

SCHT Laboratorium 2 - Płaszczyzna sterowania sieci

Tymoteusz Malec, Kacper Średnicki

Politechnika Warszawska, Cyberbezpieczeństwo 23Z

18 listopada, 2023

Spis treści

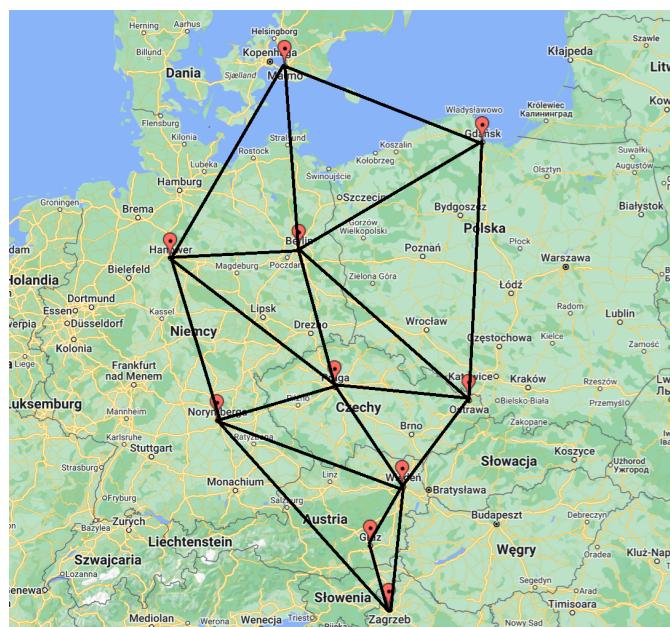
1. Wstęp	2
2. Część 1	2
2.1. Zadanie 1	2
2.1.1. TCP	3
2.1.2. UDP	4
2.1.3. Wnioski	4
2.2. Zadanie 2	4
2.2.1. TCP	5
2.2.2. UDP	5
2.2.3. TCP i UDP	6
2.2.4. Wnioski	6
3. Część 2	6
3.1. Zadanie 3	6
3.1.1. Plik konfiguracyjny	6
3.1.2. Zarządzanie siecią	6
3.1.3. Sterowanie przez użytkownika	8
3.2. Zadanie 4	9
3.2.1. Powtórzone testy	9
3.2.2. Eksperyment 1	9
3.2.3. Eksperyment 2	10
3.2.4. Eksperyment 3	11
3.2.5. Pozostałe eksperymenty	12
3.3. Wnioski	12
4. Podsumowanie	13
5. Bibliografia	13

1. Wstęp

Laboratorium nr 2 z przedmiotu Sieci i Chmury Teleinformatyczne dotyczyło płaszczyzny sterowania sieci. Przed przystąpieniem do wykonania zadań zainstalowaliśmy maszynę wirtualną z **emulatorem sieci Mininet** i kontrolerem **ONOS**. Zapoznaliśmy się z dokumentacją kontrolera oraz przetestowaliśmy jego podstawowe funkcjonalności na domyślnej sieci.

2. Część 1

W pierwszej kolejności rozszerzyliśmy naszą sieć, opracowaną w ramach poprzedniego ćwiczenia laboratoryjnego, o dodatkowe łącza. Dzięki temu uzyskaliśmy sieć, której graf nie jest już drzewem. Dodatkowe łącza zostały dodane do skryptu z poprzedniego ćwiczenia definiującego topologię sieci, a ich charakterystyki obliczone przez przygotowane w nim algorytmy. Mapę sieci w jej pełnej wersji przedstawiliśmy na rysunku 1.



Rys. 1. Mapa sieci w jej pełnej wersji

Następnie podjęliśmy się konfiguracji kontrolera ONOS. Aktywowaliśmy aplikację OpenFlow, aby kontroler mógł zarządzać switchami. Z wykorzystaniem protokołu OpenFlow kontroler może w szczególności dodawać, aktualizować i usuwać flowy. Naszą sieć z kontrolerem ONOS uruchamialiśmy poleceniem `sudo mn --custom main.py --topo mytopo --link tc --mac --switch ovs,protocols=OpenFlow14 --controller remote,ip=192.168.43.18 --arp`. Podobnie jak podczas realizacji pierwszego ćwiczenia, pliki pomiędzy maszyną wirtualną a naszym komputerem przesyialiśmy z wykorzystaniem narzędzia `pscp`.

2.1. Zadanie 1

W ramach zadania pierwszego skonfigurowaliśmy switche w taki sposób, aby możliwe było zrealizowanie tych samych sesji transportowych, tymi samymi drogami przesyłu danych, co w zadaniu czwartym z poprzedniego ćwiczenia laboratoryjnego (pojedyncze sesje TCP i UDP dla relacji Malmo - Graz (h1 - h10) i Hanower - Norymberga (h4 - h6)). W tym celu należało przygotować flowy zgodnie z dokumentacją **styku REST** kontrolera ONOS, a następnie wysłać je do kontrolera.

W celu łatwiejszego tworzenia pliku konfiguracyjnego w formacie **JSON** dla konkretnych relacji, przygotowaliśmy metodę `add_flow`. Tworzy ona flowa, który dla danego urządzenia przekierowuje na wybrany port wyjściowy pakiet z konkretnego portu wejściowego i z danym docelowym adresem IP. Wymienione kryteria zostały sparametryzowane we wspomnianej metodzie.

Następnie wywoływaliśmy metodę `add_flow` dla połączeń realizowanych w poprzednim ćwiczeniu. Poniżej przedstawiliśmy przykład użycia tej metody do konfiguracji połączenia Malmo - Graz (h1-s1-s2-s5-s7-s9-s10-h10). Całość konfiguracji znajduje się w pliku `flows_1.py`.

```
add_flow(1, 2, 1, 10)
add_flow(2, 5, 2, 10)
add_flow(5, 4, 2, 10)
add_flow(7, 4, 2, 10)
add_flow(9, 3, 2, 10)
add_flow('a', 1, 2, 10)
add_flow('a', 2, 1, 1)
add_flow(9, 2, 3, 1)
add_flow(7, 2, 4, 1)
add_flow(5, 2, 4, 1)
add_flow(2, 2, 5, 1)
add_flow(1, 1, 2, 1)
```

Wszystkie reguły zapisywaliśmy do pliku `result.json`, a następnie używaliśmy narzędzia `curl` w celu wysłania flowów do kontrolera przez styl REST API. Zgodnie z poleceniem skorzystaliśmy ze skryptu `bat` wywołującego narzędzie `curl`.

```
curl --user onos:rocks -X POST -H "Content-Type:application/json" -H "Accept:application/json" -d @result.json "http://192.168.43.18:8181/onos/v1/flows"
```

Po umieszczeniu wpisów, upewniliśmy się o poprawności konfiguracji węzłów wykonując polecenie `ping` dla skonfigurowanych relacji. Pakiety przesyłane były prawidłowo.

