SCHT

Laboratorium 2

Płaszczyzna sterowania sieci

Jakub Grzechnik **325278**

Jakub Strzelczyk **325325**

19 listopada 2023

Spis treści

Wydział Elektroniki i Technik Informacyjnych
POLITECHNIKA WARSZAWSKA

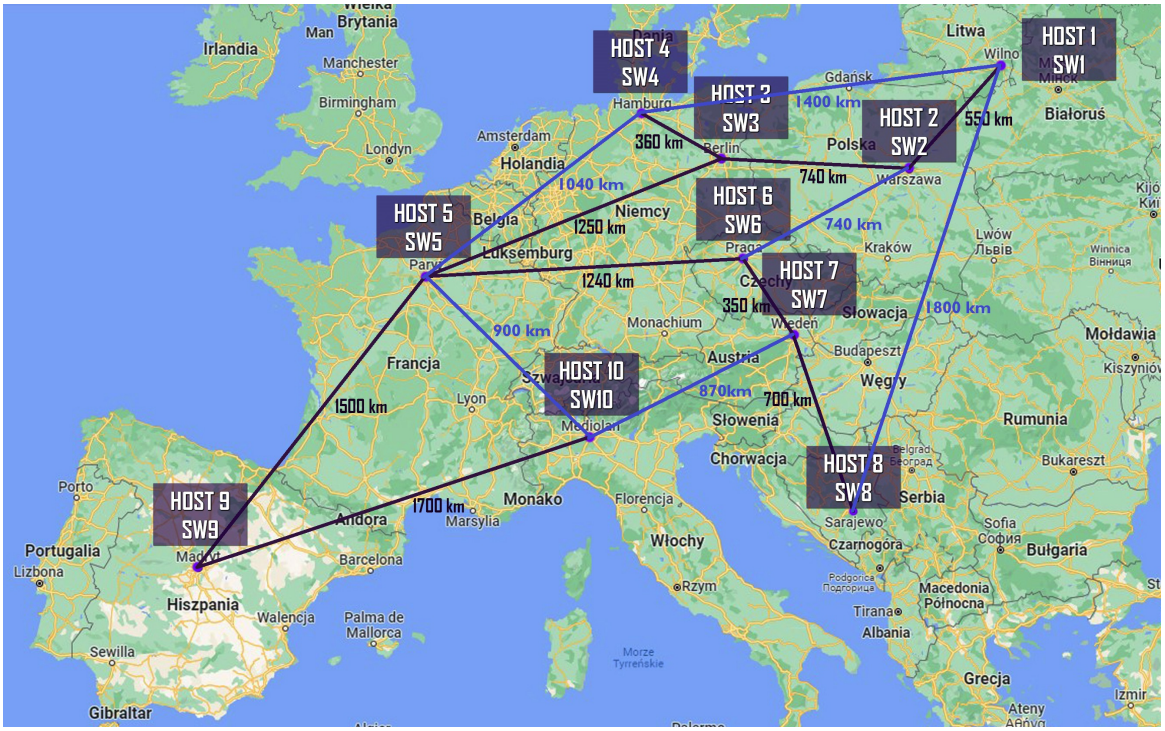
1. Cel laboratorium

Celem ćwiczenia jest zapoznanie się z możliwością sterowania siecią przy użyciu kontrolera ONOS, oraz możliwościami tworzenia aplikacji sterujących siecią i wpływania przy ich użyciu na charakterystyki ruchu przesyłanego przez sieć.

