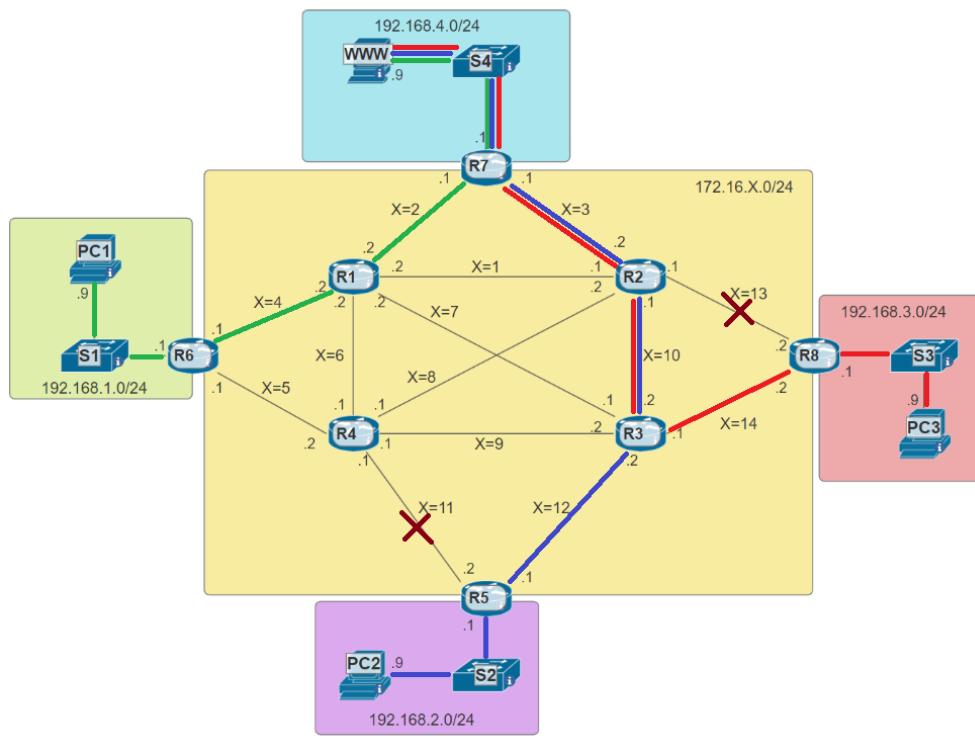
Na wykresie (Rys. 3.6) można zaobserwować chwilowy spadek przychodzących pakietów w momencie zmiany trasy dla ruchu z PC2, gdy wyłączony zostaje ruter R4. Podobny spadek

występuje dla ruchu z PC3 podczas wyłączenia rutera R2, kiedy cały ruch z obu komputerów PC1 i PC2 został przekierowany na łącze z interfejsami typu serial o przepustowości 2 Mb/s. Na wykresie nie widać też zmiany ruchu z PC3 po włączeniu się rutera R2, ponieważ protokół OSPF jest protokołem stanu łączna, a nie wektora odległości i liczba ruterów pomiędzy komputerami nie wpływa na decyzje dotyczące wyznaczania trasy.