Dzięki przeprowadzeniu opisanych wyżej konfiguracji, narzędziem **Iperf** mogliśmy powtórzyć testy z zadania czwartego z poprzedniego ćwiczenia laboratoryjnego, tym razem z wykorzystaniem kontrolera ONOS. Poniżej przedstawiliśmy wyniki testów oraz ich porównanie z wynikami uzyskanymi w poprzednim ćwiczeniu, w którym korzystaliśmy z domyślnego kontrolera. Testy przeprowadziliśmy przy przepustowości łącznej wynoszącej $10\frac{Mb}{s}$.

2.1.1. TCP

Manipulując parametrami narzędzia Iperf (pozostawiając przy tym pozostałe parametry na domyślnych wartościach) przyglądamy się ewentualnym różnicom w parametrach jakości przesyłu danych protokołem TCP, względem wartości z poprzedniego ćwiczenia. Porównanie tych wartości przedstawiliśmy w tabelach 1, 2 i 3. Tak jak w przypadku poprzedniego ćwiczenia, przesyłaliśmy dane o wielkości 10MB.

Maksymalna przepustowość (parametr -b)

Maks. przepustowość [Mb/s]	Zbadana przepustowość [Mb/s]	Poprzednio [Mb/s]	Różnica [Mb/s]
8	8,10	8,06	+0,04
10	10,10	9,89	+0,21
12	9,87	10,00	-0,13

Tab. 1. Porównanie przepustowości uzyskanych przy manipulacji parametrem `-b` dla relacji Malmo - Graz

Wielkość bufora (parametr -l)

Wielkość bufora [B]	Przepustowość [Mb/s]	Poprzednio [Mb/s]	Różnica [Mb/s]
2	5,96	5,77	+0,19
6	9,41	8,95	+0,46
10	9,69	9,66	+0,03

Tab. 2. Porównanie przepustowości uzyskanych przy manipulacji parametrem `-l` dla relacji Malmo - Graz

Wielkość okna (parametr -w)

Wielkość okna [kB]	Przepustowość [Mb/s]	Poprzednio [Mb/s]	Różnica [Mb/s]
40	8,88	8,64	+0,24
60	9,11	9,49	-0,38
80	9,14	9,48	-0,34

Tab. 3. Porównanie przepustowości uzyskanych przy manipulacji parametrem -w dla relacji Malmo - Graz

Testy dla drugiej relacji

Podobnie jak w poprzednim ćwiczeniu, takie same testy przeprowadziliśmy dla relacji Hanower - Norymberga (h4 - h6). Uzyskane przepustowości pokrywały się tymi zbadanymi w poprzednim ćwiczeniu (odnotowano różnice nie większe niż $0,40 \frac{Mb}{s}$).

2.1.2. UDP

Powtórzyliśmy testy także dla protokołu UDP. Manipulując parametrami -b (liczba bloków wysyłanych na sekundę) i -l (wielkość bloku), identycznie jak w poprzednim ćwiczeniu obserwowaliśmy bezpośredni wpływ iloczynu tych parametrów na jakość przesyłu danych. Powtórzyliśmy zatem testy manipulując jedynie wielkością bloku, przy $1024 pps$. Porównanie wartości przedstawiliśmy w tabeli 4.

Wielkość bloku [B]	Prędkość przesyłu [Mb/s]	Jitter [ms]	Jitter poprzednio [ms]	Strata pakietów [%]	Strata poprzednio [%]	Średnie opóźnienie [ms]	Opóźnienie poprzednio [ms]
1024	8,39	0,931	0,442	0	0	41,33	12,19
1280	10,50	0,767	0,378	8,2	1,5	917,68	656,37
1536	12,60	0,784	0,476	30	46	736,52	789,12

Tab. 4. Porównanie parametrów jakości przesyłu danych protokołem UDP dla relacji Malmo - Graz

Analogicznie jak w poprzednim ćwiczeniu, odnotowano straty pakietów i wzrost opóźnienia w miarę zbliżania oraz po przekroczeniu przepustowości łączy przez prędkość przesyłu danych. Wartości parametrów są w przybliżeniu równe tym uzyskanym poprzednio.

Testy dla drugiej relacji

Takie same testy przeprowadziliśmy dla relacji Hanower - Norymberga (h4 - h6) i uzyskaliśmy bardzo zbliżone wartości parametrów jakości przesyłu danych, co w poprzednim ćwiczeniu. Dla prędkości przesyłu mniejszych niż przepustowość łączy jakość transmisji była dobra, a przy przekroczeniu przepustowości łączy o $2,6 \frac{Mb}{s}$ odnotowano wzrost opóźnienia oraz stratę pakietów na poziomie 39%.

2.1.3. Wnioski

Powtarzając testy z poprzedniego ćwiczenia z wykorzystaniem kontrolera ONOS, w obu przypadkach (dla protokołu TCP i UDP), uzyskaliśmy w przybliżeniu takie same wartości parametrów przesyłu danych. Potwierdziliśmy zatem poprawną konfigurację węzłów przy użyciu styku REST.

2.2. Zadanie 2

W ramach zadania drugiego należało zrealizować te same sesje transportowe co w zadaniu piątym z poprzedniego ćwiczenia (jednoczesne sesje TCP i UDP) w taki sposób, aby uzyskać poprawę wartości parametrów przesyłu danych. W tym celu skonfigurowaliśmy węzły z wykorzystaniem styku REST kontrolera ONOS tak, aby strumienie przesyłane były innymi, optymalnymi drogami (wykorzystując łącza dodane w ramach rozwinięcia sieci do jej pełnej wersji).

W naszym przypadku badaną parą relacji była para Hanower - Norymberga (h4 - h6) i Gdańsk - Wiedeń (h3 - h7). Drogi przesyłu danych dla tych relacji skonfigurowaliśmy następująco:

- Hanower - Norymberga: h4-s4-s6-h6 (poprzednio: h4-s4-s2-s5-s6-h6)
- Gdańsk - Wiedeń: h3-s3-s8-s7-h7 (poprzednio: h3-s3-s2-s5-s7-h7).

Następnie przystąpiliśmy do obserwacji ewentualnej poprawy parametrów jakości przesyłu danych, wykonując te same testy (dla poszczególnych kombinacji sesji transportowych) co w ramach poprzedniego ćwiczenia. Tak jak w zadaniu pierwszym, podczas testów przepustowość łączy wynosiła $10 \frac{Mb}{s}$.