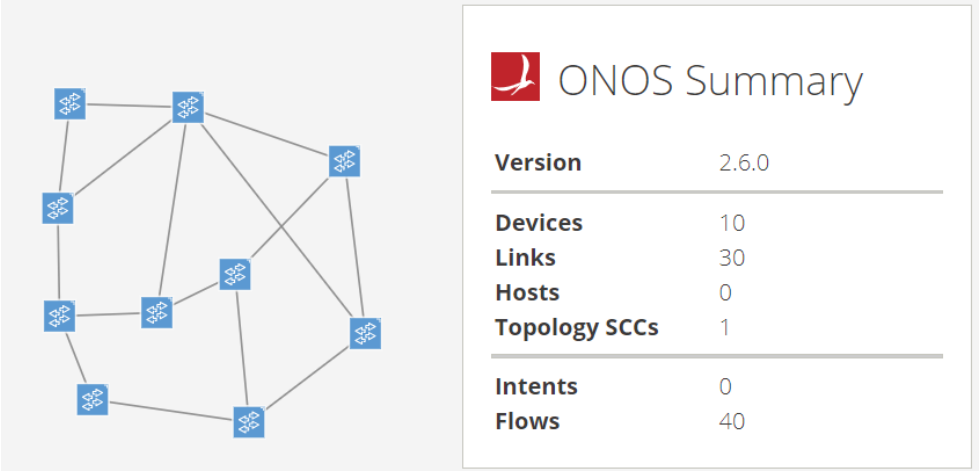
2. Zadanie 1

2.1. Przygotowanie

Na początku rozszerzyliśmy naszą sieć z poprzedniego ćwiczenia o nowe połączenia, tak aby graf sieci nie miał struktury drzewa. Zaaktualizowaliśmy także kod z topologią w Pythonie używany do uruchomienia topologii w Mininecie. W nowych połączeniach uwzględniliśmy opóźnienia wynikające z prędkości propagacji sygnału światłowodowego oraz odległości między miastami.



Rysunek 1. Mapa emulowanej sieci



Rysunek 2. GUI kontrolera Onos

Uruchomiliśmy kontroler sieci ONOS i emulator Mininet ze zdefiniowaną siecią. Możemy obserwować konkretne połączenia po zalogowaniu się do graficznego interfejsu kontrolera Onos co przedstawia Rysunek 2 przedstawiony powyżej. Następnie przygotowaliśmy pliki konfiguracyjne kontrolera Onos, tak abyśmy mogli zrealizować nasze połączenia z poprzedniego ćwiczenia (h2-h1 oraz h2-h7). Będą one realizowane tymi samymi drogami co poprzednio, żeby precyzyjnie porównać otrzymane wyniki. W opisanych plikach zdefiniowaliśmy kryteria przepływów, obejmujące obejmujące m.in. priorytet (priority), timeout, stałość przepływu (isPermanent), identyfikator urządzenia źródłowego (deviceID), port wyjściowy (port), typ ramki Ethernet (ETH_TYPE) oraz adres IPv4 hosta docelowego (ip). Poniżej przedstawiamy fragment ręcznie przygotowanego pliku w formacie .json zgodnie z dokumentacją styku REST kontrolera ONOS.

Listing 1. Fragment pliku Onos21.json

```
{"flows": [{
. . .
 "priority": 40000,
      "timeout": 0,
      "isPermanent": true,
      "deviceId": "of:0000000000000002",
      "treatment": {
        "instructions": [
          {
            "type": "OUTPUT",
            "port": "2"
          }
        ]
      },
      "selector": {
        "criteria": [
          {
            "type": "ETH_TYPE",
            "ethType": "0x0800"
          },
          {
            "type": "IPV4_DST",
            "ip": "10.0.0.1/32"
          }
        ]
      }
    },
. . .
]}
```

Następnie zdefiniowaliśmy i uruchomiliśmy skrypt *.bat z sekwencjami wywołań narzędzia curl konfigurującymi węzły sieci potrzebne do naszych eksperymentów przy użyciu operacji styku REST kontrolera ONOS.

Listing 2. Plik Onos2.bat

```
curl --user onos:rocks -X POST "http://192.168.0.80:8181/onos/v1/flows"
-d @Onos21.json -H "Content-Type: application/json" -H "Accept: application/json"

curl --user onos:rocks -X POST "http://192.168.0.80:8181/onos/v1/flows"
-d @Onos27.json -H "Content-Type: application/json" -H "Accept: application/json"
pause
```

Skorzystaliśmy z wcześniej opracowanych skryptów uruchamiających narzędzie iperf i zrealizowaliśmy wybrane sesje transportowe. Postępowaliśmy krok po kroku tak jak w zadaniu 4. z laboratorium numer 1. Zaczęliśmy od sesji UDP pomiędzy hostami h2 (klient) a h1 (serwer).