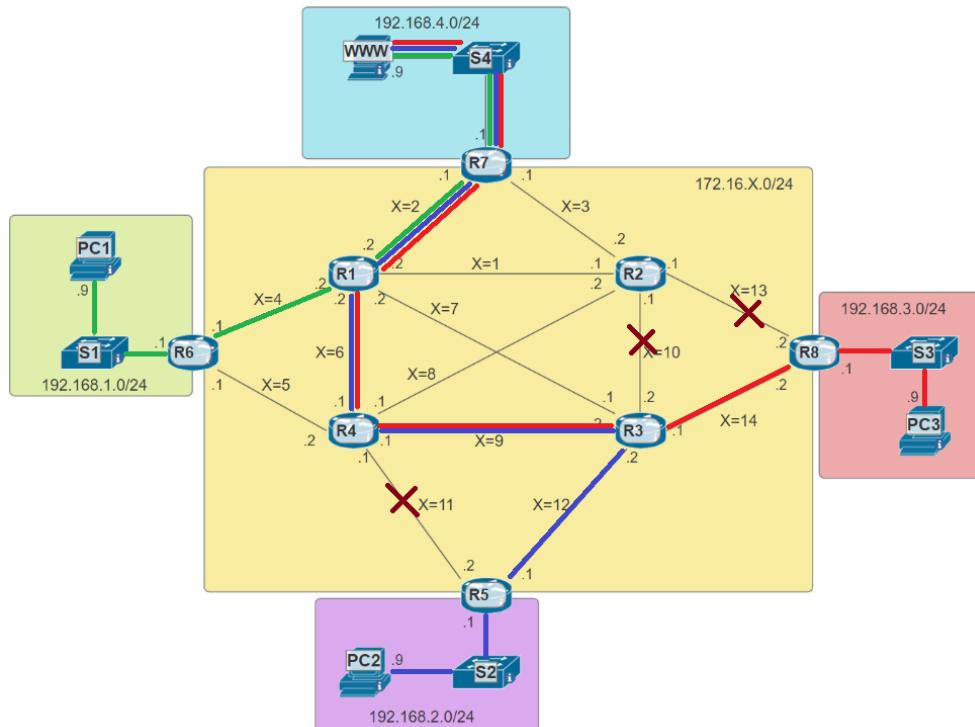
3.1.3 Scenariusz 3



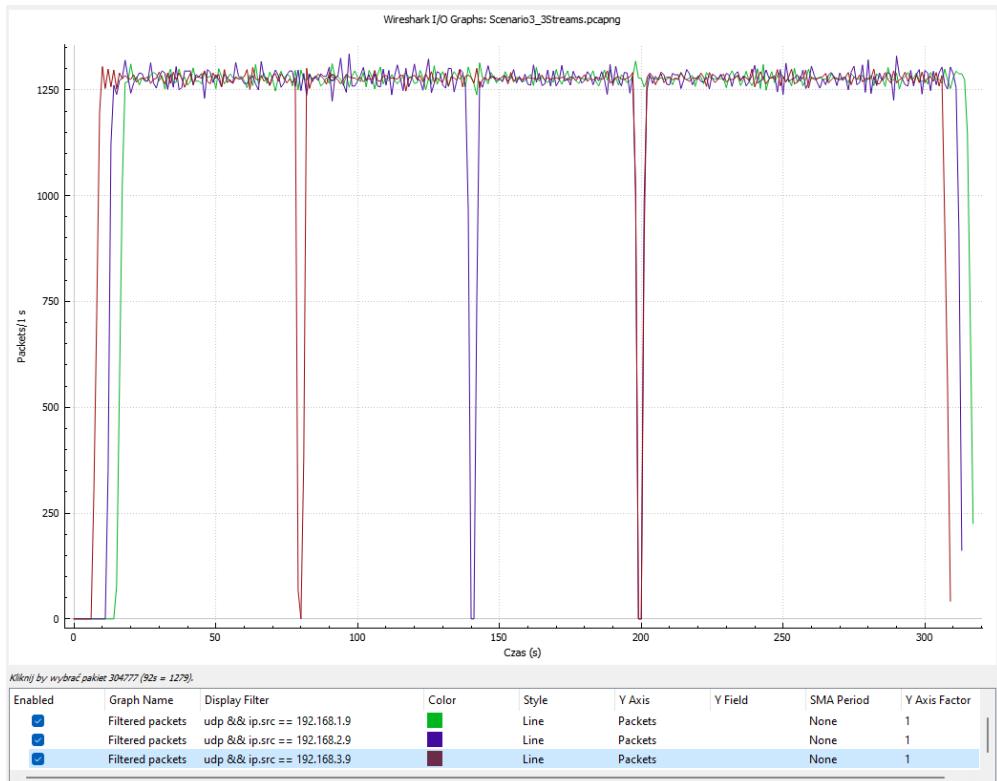
Rysunek 3.7: Trasy pakietów z komputerów źródłowych do komputera *WWW* - 1 awaria.



Rysunek 3.8: Trasy pakietów z komputerów źródłowych do komputera *WWW* - 2 awarie.



Rysunek 3.9: Trasy pakietów z komputerów źródłowych do komputera *WWW* - 3 awarie.



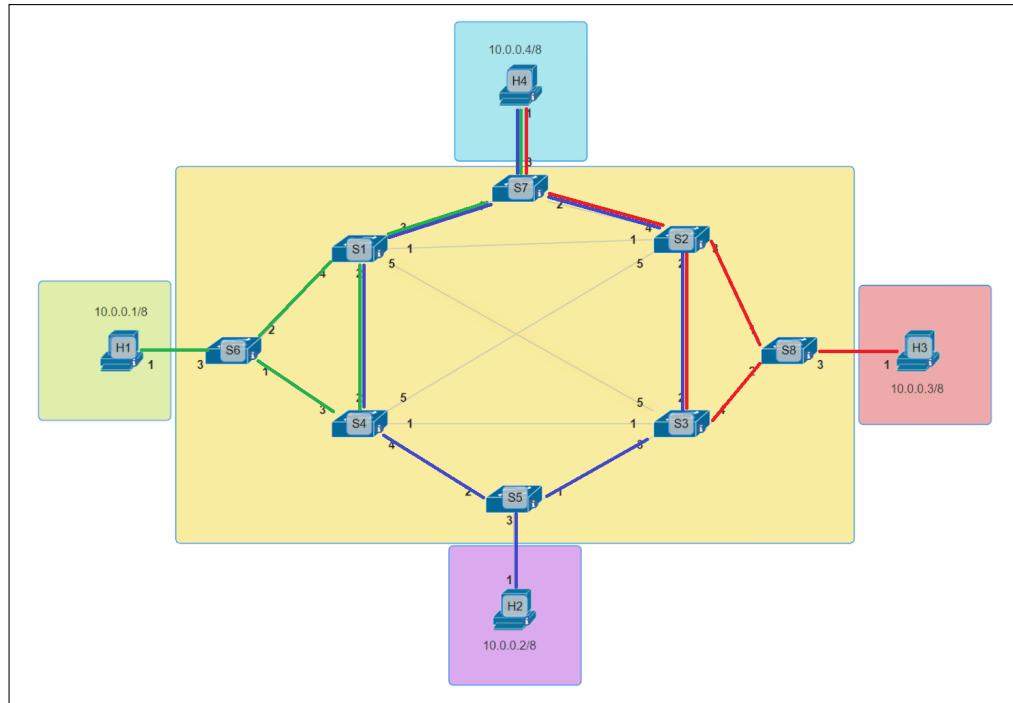
Rysunek 3.10: Wykres zależności liczby wysłanych pakietów na sekundę od czasu (Scen. 3).

Na wykresie (rys. 3.10) można zaobserwować spadki w liczbie pakietów wychodzących na interfejsie g0/2/0 routera R7 w czasie gdy połączenia uległy awarii i trasa ruchu z komputerów PC2 i PC3 była zmieniana. Przy pierwszym spadku zmieniona została tylko trasa od PC3, w drugim od PC2, a w trzecim ruch z obu tras został przekierowany.

3.2 Wyniki - SDN

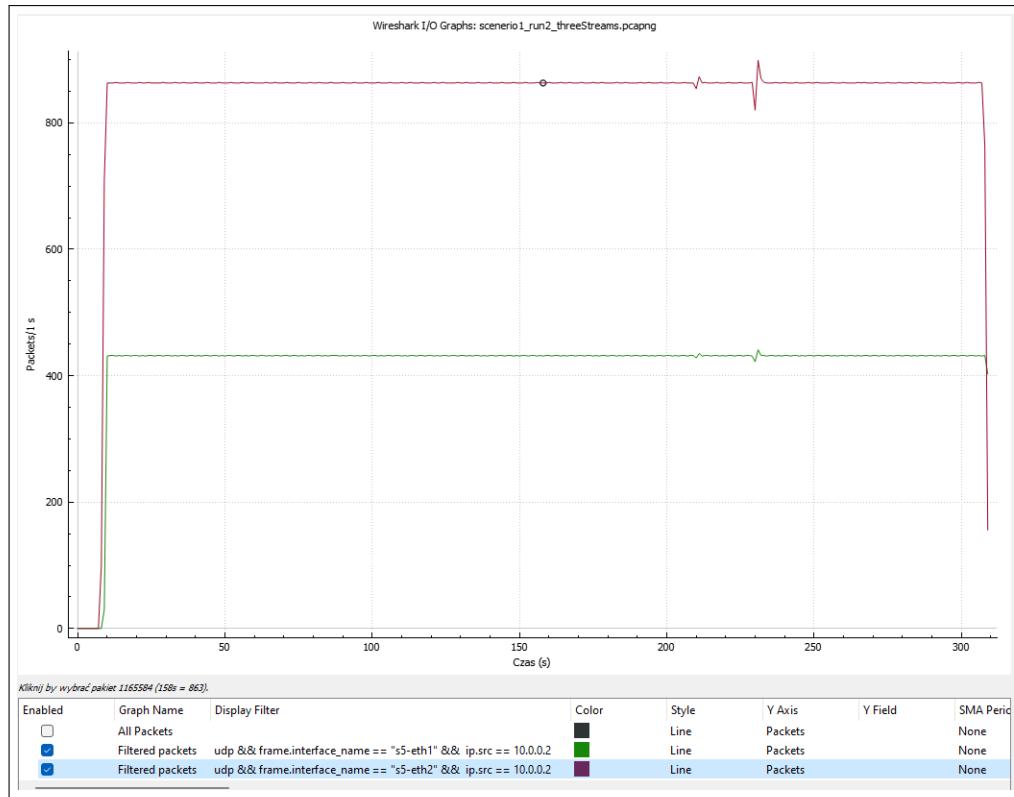
Dla każdej konfiguracji sieci w danym scenariuszu pobrano tablice przepływów i na ich podstawie narysowano diagramy tras pakietów wychodzących z poszczególnych hostów. W konfiguracji sterownika sieci ustwiono, że do jednego adresu docelowego, jeżeli jest to możliwe, dobierane są 2 trasy.