2.2.1. TCP

Badając parametry jakości przesyłu danych dla dwóch realizowanych jednocześnie sesji TCP, analogicznie jak w poprzednim ćwiczeniu, przesyłaliśmy dane o wielkości 30MB. Ponownie ograniczaliśmy przepustowość parametrem -b i analizowaliśmy uzyskiwane wartości. Wyniki testów z obu ćwiczeń zostały przedstawione w tabeli 5.

Maks. przepustowość [Mb/s]	Zbadana przepustowość [Mb/s]	Przepustowość poprzednio [Mb/s]	Czas przesyłu [s]	Czas poprzednio [s]	Retries	Retries poprzednio
3 / 3	3,01 / 3,01	3,01 / 2,74	83,54 / 83,54	83,73 / 91,94	0 / 0	475 / 498
5 / 5	5,02 / 5,02	4,91 / 4,75	50,12 / 50,12	51,24 / 52,99	0 / 0	379 / 331
7 / 7	7,03 / 7,03	6,19 / 4,98	35,80 / 35,80	40,64 / 50,56	0 / 0	291 / 338
10 / 10	10,00 / 10,00	5,54 / 5,85	25,06 / 25,06	45,43 / 42,99	212 / 262	266 / 213

Tab. 5. Porównanie zbadanych z wykorzystaniem dwóch strumieni TCP parametrów jakości obsługi ruchu dla relacji Hanower - Norymberga i Gdańsk - Wiedeń (<wynik dla Hanower - Norymberga> / <wynik dla Gdańsk - Wiedeń>)

Testy pokazały, że po optymalnym skonfigurowaniu dróg uzyskaliśmy widoczną poprawę jakości przesyłu danych. Zaobserwowaliśmy wykorzystywaną w pełni przepustowość łącz, a co za tym idzie - znacznie krótszy czas przesyłu względem poprzedniego ćwiczenia. Odpowiedni dobór dróg sprawił również, że pakiety musiały być wysyłane ponownie rzadziej niż poprzednio.

Następnie przeprowadziliśmy analogiczny eksperyment dla trzech sesji TCP, dodając sesję dla relacji Malmo - Praga (dla tej relacji, mimo rozszerzenia sieci do jej pełnej wersji, optymalną drogą przesyłu danych pozostała ta z poprzedniego ćwiczenia: h1-s1-s2-s5-h5). W tym eksperymencie również zaobserwowaliśmy znaczną poprawę jakości przesyłu danych. W przypadku każdej z relacji dane przesyłane były z maksymalną przepustowością udostępnianą przez łącz. Osiągnęliśmy zatem trzykrotny (w przybliżeniu) wzrost przepustowości uzyskanej dla konkretnej relacji, względem poprzedniego ćwiczenia.

2.2.2. UDP

Powtórzyliśmy również testy dla realizowanych jednocześnie sesji UDP, wykonane w poprzednim ćwiczeniu. Ponownie manipulowaliśmy wielkością bloku przy stałych 1024pps. Wyniki przeprowadzonych testów, na tle wyników z poprzedniego ćwiczenia, zostały przedstawione w tabeli¹ 6.

Wielkość bloku [B]	Prędkość przesyłu [Mb/s]	Jitter [ms]	Jitter poprzednio [ms]	Strata pakietów [%]	Strata poprzednio [%]	Średnie opóźnienie [ms]	Opóźnienie poprzednio [ms]
768	6,29	0,557 / 0,696	0,370 / 1,609	0 / 0	24 / 19	3,39 / 6,56	600 / 605
1024	8,39	0,304 / 0,420	0,271 / 0,654	0 / 0	44 / 39	3,55 / 6,74	834 / 837
1152	9,44	0,407 / 0,544	1,252 / 0,197	0 / 0	48 / 47	4,84 / 8,48	934 / 937
1280	10,50	0,577 / 0,570	9,914 / 4,164	0 / 0	66 / 31	458,5 / 463,7	1090 / 1123

Tab. 6. Porównanie zbadanych z wykorzystaniem dwóch strumieni UDP parametrów jakości obsługi ruchu dla relacji Hanower - Norymberga i Gdańsk - Wiedeń (<wynik dla Hanower - Norymberga> / <wynik dla Gdańsk - Wiedeń>)

Podobnie jak w przypadku sesji TCP, dla sesji UDP w tym zadaniu zaobserwowaliśmy znaczną poprawę jakości przesyłu danych. Dzięki wysłaniu strumieni drogami nieobejmującymi wspólnych łącz, w żadnej z sesji nie wystąpiły straty pakietów (nawet przy prędkościach przesyłu na poziomie przepustowości łącz). Dla porównania w poprzednim ćwiczeniu, przy tych samych prędkościach przesyłu danych, odnotowane zostały straty nawet większe niż 50%. Ogromnemu zmniejszeniu uległo także średnie opóźnienie, natomiast dla największej badanej prędkości przesyłu nastąpił także duży spadek jitteru.

¹ W komórkach tabeli, w których nie występuje znak '/', wartość w nich się znajdująca jest taka sama dla obu przypadków rozdzielonych w tabeli tym znakiem.

2.2.3. TCP i UDP

Na koniec części drugiej przeprowadziliśmy ponownie także ostatni eksperyment z poprzedniego ćwiczenia laboratoryjnego, oczywiście przy zmodyfikowanych (zgodnie z opisem we wstępie) drogach przesyłu danych. Ponownie przesyliśmy dane o wielkości 30MB w sesji TCP pomiędzy Gdańskiem a Wiedniem, przy jednocześnie (trwającej 30 sekund) sesji UDP pomiędzy Hanowerem a Norymbergą. Uzyskane wyniki zostały zestawione z wynikami z poprzedniego ćwiczenia w tabeli¹ 7 (należy zwrócić uwagę na inną niż w przypadku wcześniejszych tabel konwencję prezentacji wartości - szczegółowy opis pod tabelą).

Wielkość bloku UDP [B]	Prędkość przesyłu danych UDP [Mb/s]	Strata UDP [%]	Przesłane dane w ciągu pierwszych 30s TCP [MB]	Czas przesyłu danych TCP [s]
768	6,29	0,0 / 0,8	30,00 / 15,02	25,14 / 43,58
1024	8,39	0,0 / 1,3	30,00 / 7,56	25,14 / 47,43
1280	10,50	12,0 / 5,8	30,00 / 3,15	25,16 / 49,66
1536	12,60	45,0 / 45,6	30,00 / 2,21	25,06 / 50,45

Tab. 7. Porównanie zbadanych z wykorzystaniem jednego strumienia TCP (w relacji Gdańsk - Wiedeń) i jednego UDP (w relacji Hanover - Norymberga) parametrów jakości obsługi ruchu (<wynik uzyskany w ramach tego ćwiczenia> / <wynik z poprzedniego ćwiczenia>)

Jak wynika z powyższej tabeli, testy wykazały poprawę jakości przesyłu danych w przypadku sesji TCP. W poprzednim ćwiczeniu sesja UDP, wykorzystując współdzielone łącze, podczas swojego trwania w dużym stopniu uniemożliwiała sesji TCP przesyłanie danych. Konfiguracja dróg przesyłu nie obejmująca wspólnego łącza, pozwoliła sesjom przesyłać dane niezależnie od siebie. Dzięki temu, mogliśmy zaobserwować szybką i skuteczną transmisję TCP oraz jakościową transmisję UDP, gdy prędkość przesyłu danych nie przekraczała przepustowości sieci.