Porównując wyniki w zadaniach będziemy stosować oznaczenie, gdzie wynik z poprzedniego laboratorium będzie znajdował się zawsze w nawiasie za aktualnym wynikiem, np. 1.01Mbit/s (1.10Mbit/s).

2.2. Sesja UDP

Najpierw przeprowadziliśmy eksperyment weryfikujący połączenie między hostami h1 i h2. Uruchomiliśmy aplikację generatora ruchu iperf, h1 działał jako serwer, a h2 jako klient. Łącze pomiędzy tymi hostami miało przepustowość 10 Mbps, natomiast pomiędzy hostami, a serwerami 5 Mbps. Pierwszy test został wykonany przy użyciu parametrów domyślnych. Nastąpił przesył danych z średnią przepływnością 1,05 MB/s, co jest wartością identyczną do tej z poprzednich laboratoriów. W kolejnym kroku modyfikowaliśmy parametry sesji UDP i badaliśmy różne scenariusze.

TEST 1 Zmieniliśmy parametry na przedstawione poniżej:

```
h1 iperf -e -u -i 1 -s -p 5001 >/home/mininet/UDPserver.txt&
h2 iperf -e -i 1 -c h1 -p 5001 -u -S 0x10 -t 10 -b 300pps -l 1470 -w 1M
>/home/mininet/UDPclient.txt&
```

Klient przesyłał dane z szybkością 3.52Mbit/s. Nie utracono pakietów, połączenie było stabilne, wartość jittera wyniosła średnio 0.022ms. Średni czas przesyłu pakietu wyszedł zbliżony do wyniku z laboratorium 1 - **3.975ms (4.287ms)**.

TEST 2 Zmieniliśmy parametr -b dotyczący szybkości przesyłania na 420pps. Średnia prędkość wysyłania danych przez klienta wynosiła 4.95, parametr latency miał średnią wartość **102ms (91ms)**.

TEST 3 Przy ustawieniu parametrów -b na 500 i -l na 1470 wystąpiły straty pakietów na poziomie około **17 (poprzednio 15)**%. Szybkość przesyłu danych po stronie wyniosła 5.88 Mbit/s, przewyższając przepustowość łącza (5 Mbit/s). To spowodowało buforowanie i przekierowywanie pakietów, prowadząc do wzrostu strat i znacznych opóźnień (Latency avg: **246.843ms (236.8ms)**. Wyższa prędkość przesyłu danych przekraczająca dostępną przepustowość spowodowała przeciążenie sieci.

TEST 4 Zwiększenie maksymalnego rozmiaru kolejek w Pythonie przy tej prędkości spowodowało wydłużenie czasu przesyłu, ale zmniejszenie strat. Zwiększając rozmiar buforów, zmniejszamy ryzyko odrzucenia danych. Skutki zmiany tych parametrów były takie same jak w przypadku poprzedniego ćwiczenia.

TEST 5 Przesłaliśmy tę samą ilość danych mniejszymi pakietami (parametry: "l" - 100, "b" - 4000). Klient przesyłał dane z szybkością 3.2Mbit/s. Utracono taki sam procent pakietów co poprzednio (0.1%). Średni czas przesyłu pakietu wyniósł **5.805ms (6.306ms)**.

TEST 6 Ustawiliśmy przepływność na 6000 pakietów na sekundę (pps). Pomimo nieco niższej szybkości przesyłu danych niż przepustowość łącza, odnotowaliśmy znaczne straty pakietów - około **27% (poprzednio 27%)**.

Nie opisujemy jak parametry przesyłu danych zależą od typu, parametrów generowanego strumienia danych, wykonaliśmy to podczas laboratorium 1. Powyżej przedstawiliśmy jedynie porównanie wartości uzyskiwanych parametrów. Potwierdzamy zdolności realizacji sesji UDP między hostami. Jeżeli chodzi o UDP to wyniki są bardzo zbliżone do tych z ćwiczenia nr 1. To potwierdza, że oba eksperymenty opierały się na tym samym sposobie przesyłu danych.