3.2.1 Scenariusz 1

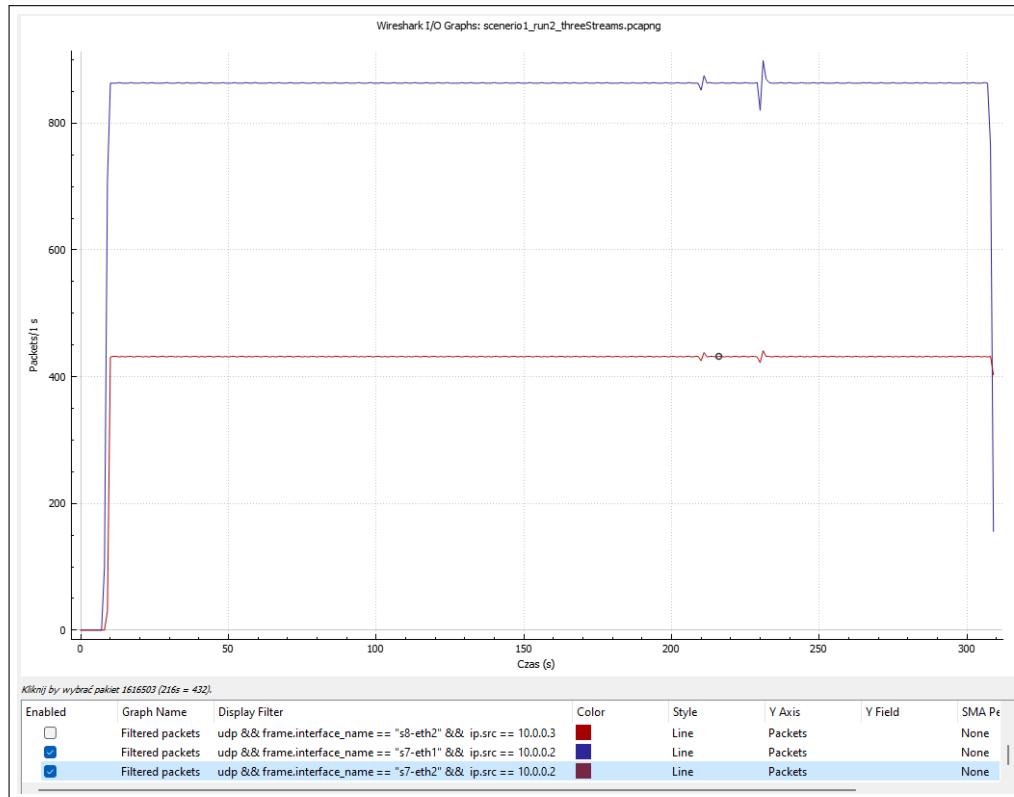


Rysunek 3.11: Trasy pakietów z hostów źródłowych 1-3 do hosta h_4 - brak awarii.

Na poniższych wykresach (Rys. 3.12, 3.13) można zaobserwować rozdzielenie się 3 strumieni ruchu trafiających do przełącznika $s5$ na interfejsy $s5\text{-}eth3$ pomiędzy interfejsy $s5\text{-}eth1$ i $s5\text{-}eth2$. Interfejsy, którymi ruch został transmitowany zostały wybrane na podstawie mechanizmu dobierania trasy opisanego w rozdziale *Teoria - Ryu*. Rozdzielenie ruchu pozwoliło na zrównoważone obciążenie łączy.

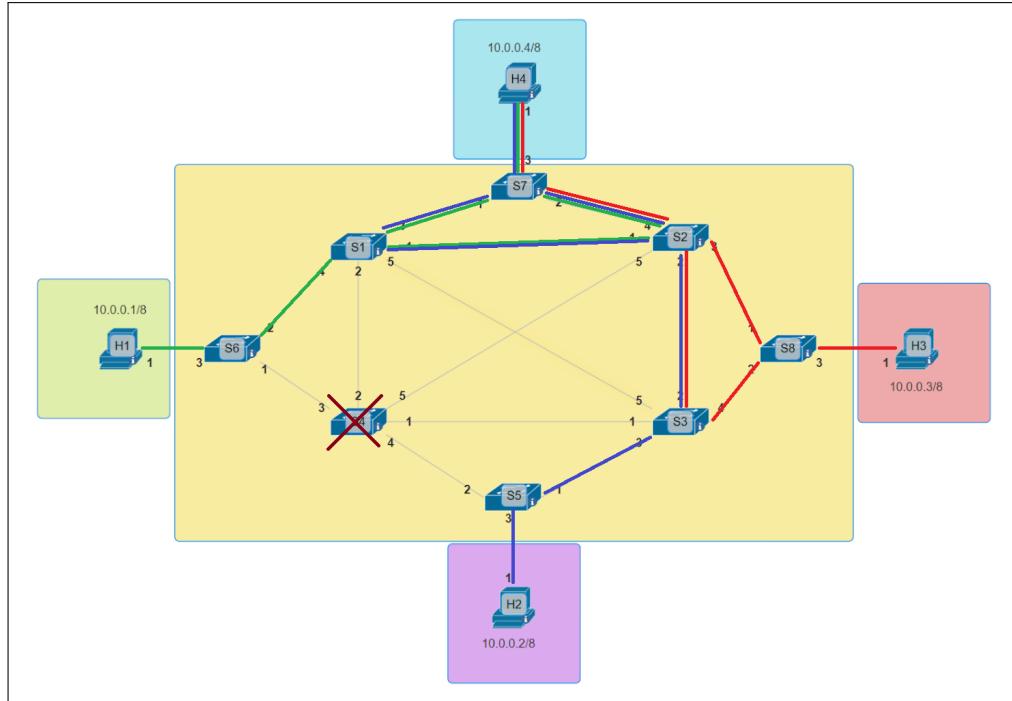


Rysunek 3.12: Wykres zależności liczby transmitowanych pakietów na sekundę od czasu dla s5 i h2 (Scen. 1).