2.2.4. Wnioski

Przeprowadzone testy i obserwacje pokazały jak ważny, poza odpowiednim doborem parametrów przesyłu, jest dobór dróg, którymi podążają pakiety. Dzięki pokierowaniu strumieni danych innymi drogami, uniknęliśmy współdzielenia łączy przez strumienie i co za tym idzie - rywalizacji o zasoby. Po skonfigurowaniu w sposób optymalny dróg przesyłu danych uzyskaliśmy realną poprawę jakości transmisji.

3. Część 2

Druga część ćwiczenia dotyczyła zautomatyzowania procesu konfiguracji połączeń, na potrzebę optymalnego realizowania sesji transportowych. W ramach realizacji tej części przygotowaliśmy aplikację w języku Python oraz wykonaliśmy z jej wykorzystaniem eksperymenty na naszej sieci.

3.1. Zadanie 3

Stworzenie wspomnianej aplikacji było głównym elementem zadania trzeciego. W celu zachowania zwięzości sprawozdania, poniżej zostały opisane wyłącznie kluczowe i nietrywialne elementy implementacji. Pełen kod aplikacji został natomiast przesłany wraz ze sprawozdaniem.

3.1.1. Plik konfiguracyjny

W pierwszej kolejności zajęliśmy się przygotowaniem pliku konfiguracyjnego, z którego nasza aplikacja wczytuje informacje o węzłach i łączach sieci. Skorzystaliśmy w tym celu z formatu pliku JSON. W pliku `network.json` znajduje się lista wszystkich hostów (nazw miast), a także lista łączy między nimi. Dla każdego łącza określony jest jego początek i koniec (poprzez miasta oraz numery portów), a także jego przepustowość oraz opóźnienie. W celu uniknięcia tworzenia pliku ręcznie, zmieniliśmy skrypt uruchamiający naszą sieć tak, by automatycznie go generował (bez uwzględniania numerów portów). Przygotowaliśmy także krótki skrypt `ports.py`, pobierający wszystkie łącza używając stylisty REST. Skrypt pobiera z kontrolera numery portów i dodaje je do utworzonego pliku konfiguracyjnego.

3.1.2. Zarządzanie siecią

Zarządzanie sesjami jest realizowane w naszej aplikacji przez klasę `Manager`. Klasa ta, przyjmuje ścieżkę do pliku z konfiguracją sieci. Na podstawie wczytanych danych tworzone są obiekty reprezentujące

switcha i łącza. Ustawiany jest także adres IP kontrolera ONOS, aby aplikacja mogła używać styku REST API do wysyłania flowów.

Za tworzenie nowej ścieżki w klasie `Manager` odpowiada metoda `add_path` (nie została umieszczona w sprawozdaniu ze względu na swoją długość). Przyjmuje ona jako argumenty: hosta początkowego i końcowego, typ sesji (TCP, UDP lub sesja do pingowania) i wymaganą minimalną przepustowość. Na etapie projektowania założyliśmy, że między dowolnymi dwoma hostami może istnieć maksymalnie jedna sesja TCP i maksymalnie jedna sesja UDP, lub jedynie sesja do pingowania.

Po zweryfikowaniu, że wymagania te są spełnione, klasa `Manager` próbuje znaleźć optymalną drogę przesyłu dla nowej sesji. W metodzie `find_shortest`, wykorzystującej bibliotekę `networkx`, tworzony jest graf składający się **jedynie z łączami, na których sesja będzie mogła być realizowana w kontekście dostępnej przepustowości**. Określanie, czy łącze może obsługiwać nową sesję, zapewniane jest przez klasę `Link`. Każde łącze przechowuje wszystkie sesje, które są z jego wykorzystaniem realizowane.

Dzięki wykonaniu zadań w ramach poprzedniego ćwiczenia laboratoryjnego wiemy jak zachowują się sesje TCP i UDP, gdy współdzielą ze sobą łącze. Założyliśmy, że każda sesja UDP wykorzystuje dokładnie tyle przepustowości, ile zostanie dla niej określone, a następnie reszta sesji TCP dzieli się pozostałą przepustowością po równo. Na tej podstawie wywnioskowaliśmy wzory na **maksymalną możliwą przepustowość** dostępną dla nowej sesji TCP i UDP na danym łączu.

W przypadku UDP należy znaleźć taką sesję TCP, dla której minimalna wymagana przepustowość jest największa, a następnie pomnożyć ją przez liczbę realizowanych sesji TCP. Uzyskamy wtedy minimalną przepustowość, która musi być pozostawiona dla realizowanych już sesji TCP, aby przepustowość żadnej z nich nie spadła poniżej wymaganej. Tę wartość oraz sumę przepustowości wszystkich realizowanych już sesji UDP, odejmujemy od przepustowości łączego i otrzymujemy ostateczny wynik **dla UDP**.

W przypadku TCP natomiast, odejmujemy od przepustowości łączego sumę przepustowości wszystkich realizowanych już sesji UDP. Uzyskujemy w ten sposób część przepustowości dostępną dla sesji TCP, którą sesje tego typu dzielą po równo. Musimy sprawdzić następnie, czy przy podzieleniu tej wartości przez liczbę istniejących sesji TCP powiększonej o 1 wartość, którą uzyskamy, nie będzie mniejsza niż minimalna przepustowość kolejnego sesji TCP. Jeżeli wszystkie sesje spełniają ten warunek, to wartość ta jest ostatecznym wynikiem **dla TCP**. W przeciwnym wypadku, łącze to nie może obsługiwać kolejnej sesji TCP. Obliczenia te realizowane są przez metodę `max_possible`, przedstawioną poniżej.

```
def max_possible(self):
    tcp_min = max((s.bandwidth for s in self.tcp_sessions),
                  default=0) * len(self.tcp_sessions)
    udp = self.max_bandwidth - tcp_min - sum(
        s.bandwidth for s in self.udp_sessions)
    tcp_part = (self.max_bandwidth - sum(
        s.bandwidth for s in self.udp_sessions)) / (
            len(self.tcp_sessions) + 1)
    if all(tcp_part >= s.bandwidth for s in self.tcp_sessions):
        return udp, tcp_part
    return udp, 0
```