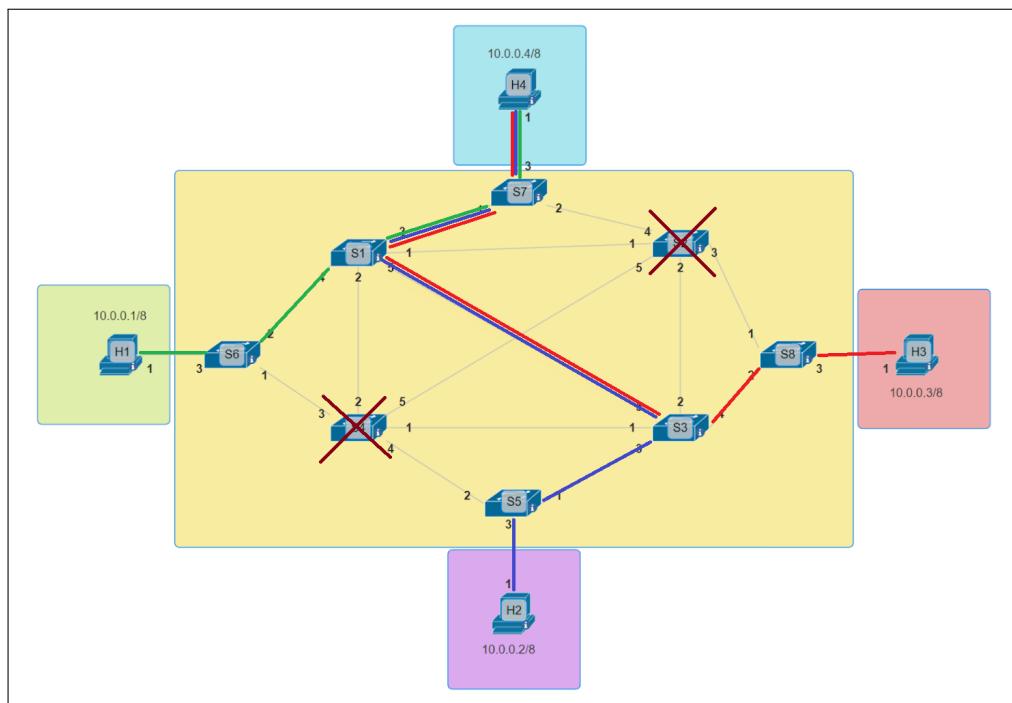


Rysunek 3.13: Wykres zależności liczby transmitowanych pakietów na sekundę od czasu dla s7 i h2 (Scen. 1).