Klasa `Link` posiada także metodą `can_handle`, która korzystając z powyższej metody określa czy sesja, o której użytkownik prosi, zmieści się na łączu i czy te łącze powinno zostać dodane do grafu.

```
def can_handle(self, session):
    if session.session_type == 'PING':
        return True
    udp_max, tcp_max = self.max_possible()
    if session.session_type == 'UDP':
        return session.bandwidth <= udp_max
    return session.bandwidth <= tcp_max
```

Po dodaniu wszystkich łącz, które mogą obsługiwać sesję, używamy funkcji oferowanej przez bibliotekę `networkx` - `find_shortest`, która używając algorytmu Dijkstry znajduje najkrótszą możliwą ścieżkę. Jako waga podawana jest funkcja, która pobiera **wartość opóźnienia z łączami**. Jeżeli ścieżka zostanie znaleziona, to uzyskiwane są wszystkie łącza, które wykorzystuje, a następnie z użyciem przechowywanych w nich danych tworzone są flowy, w których kryteriami są: port wejściowy, IP hosta początkowego, IP hosta docelowego i typ sesji. Flowy w formacie JSON wysyłane są przez styl REST do kontrolera ONOS. Odpowiedzią jest lista wszystkich id flowów i nazw switchów, dla których zostały one utworzone. Wraz z

wyliczoną ścieżką zapisywane są w obiekcie reprezentującym sesję, a sesja ta jest dodawana do listy sesji dla wszystkich łącz, przez które przechodzi oraz do listy wszystkich sesji w klasie `Manager`.

Dodaliśmy również funkcjonalność obejmującą kończenie wcześniejszej utworzonej sesji, aby ułatwić późniejsze testowanie, bez konieczności resetowania kontrolera. Dane zapisane w obiekcie sesji wykorzystywane są do usunięcia jej z każdego łącza, przez które przechodziła. Następnie, przy użyciu styku REST, wysyłane jest żądanie usunięcia wszystkich flowów zapisanych w tej sesji. W takiej sytuacji ONOS nie kieruje już pakietów na tej drodze w ustalony wcześniej sposób i przepustowość oferowana przez te łączta staje się dostępna dla nowych sesji.

Dodatkowo w klasie `Manager` utworzyliśmy bardzo przydatną w testach metodę o nazwie `test_between`. Znajduje ona najkrótsze ścieżki dla kolejnych progów minimalnej przepustowości. Może zdarzyć się bowiem tak, że najkrótszą pod względem opóźnienia drogą może być realizowana jedynie sesja o niewielkiej przepustowości, natomiast sesje o większej przepustowości muszą być kierowane dłuższą ścieżką. Metoda sortuje malejąco łączta po maksymalnej możliwej przepustowości sesji, jaką mogą zrealizować. Następnie, dla każdej kolejnej grupy łącz o tej samej przepustowości znajdowane są najkrótsze ścieżki tym samym algorytmem, co w metodzie `add_path`. Wynikiem tych operacji są kolejne progi przepustowości, dla których ścieżka ulegnie zmianie.

Klasa `Manager` oferuje również możliwość czytelnego wyświetlania aktywnych sesji. Wyświetlane są: typ sesji, minimalna przepustowość, szacowana przepustowość, ścieżka i suma opóźnień na łączach. W celu obliczania szacowanej przepustowości założyliśmy, że sesje UDP zawsze wykorzystują tyle przepustowości ile żądano, a sesje TCP dzielą się dostępną przepustowością po równo - mogą zatem często wykorzystywać więcej niż żądano. Szacowaną przepustowość jest minimum wyników poniższej metody dla każdego łącza w sesji.

```
def estimate_bandwidth(self, session):
    if session.session_type == 'PING':
        return 0
    if session.session_type == 'UDP':
        return session.bandwidth
    return (self.max_bandwidth - sum(
        s.bandwidth for s in self.udp_sessions)) / len(self.tcp_sessions)
```

Przygotowaliśmy także metodę `generate_iperf`, tworzącą plik TXT z poleceniami generatora ruchu Iperf dla nowej sesji zatwierdzonej przez aplikację. Metoda była pomocna podczas testowania parametrów przesyłu danych przeprowadzonych w dalszej części ćwiczenia.

3.1.3. Sterowanie przez użytkownika

Po uruchomieniu aplikacji program próbuje załadować adres IP ONOSa z pliku `ip.txt`. W przypadku, gdy nie znajduje takiego pliku, prosi użytkownika o podanie IP. Następnie pojawia się informacja o możliwości wyświetlenia dostępnych poleceń, poprzez wpisanie `help` w konsoli. Program w pętli oczekuje na podanie danych wejściowych przez użytkownika. Dostępne polecenia to:

- `help` - wyświetla informację o wszystkich dostępnych poleceniach
- `hosts` - wyświetla listę wszystkich dostępnych w sieci hostów
- `ping <start_host> <end_host>` - tworzy ścieżkę do pingowania między hostem `start_host`, a hostem `end_host`
- `start <start_host> <end_host> <session_type> <minimum_bandwidth>` - tworzy ścieżkę dla sesji o typie `session_type`, między hostem `start_host` i hostem `end_host`, o minimalnej wymaganej przepustowości `minimum_bandwidth`
- `list` - wyświetla wszystkie realizowane obecnie sesje
- `end <session_id>` - kończy sesję o id `session_id`
- `source <file>` - wykonuje wszystkie polecenia zawarte w pliku ze ścieżki `file`
- `test <host_a> <host_b> <session_type>` - wykonuje test dla sesji między hostem `<host_a>` i hostem `<host_b>`, o typie `session_type`, opisany wyżej przy metodzie `test_between`
- `exit` - kończy wykonywanie programu, kończąc przed tym wszystkie realizowane sesje.

Przy każdym poleceniu wydawanym przez użytkownika, program sprawdza liczbę argumentów, istnienie hostów o podanych nazwach, poprawność typu sesji i poprawność formatu podanych liczb. Oto przykład działania aplikacji:

```

start malmo graz tcp 5
Utworzono nowa sciezke:
[0]: Type: TCP, Requested: 5.0 Mb/s, Estimate: 10.0 Mb/s, Path: [Malmo -> Berlin
-> Praga -> Wieden -> Graz], Link delay: 7.21 ms
start malmo graz udp 8
Utworzono nowa sciezke:
[1]: Type: UDP, Requested: 8.0 Mb/s, Estimate: 8.0 Mb/s, Path: [Malmo -> Hanower
-> Norymberga -> Zagrzeb -> Graz], Link delay: 10.25 ms
test malmo wieden tcp
Max: 5.0 Mb/s, Path: [Malmo -> Berlin -> Praga -> Wieden], Link delay: 6.18 ms
Max: 10.0 Mb/s, Path: [Malmo -> Gdansk -> Ostrawa -> Wieden], Link delay: 7.92 ms
list
[0]: Type: TCP, Requested: 5.0 Mb/s, Estimate: 10.0 Mb/s, Path: [Malmo -> Berlin
-> Praga -> Wieden -> Graz], Link delay: 7.21 ms
[1]: Type: UDP, Requested: 8.0 Mb/s, Estimate: 8.0 Mb/s, Path: [Malmo -> Hanower
-> Norymberga -> Zagrzeb -> Graz], Link delay: 10.25 ms
end 0
Usunieto sesje

```

3.2. Zadanie 4

W ramach zadania czwartego przeprowadziliśmy serię eksperymentów, w celu przetestowania i potwierdzenia poprawności działania przygotowanej aplikacji. W pierwszej kolejności podjęliśmy się powtórzenia testów wykonanych w zadaniu piątym z poprzedniego ćwiczenia laboratoryjnego. Trudność zilustrowania pełnych możliwości przygotowanej implementacji na ich przykładzie skłoniła nas jednak do zrealizowania dodatkowo kilku nowych eksperymentów. Przypadki eksperimentalne zostały zaprojektowane tak, aby zobrazować sposób doboru ścieżek przez aplikację w różnych sytuacjach. Podczas wykonywania testów przepustowość sieci została ustawiona na $10 \frac{Mb}{s}$.

3.2.1. Powtórzone testy

Powtórzyliśmy testy dla badanej w zadaniu piątym z poprzedniego ćwiczenia pary relacji: Hanower - Norymberga i Gdańsk - Wiedeń. Z wykorzystaniem naszej aplikacji uruchomiliśmy sesje transportowe dla tych relacji. Program konfigurował połączenia wysyłając flowy do kontrolera oraz generował pliki tekstowe z poleceniami narzędzia Iperf, które zostały przez nas użyte do zbadania parametrów jakości przesyłu danych. Zgodnie z oczekiwaniami, wyznaczone przez naszą aplikację optymalne drogi przesyłu danych dla poszczególnych sesji okazały się identyczne, jak te ustalone przez nas ręcznie w zadaniu drugim w tym ćwiczeniu (2.2). Poskutkowało to uzyskaniem w przybliżeniu identycznych wartości parametrów jakości przesyłu, co w zadaniu drugim. Z wykorzystaniem naszej aplikacji i kontrolera ONOS uzyskaliśmy zatem widoczną poprawę jakości przesyłu danych, w stosunku do jakości uzyskanej przy użyciu domyślnego kontrolera.

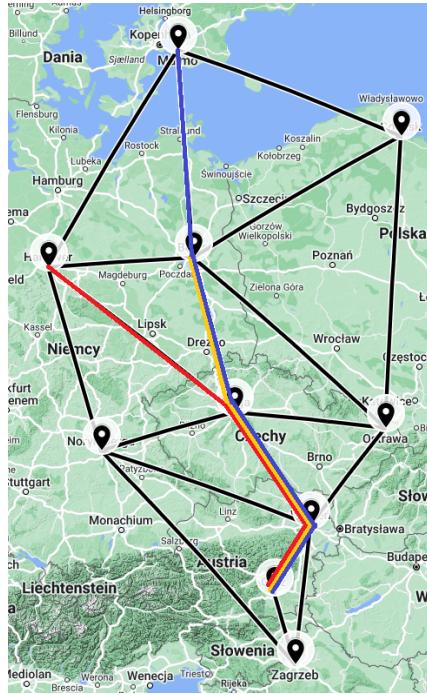
3.2.2. Eksperiment 1

Pierwszy eksperiment obejmował realizację trzech sesji TCP, dla relacji: Hanower - Graz (ozn. 1), Berlin - Graz (ozn. 2) i Malmo - Graz (ozn. 3). Trzech klientów wysyłało strumienie danych do serwera umieszczonego w Grazie.

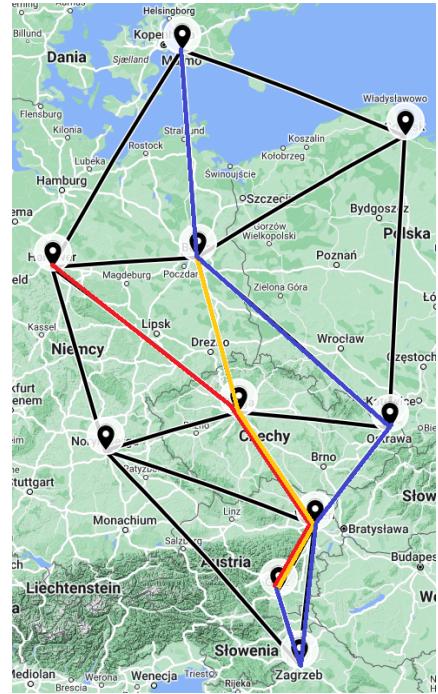
Najpierw uruchomiliśmy te trzy sesje, podając jako minimalną wymaganą przepustowość $3 \frac{Mb}{s}$ dla każdej z nich. W efekcie, dla każdego ze strumieni danych zostały wybrane najkrótsze (pod względem opóźnienia) drogi, obejmujące łączą współdzielone z pozostałymi strumieniami (rys. 2). Stało się tak, ponieważ suma minimalnych przepustowości wymaganych przez sesje TCP nie przekroczyła przepustowości sieci. Protokół TCP w przybliżeniu sprawiedliwie rozdzielił przepustowości pomiędzy sesje, odnotowano przepustowości kolejno: $4, 26 \frac{Mb}{s}; 3, 39 \frac{Mb}{s}; 3, 70 \frac{Mb}{s}$.

Następnie zbadaliśmy zachowanie aplikacji w kontekście doboru dróg w sytuacji, gdy suma żądanych przez sesje minimalnych przepustowości przekracza przepustowość sieci. Wymagana minimalna przepustowość wynosiła $4 \frac{Mb}{s}$ dla każdej sesji. Aplikacja uwzględniała, że dla sesji nr 3 (uruchomionej jako ostatniej) nie wystarczy przepustowości na łączach współdzielonych przez dwie zatwierdzone już sesje. W związku z tym, strumień danych dla sesji nr 3 został pokierowany inną, dłuższą pod względem opóźnienia drogą (rys. 3). Przepustowość dla dwóch sesji współdzielących łączą została przydzielona (z wykorzys-

taniem całej dostępnej przepustowości łączy) w przybliżeniu po równo: $4,81 \frac{Mb}{s}$ i $5,46 \frac{Mb}{s}$. Sesja nr 3 samodzielnie osiągnęła natomiast przepustowość w przybliżeniu równą przepustowości sieci.