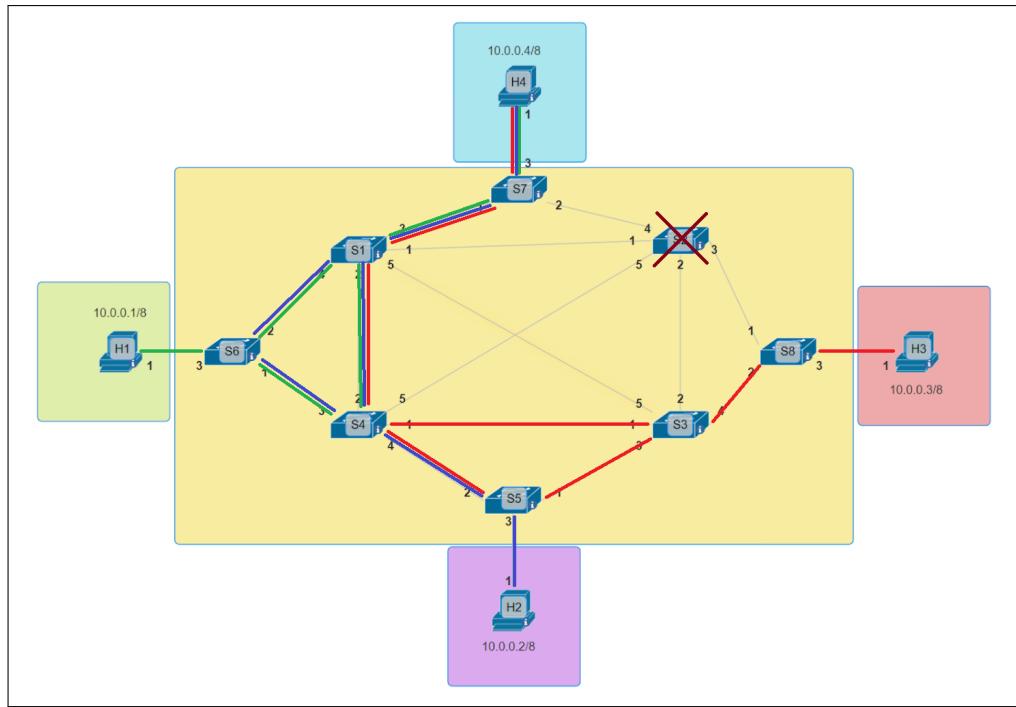
3.2.2 Scenariusz 2



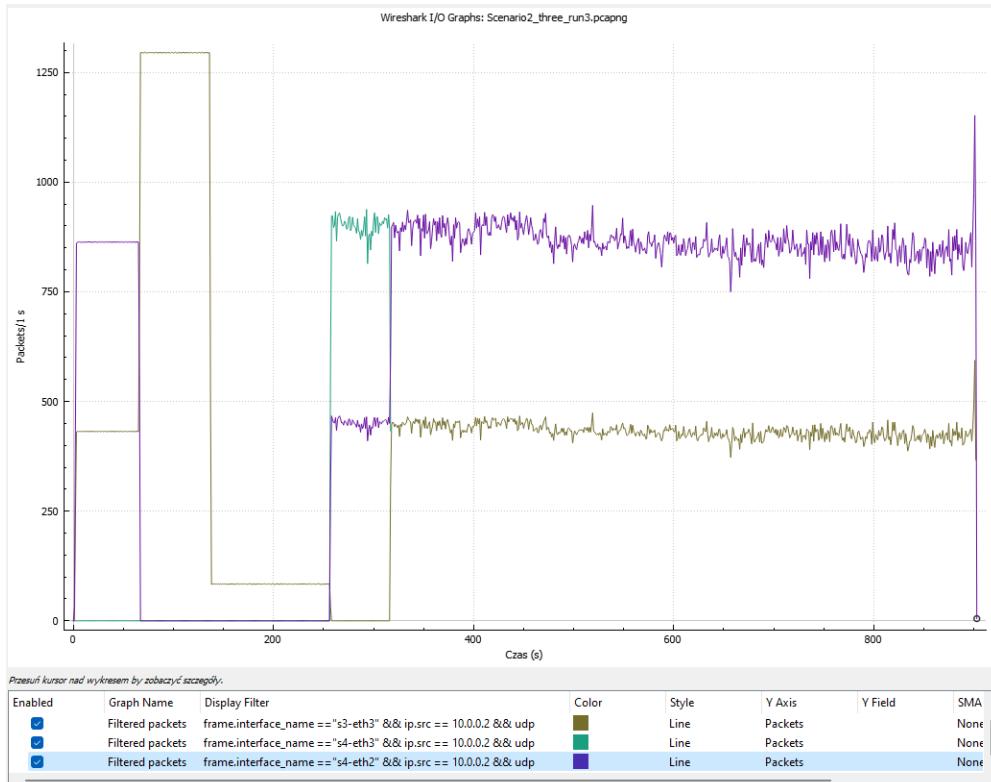
Rysunek 3.14: Trasy pakietów z hostów źródłowych 1-3 do hosta h_4 - wyłączony S4.



Rysunek 3.15: Trasy pakietów z hostów źródłowych 1-3 do hosta h_4 - wyłączone S4 S2.



Rysunek 3.16: Trasy pakietów z hostów źródłowych 1-3 do hosta h_4 - wyłączony S2.



Rysunek 3.17: Wykres zależności liczby transmitowanych pakietów na sekundę od czasu dla h2 (Scen. 2)

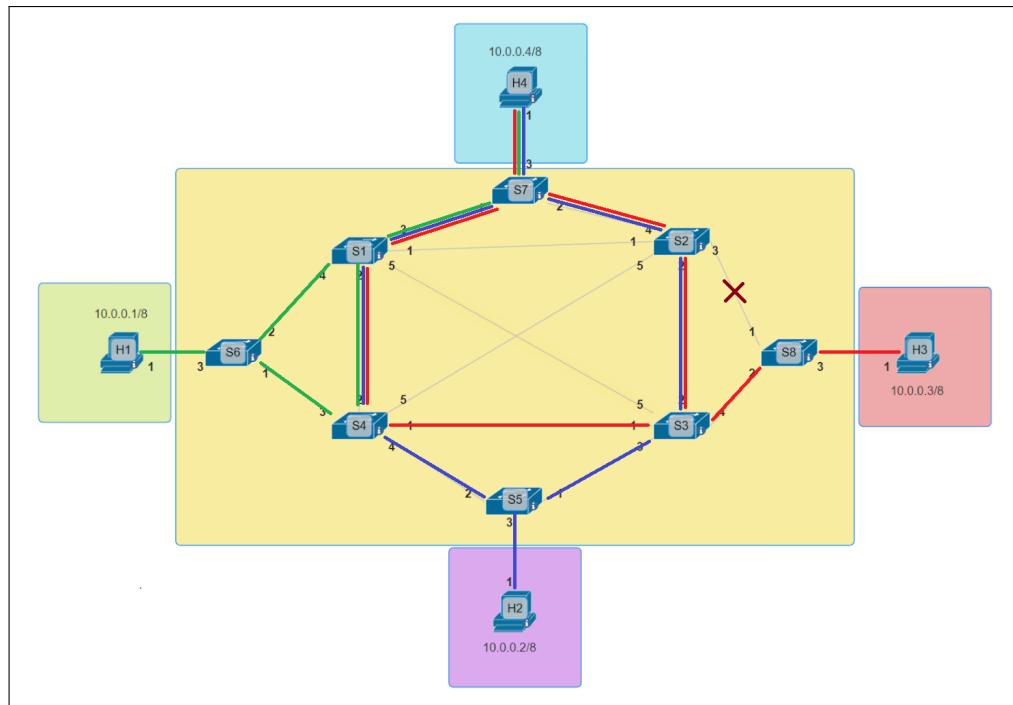
Jak można zaobserwować na Rys. 3.3 - 3.5 dzięki logice zastosowanej w sterowniku trasa pomiędzy przełącznikami S3 - S1 została wybrana tylko w momencie gdy nie była dostępna żadna inna trasa. Algorytm oblicza opłacalność tras nie tylko na podstawie liczby prze-

łączników pomiędzy źródłem a celem, ale także na podstawie przepustowości dostępnych łączy. Aby oddać realia laboratoryjne łącza S4 - S2 i S1 - S3 miały przepustowość 2 Mbps a pozostałe 50 Mbps, co poskutkowało otrzymanym wynikiem.

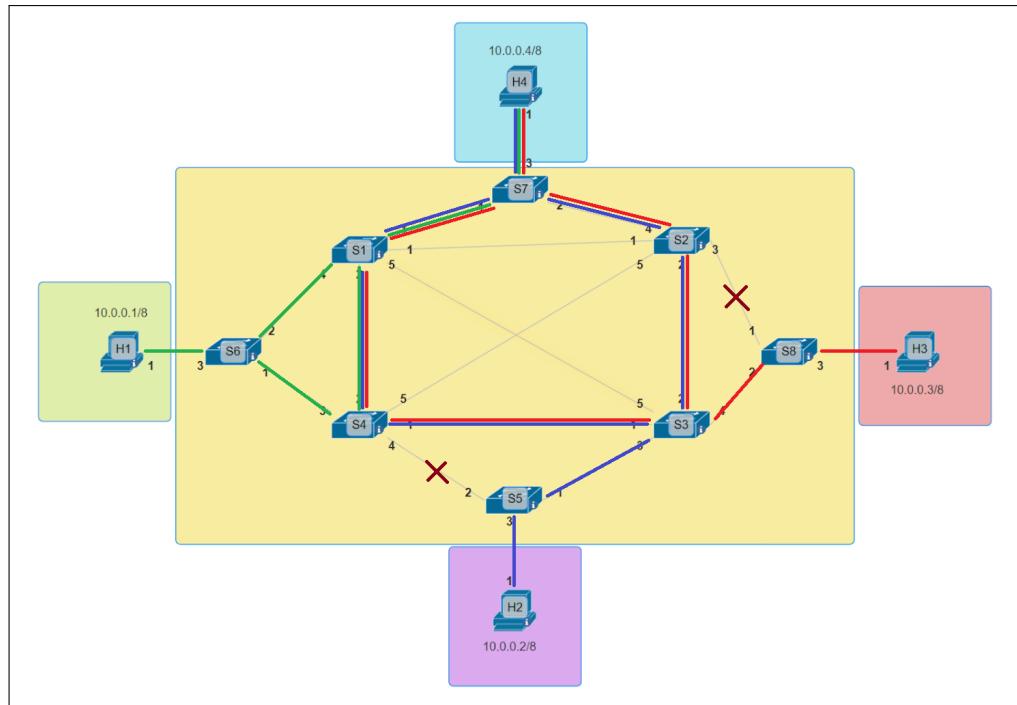
Na powyższym wykresie (Rys. 3.17) można zauważyć, jak ruch z $h2$ zmieniał swoją trasę podczas trwania symulacji. Ciekawe jest to, że trasa wybrana podczas braku dostępności przełącznika S2 dla 2 strumieni ma mniejszą wagę (weight:4) niż ta wybrana dla 1 strumienia (weight:6) co potwierdza, że dobór tras posiada element losowości. Co więcej, po przywróceniu do działania przełącznika S2 ruch był rozdzielany tak jak na początku symulacji.

3.2.3 Scenariusz 3

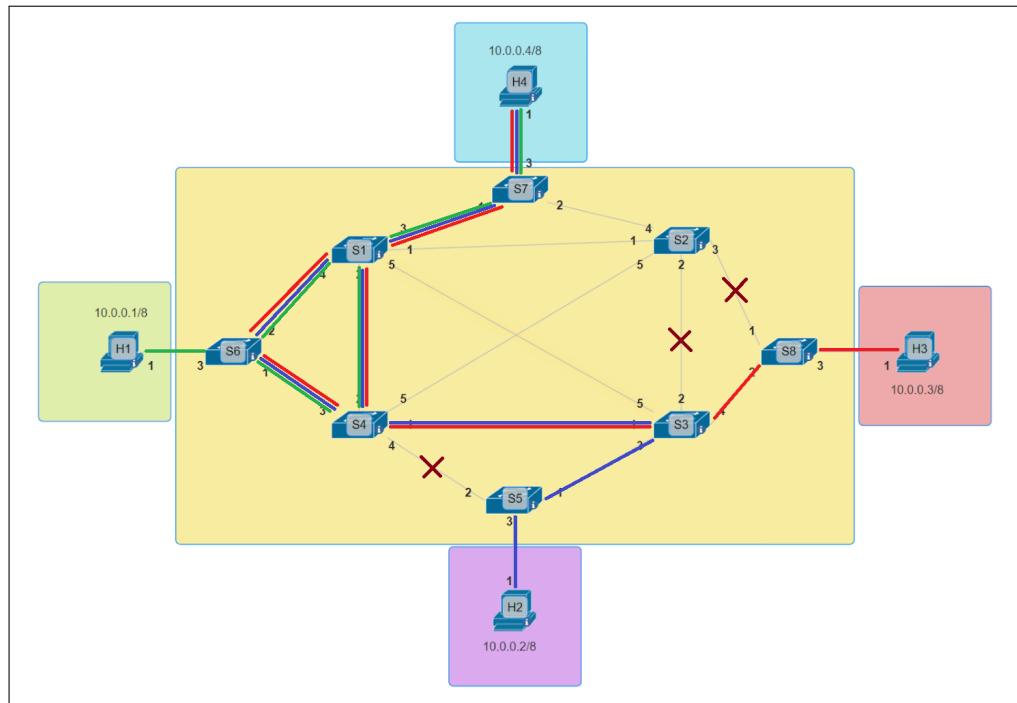
W tym scenariuszu zasymulowano awarię 3 połączeń. Na przykładzie ruchu z komputera 2 na przełączniku $s4$ (Rys. 3.21) można zaobserwować wpływ, jaki ma czas aktualizacji trasy na sterowniku oraz czas ważności przepływu na przełączniku, na wybór trasy dla ruchu w sieci. Oba te czasy ustalono na 20 s. Na wykresie można zauważyć spadek ruchu pakietowego na interfejsie $s4\text{-}eth2$. Oznacza to, że trasa przeliczana jest tuż po wystąpieniu awarii, ponieważ przełącznik $s4$ nie otrzymuje pakietów od przełącznika $s3$. Ruch na $s3$ obsłużono przez przepływ, który ma tylko jeden interfejs wyjściowy ($s3\text{-}eth2$) dla zadanego ruchu, dlatego na interfejsie $s4\text{-}eth1$ nie pojawia się żaden ruch. Po upływie czasu ważności przepływu przełącznik $s3$ usuwa przepływ odpowiadający za obsługę pakietów przychodzących z $h2$ do $h4$ pojawiających się na interfejsie $s3\text{-}eth3$. Następnie zwraca się do sterownika o otrzymanie nowego przepływu obsługującego wcześniejszy wspomniany ruch. Sterownik wysyła przepływ obliczony tuż po awarii, który wskazuje już 2 interfejsy ($s4\text{-}eth1$ i $s4\text{-}eth2$) jako wyjściowe, na które przełącznik $s3$ może kierować zadany ruch. Po ok. 20 s znowu pojawia się ruch na interfejsie $s4\text{-}eth2$.