Rys. 2. Przypadek dla minimalnego wymaganego bw $3 \frac{Mb}{s}$



Rys. 3. Przypadek dla minimalnego wymaganego bw $4 \frac{Mb}{s}$

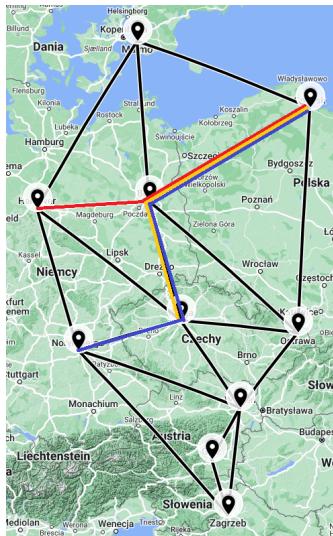
3.2.3. Eksperyment 2

W drugim eksperymencie przeprowadziliśmy testy podobnego typu co w eksperymencie pierwszym, tym razem jednak dla sesji UDP. Sesje realizowane były dla następujących relacji: Hanower - Gdańsk (ozn. 1), Praga - Gdańsk (ozn. 2) i Norymberga - Gdańsk (ozn. 3). Wyróżniliśmy trzy przypadki eksperymentalne.

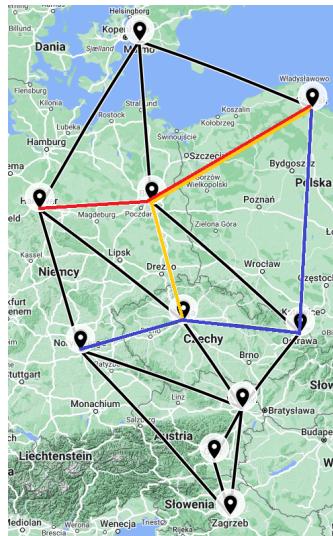
W pierwszym z nich wymagana minimalna prędkość przesyłu wynosiła $2 \frac{Mb}{s}$ dla każdej sesji. Strumienie zostały poprowadzone z wykorzystaniem współdzielonego łącza (rys. 4), ponieważ najkrótsze pod względem opóźnienia drogi dla poszczególnych sesji obejmowały te łącza, a suma wymaganych prędkości przesyłu była mniejsza od przepustowości sieci. Jakość transmisji była niezakłócona, nie zostały odnotowane straty pakietów.

Drugi przypadek obejmował sytuację, w której jedna z sesji nie mogła być zrealizowana przez łącze współdzielone przez dwie zatwierdzone już sesje. W tym celu, minimalna wymagana prędkość przesyłu została ustawiona na $4 \frac{Mb}{s}$ dla każdej sesji. W efekcie, strumień sesji nr 3 (uruchomionej jako ostatniej) został pokierowany inną drogą (rys. 5). Dzięki temu, każda sesja została zrealizowana bez strat pakietów.

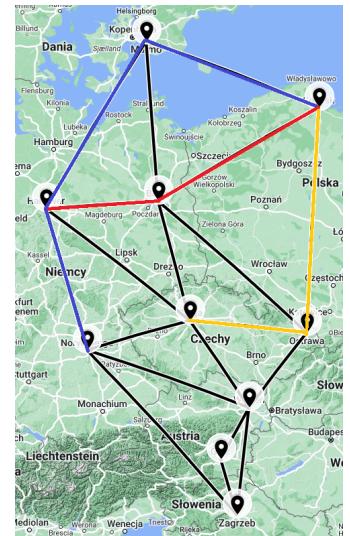
W trzecim przypadku, dla każdej sesji minimalna wymagana prędkość przesyłu wynosiła $6 \frac{Mb}{s}$. Przepustowość łącza, wymagana do bezstratnej i jakościowej realizacji dowolnych dwóch sesji UDP, wynosiła więc $12 \frac{Mb}{s}$, czyli więcej niż przepustowość łącz w naszej sieci. Każdy ze strumieni został pokierowany zatem taką drogą, aby uniknąć współdzielenia jakiegokolwiek łącza (rys. 6). Najkrótszą pod względem opóźnienia i nieobejmującą współdzielonych łącz drogą w przypadku sesji nr 2 (uruchomionej jako drugiej) okazała się droga przez Ostrawę. W przypadku sesji nr 3 (uruchomionej jako ostatniej) była to droga przez Hanower oraz Malmö. W tym przypadku jakość transmisji była również niezakłócona, bez strat pakietów.



Rys. 4. Przypadek dla min. wymaganego bw $2 \frac{Mb}{s}$



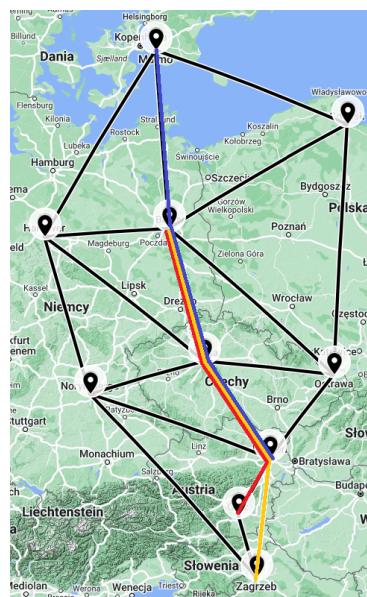
Rys. 5. Przypadek dla min. wymaganego bw $4 \frac{Mb}{s}$



Rys. 6. Przypadek dla min. wymaganego bw $6 \frac{Mb}{s}$

3.2.4. Eksperyment 3

W trzecim eksperymencie przetestowaliśmy zachowanie aplikacji w przypadku uruchomienia jednocześnie dwóch sesji TCP i UDP. Sesje TCP zostały uruchomione dla relacji Graz - Berlin oraz Zagrzeb - Berlin, natomiast sesja UDP dla relacji Wiedeń - Malmö. Minimalna wymagana przepustowość dla obu sesji TCP została ustaliona na $3 \frac{Mb}{s}$, a dla sesji UDP na $2 \frac{Mb}{s}$. Suma tych trzech wartości nie przekroczyła przepustowości łączy, zatem każde łącze w sieci było w stanie obsłużyć wszystkie trzy sesje jednocześnie. Najkrótsze pod względem opóźnienia ścieżki wyznaczone dla poszczególnych sesji obejmowały dwa łącza współdzielone z dwoma pozostałymi strumieniami (rys. 7). Sesja UDP wykorzystała w takim razie ustalone $2 \frac{Mb}{s}$, natomiast dla sesji TCP pozostała przepustowość oferowana przez łącza została rozdzielona w sposób sprawiedliwy. Odnotowaliśmy przepustowości wynoszące $4,42 \frac{Mb}{s}$ i $4,02 \frac{Mb}{s}$ w przypadku sesji TCP oraz praktycznie bezstratną transmisję o żądanej prędkości przesyłu w przypadku sesji UDP.