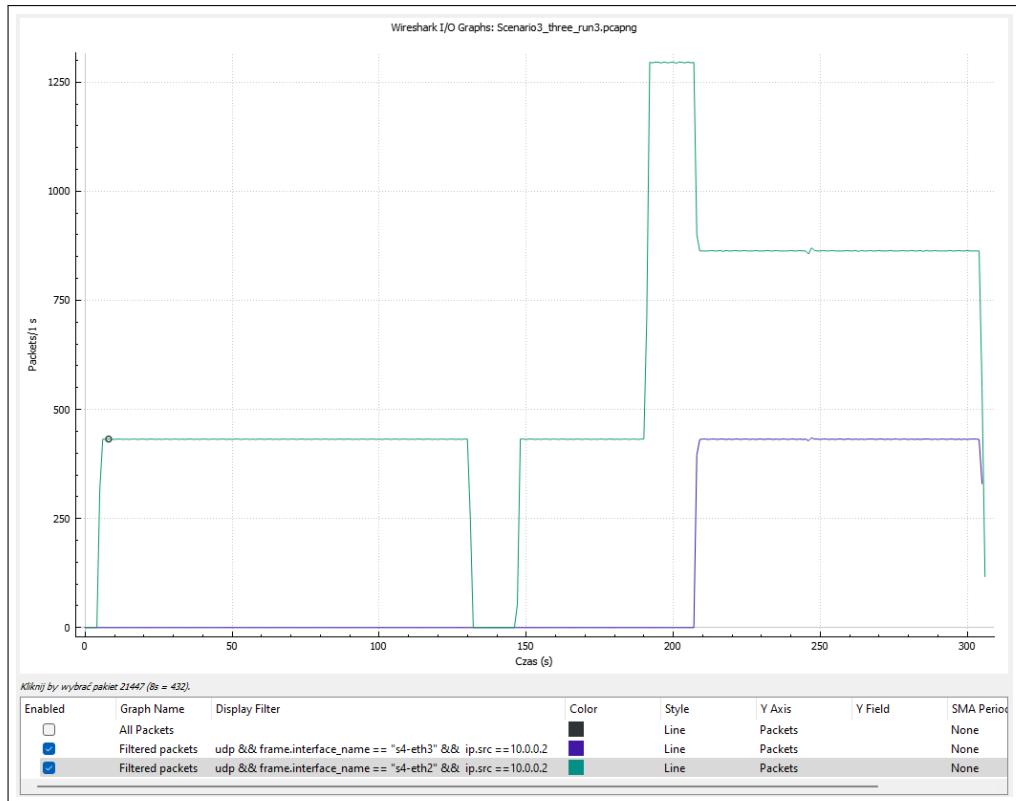
Rysunek 3.18: Trasy pakietów z hostów źródłowych 1-3 do hosta $h4$ - awaria S2-S8.



Rysunek 3.19: Trasy pakietów z hostów źródłowych 1-3 do hosta h_4 - awaria S4-S5.



Rysunek 3.20: Trasy pakietów z hostów źródłowych 1-3 do hosta h_4 - awaria S2-S3.



Rysunek 3.21: Wykres zależności liczby transmitowanych pakietów na sekundę od czasu dla komputera h2 na s4 (Scen. 3).

Rozdział 4

Wnioski

Ocena i porównanie dwóch różnych podejść do zarządzania sieciami komputerowymi, tj. tradycyjnej sieci rozproszonej opartej na protokole OSPF oraz nowoczesnej programowej sieci komputerowej (SDN), stanowi istotny element analizy infrastruktury sieciowej. Oba te podejścia mają swoje zalety w przypadku konkretnych zastosowań. Sieć rozproszona oparta na protokole OSPF charakteryzuje się decentralizacją zarządzania trasami, gdzie poszczególne routery komunikują się między sobą, a protokół OSPF podejmuje decyzje dotyczące wyznaczania tras na podstawie kosztu trasy do konkretnej sieci. Z drugiej strony, w podejściu SDN, sterownik sieci pełni kluczową rolę w zarządzaniu przepływami ruchu, umożliwiając dynamiczną rekonfigurację tras na podstawie bieżących warunków sieciowych.

Przeprowadzone powyżej badanie skupiło się głównie na porównaniu sposobów wyznaczania tras oraz kryteriów, na podstawie których ich dokonywano. Można zauważyć, że działanie przedstawionego sterownika jest rozszerzeniem działania protokołu OSPF. Protokół ten jest ograniczony do wyznaczenia 1 trasy do sieci docelowej w przypadku gdy alternatywna trasa ma gorszy koszt. Druga trasa jest zapisywana do tablicy rutingu tylko wtedy, gdy koszty obu tras będą równe. Protokół OSPF wyśle ruch drugą trasą dopiero, gdy adres docelowy drugiego ruchu będzie inny. Sterownik w przeciwieństwie do protokołu OSPF pozwala na wyznaczenie dowolnej liczby tras zdefiniowanej przez administratora sieci i wysyłanie nimi ruchu, który ma jeden komputer docelowy. W trakcie obliczania kosztu trasy brana jest pod uwagę nie tylko przepustowość łącza, ale także liczba węzłów między komunikującymi się komputerami. Dlatego w scenariuszu 2 po przywróceniu do działania przełącznika S2 ruch z komputera h2 był rozdzielany pomiędzy 2 trasy tak samo jak na początku w przeciwieństwie do ruchu w sieci rozproszonej.

Dzięki scentralizowanej architekturze sieci SDN sterownik posiada globaną świadomość sieci i może wspomóc tym proces podejmowania decyzji doboru tras na odległych od siebie i względnie nie powiązanych ze sobą przełącznikach. Sterownik wie co wydarzyło się w jednym miejscu w sieci i dzięki zastosowanym obliczeniom matematycznym bądź modelom uczenia maszynowego może wpływać na ruch w innym miejscu. Dla tradycyjnych sieci informacja o zmianach sieci w jednym miejscu musiała przejść przez całą sieć do innego routera i dopiero w nim podjęta byłaby decyzja czy ta zmiana wpłynie na działanie sieci w zasięgu tego routera.

Na podstawie powyższych wniosków można stwierdzić, że sieci SDN nie tylko mogą pracować jak tradycyjne sieci, ale także w znacznym stopniu rozszerzać ich podstawowe możliwości, a także stosować mechanizmy niedostępne na tradycyjnie działających ruterach.