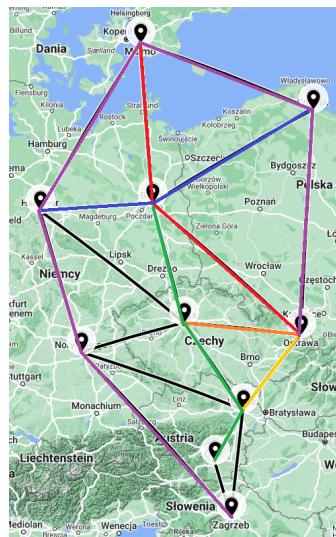


Rys. 7. Obsłużone przez współdzielone łącze sesje TCP i UDP

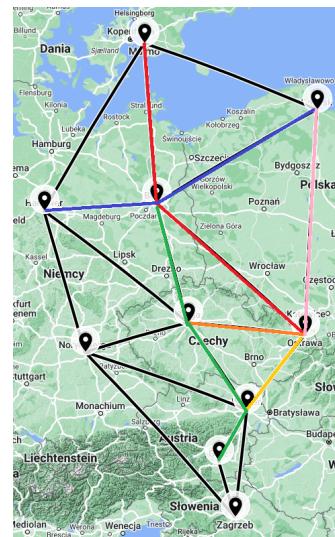
3.2.5. Pozostałe eksperymenty

Pozostałe eksperymenty obejmowały już wyłącznie obserwację działania aplikacji w kontekście doboru optymalnych ścieżek dla poszczególnych sesji. W testach tworzyliśmy, a następnie od razu usuwaliśmy sesje patrząc wyłącznie na wyniki działania programu. W eksperymencie czwartym sprawdzaliśmy, czy klasa **Manager** zablokuje utworzenie wykluczających się sesji między tymi samymi hostami (np. gdy jest aktywna sesja TCP, to powinna zezwolić na uruchomienie sesji UDP, ale nie ping itd.). Obserwowane w tym kontekście działanie programu było prawidłowe.

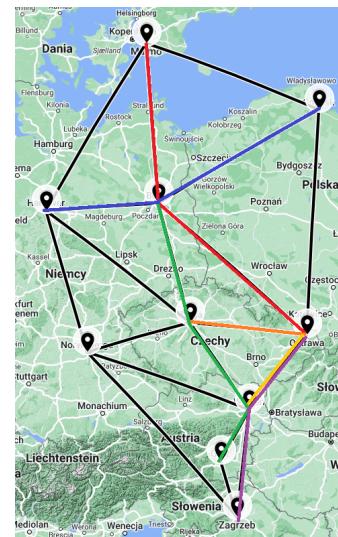
W piątym eksperymencie próbowaliśmy uruchomić takie sesje, żeby ostatnia z nich musiała zostać pokierowana możliwie najdłuższą drogą. Wszystkie utworzone sesje były sesjami UDP o minimalnych wymaganiach $9 \frac{Mb}{s}$ (rys. 8). Następnie zakończyliśmy ostatnią utworzoną sesję (fioletową) i dodaliśmy jeszcze jedną (różową), która uniemożliwiła ponowne pokierowanie fioletowej sesji nawet tą okrężną ścieżką. Ponowna próba uruchomienia fioletowej sesji zakończyła się odmową jej realizacji (rys. 9). W kolejnym kroku usunęliśmy wszystkie sesje i dodaliśmy je ponownie, ale tym razem z minimalną wymaganą przepustowością wynoszącą $2 \frac{Mb}{s}$. Fioletowa sesja, która poprzednio musiała zostać pokierowana okrężną drogą, w tym przypadku bez problemu została zrealizowana najkrótszą pod względem opóźnienia ścieżką (rys. 10).



Rys. 8. Sesja fioletowa pokierowana okrężną ścieżką



Rys. 9. Odmowa realizacji fioletowej sesji



Rys. 10. Fioletowa sesja pokierowana najkrótszą ścieżką

W szóstym eksperymencie, w celu dalszej weryfikacji poprawności działania aplikacji dla wielu jednocześnie działających sesji, tworzyliśmy i usuwaliśmy ścieżki analizując wartości zwarcane przez program. Wartości te nie zostały przedstawione w sprawozdaniu, aby zachować jego zwięzłość, zostały jednak umieszczone w przesłanym archiwum. Podczas realizacji tego eksperymetru aplikacja zwracała wyniki zgodne z naszymi oczekiwaniemi.

3.3. Wnioski

Przeprowadzone eksperymenty z wykorzystaniem przygotowanej aplikacji sterującej siecią wykazały poprawność jej działania. Unaoczniliśmy także, jak ważna w sterowaniu ruchem w sieci jest automatyzacja całego procesu. Ręczne konfigurowanie odpowiednich (pod kątem jakości transmisji) dróg przesyłu danych byłoby bardzo zmuśniające i skomplikowane. Dużym wyzwaniem jest jednak dobieranie algorytmów wyznaczających optymalne ścieżki dla poszczególnych sesji. Należy zatem starannie dostosowywać te algorytmy do własnych potrzeb, a także zachować odpowiednie proporcje pomiędzy wkładem procesów automatycznych w sterowanie ruchem, a kontrolą tych procesów przez człowieka.

4. Podsumowanie

Podsumowując, wykonanie ćwiczenia laboratoryjnego było cennym doświadczeniem. Do wiedzy na temat płaszczyzny danych sieci, nabycie w poprzednim ćwiczeniu, dołożyliśmy wiedzę na temat sterowania siecią. Podobnie jak w poprzednim ćwiczeniu, mimo pracy jedynie z emulatorem i bardzo małą siecią, udało nam się wyciągnąć wnioski na temat płaszczyzny sterowania także w rzeczywistych sieciach teleinformatycznych.

Informacja

Wraz ze sprawozdaniem przesłane zostało archiwum w formacie **zip**, w którym znajdują się: kod aplikacji, skrypty oraz polecenia przygotowane i wykorzystane podczas realizacji ćwiczenia, a także logi uzyskane w wyniku przeprowadzonych testów.

5. Bibliografia

- [1] Google. *Google Maps - My Maps*. URL: <https://www.mymaps.google.com> (visited on 11/18/2023).
- [2] Zee Source. *Zee Maps - Create and publish interactive maps*. URL: <https://www.zeemaps.com/> (visited on 11/18/2023).