Spis rysunków

1.1	Architektura SDN według Guy Pujolle [11].	5
1.2	Architektura sterownika Ryu [9].	7
1.3	Diagram sekwencji cyklicznej inicjalizacji przeliczania trasy.	7
1.4	Diagram sekwencji instalowania tras.	8
2.1	Topologia sieci rozproszonej zestawionej w laboratorium - interfejsy.	11
2.2	Topologia sieci rozproszonej zestawionej w laboratorium - adresy.	12
2.3	Sieć zestawiona w pracowni laboratoryjnej.	13
2.4	Topologia sieci SDN zestawionej w emulatorze Mininet	16
3.1	Trasy pakietów z komputerów źródłowych do komputera <i>WWW</i> - brak awarii.	19
3.2	Wykres zależności pakietów na sekundę od czasu (Scen. 1).	20
3.3	Trasy pakietów z komputerów źródłowych do komputera <i>WWW</i> - R4 jest wyłączony.	21
3.4	Trasy pakietów z komputerów źródłowych do komputera <i>WWW</i> - R4 i R2 są wyłączone.	21
3.5	Trasy pakietów z komputerów źródłowych do komputera <i>WWW</i> - R2 jest wyłączony.	22
3.6	Wykres zależności liczby wysłanych pakietów na sekundę od czasu (Scen. 2).	22
3.7	Trasy pakietów z komputerów źródłowych do komputera <i>WWW</i> - 1 awaria.	23
3.8	Trasy pakietów z komputerów źródłowych do komputera <i>WWW</i> - 2 awarie.	24
3.9	Trasy pakietów z komputerów źródłowych do komputera <i>WWW</i> - 3 awarie.	24
3.10	Wykres zależności liczby wysłanych pakietów na sekundę od czasu (Scen. 3).	25
3.11	Trasy pakietów z hostów źródłowych 1-3 do hosta <i>h4</i> - brak awarii.	26
3.12	Wykres zależności liczby transmitowanych pakietów na sekundę od czasu dla s5 i h2 (Scen. 1).	27
3.13	Wykres zależności liczby transmitowanych pakietów na sekundę od czasu dla s7 i h2 (Scen. 1).	27
3.14	Trasy pakietów z hostów źródłowych 1-3 do hosta <i>h4</i> - wyłączony S4.	28
3.15	Trasy pakietów z hostów źródłowych 1-3 do hosta <i>h4</i> - wyłączone S4 S2.	28
3.16	Trasy pakietów z hostów źródłowych 1-3 do hosta <i>h4</i> - wyłączony S2.	29
3.17	Wykres zależności liczby transmitowanych pakietów na sekundę od czasu dla h2 (Scen. 2)	29
3.18	Trasy pakietów z hostów źródłowych 1-3 do hosta <i>h4</i> - awaria S2-S8.	30
3.19	Trasy pakietów z hostów źródłowych 1-3 do hosta <i>h4</i> - awaria S4-S5.	31
3.20	Trasy pakietów z hostów źródłowych 1-3 do hosta <i>h4</i> - awaria S2-S3.	31
3.21	Wykres zależności liczby transmitowanych pakietów na sekundę od czasu dla komputera h2 na s4 (Scen. 3).	32

Bibliografia

- [1] Hailan Kuang, Yiwen Qiu, Ruifang Li, and Xinhua Liu. A hierarchical k-means algorithm for controller placement in sdn-based wan architecture. In *2018 10th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA)*, pages 263–267, 2018.
- [2] Aanchal Chaurasia, Soumya Nandan Mishra, and Suchismita Chinara. Performance evaluation of software-defined wireless networks in it-sdn and mininet-wifi. In *2019 1st International Conference on Advances in Information Technology (ICAIT)*, pages 315–319, 2019.
- [3] M. Gharbaoui, C. Contoli, G. Davoli, G. Cuffaro, B. Martini, F. Paganelli, W. Cerroni, P. Cappanera, and P. Castoldi. Demonstration of latency-aware and self-adaptive service chaining in 5g/sdn/nfv infrastructures. In *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 1–2, 2018.
- [4] Anichur Rahman, Md. Jahidul Islam, Farhana Akter Sunny, and Mostofa Kamal Nasir. Distblocksdn: A distributed secure blockchain based sdn-iot architecture with nfv implementation for smart cities. In *2019 2nd International Conference on Innovation in Engineering and Technology (ICIET)*, pages 1–6, 2019.
- [5] Farah Chahlaoui, Hamza Dahmouni, and Hassan El Alami. Multipath-routing based load-balancing in sdn networks. In *2022 5th Conference on Cloud and Internet of Things (CloudIoT)*, pages 180–185, 2022.
- [6] Shanu Bhardwaj and Ashish Girdhar. Software-defined networking: A traffic engineering approach. In *2021 IEEE 8th Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON)*, pages 1–5, 2021.
- [7] Beny Nugraha and Rathan Narasimha Murthy. Deep learning-based slow ddos attack detection in sdn-based networks. In *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 51–56, 2020.
- [8] Linux Foundation. *Open vSwitch Documentation*. <https://www.openvswitch.org/support/dist-docs/ovs-ofctl.8.html>.
- [9] RYU project team. *Ryu Documentation*. <https://book.ryu-sdn.org/en/html/arch.html>.
- [10] Nguyen Viet Ha, Du Dong Quan, and Tran Thi Thao Nguyen. Graphical user interface for ryu software defined network controller. In *2022 IEEE 8th Information Technology International Seminar (ITIS)*, pages 312–317, 2022.

- [11] Guy Pujolle. *Software Networks : Virtualization, SDN, 5G, and Security*, volume 1. John Wiley Sons, Incorporated, 2nd ed edition, 2020.
- [12] Farah Chahlaoui, Hamza Dahmouni, and Hassan El Alami. Multipath-routing based load-balancing in sdn networks. In *2022 5th Conference on Cloud and Internet of Things (CIoT)*, pages 180–185, 2022.
- [13] Truong Thu Huong, Ngo Do Dang Khoa, Nguyen Xuan Dung, and Nguyen Huu Thanh. A global multipath load-balanced routing algorithm based on reinforcement learning in sdn. In *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 1336–1341, 2019.
- [14] Mininet Project Contributors. *Mininet documentation*. <https://github.com/mininet/mininet/wiki/Documentation>.
- [15] Saleh Asadollahi, Bhargavi Goswami, and Mohammed Sameer. Ryu controller’s scalability experiment on software defined networks. In *2018 IEEE International Conference on Current Trends in Advanced Computing (ICCTAC)*, pages 1–5, 2018.
- [16] Dharshan Kumar. *Multipath load balancing*. <https://github.com/dharshankumar2002/Multipath-Load-Balancing>.
- [17] Wendell Odom. *CCNA 200-301 Official Cert Guide*, volume 1. Cisco Press, 2019.
- [18] Kacper Handzlik. *Repozytorium pracy*. https://github.com/KacperHandzlik/praca_inzynierska/tree/main.