2.3. Sesja TCP

Uruchomiliśmy sesję TCP między dwoma hostami, h2 (klient) a h7 (serwer). Wcześniej skonfigurowaliśmy przepustowość wszystkich połączeń przekazujących dane z h2 do h7 na poziomie 10 Mbit/s. Przy użyciu generatora ruchu przesłaliśmy 15 MB danych, modyfikując jednocześnie wartości rozmiaru okna i bufora.

```
h7 iperf -e -i 1 -s -p 5001 >/home/mininet/TCPserver.txt&
h2 iperf -e -i 1 -c h7 -p 5001 -N -S 0x08 -n 15M >/home/mininet/TCPclient.txt
```

Rozpoczęliśmy od weryfikacji poprawności działania narzędzia dla standardowego przypadku opisanego wcześniej. Klient uzyskał średnią przepustowość na poziomie **9.78Mbit/s (9.51Mbit/s)**. W tym konkretnym przypadku, domyślnie ustawiona wartość okna wynosiła 85 kB.

TEST 1 Tak jak w laboratorium nr 1 zmienialiśmy wartości rozmiaru bufora. Dla kolejnych wartości rozmiaru bloku danych otrzymywaliśmy średnie przepustowości przesyłu danych od klienta do serwera.

Rozmiar bloku danych: **1** — Przepustowość: **3.62Mbit/s (3.61Mbit/s)**
Rozmiar bloku danych: **2** — Przepustowość: **7.05Mbit/s (7.19Mbit/s)**
Rozmiar bloku danych: **3** — Przepustowość: **9.67Mbit/s (9.76Mbit/s)**
Rozmiar bloku danych: **4** — Przepustowość: **10.01Mbit/s (9.97Mbit/s)**
Rozmiar bloku danych: **5** — Przepustowość: **10.76Mbit/s (10.7Mbit/s)**
Rozmiar bloku danych: **6** — Przepustowość: **10.64Mbit/s (10.6Mbit/s)**
Rozmiar bloku danych: **7** — Przepustowość: **10.6Mbit/s (10.6Mbit/s)**
Rozmiar bloku danych: **8** — Przepustowość: **10.52Mbit/s (10.5Mbit/s)**

W tym przypadku przy parametrze "-l" równym 4, przepustowość również ustabilizowała się, nie rosnąc dalej wraz ze zwiększaniem rozmiaru bufora, tak jak poprzednio. Następnie zwiększyliśmy przepustowość łącza do 20 Mbit/s. Na przykład, dla parametru -l ustawionego na 7, średnia przepustowość połączenia wyniosła **20.18Mbit/s (20.23 Mbit/s)**. Tak samo jak w ćwiczeniu 1. początkowo czynnikiem wpływającym na przepustowość jest wielkość bufora, jednak po przekroczeniu określonej wartości, ograniczeniem staje się przepustowość łącza.

TEST 2 Parametr -l przywrócono do ustawienia domyślnego. Analizie poddaliśmy wpływ rozmiaru okna na przepustowość łącza i czas RTT. Ze względu na ilość wykonanych doświadczeń, porównaliśmy jedynie kilka testów z laboratorium 1.
Rozmiar okna: 10kB — RTT: **139ms (142ms)** — Przepustowość: **6.57Mbit/s (6.52Mbit/s)**
Rozmiar okna: 40kB — RTT: **118ms (122ms)** — Przepustowość: **7.96Mbit/s (7.93Mbit/s)**
Rozmiar okna: 70kB — RTT: **103ms (107ms)** — Przepustowość: **9.07Mbit/s (9.02Mbit/s)**
Rozmiar okna: 120kB — RTT: **125ms (125ms)** — Przepustowość: **9.56Mbit/s (9.56Mbit/s)**
Rozmiar okna: 200kB — RTT: **192ms (190ms)** — Przepustowość: **6.60Mbit/s (9.62Mbit/s)**

Tak jak w ćwiczeniu pierwszym zmniejszenie rozmiaru okna poniżej optymalnej wartości ogranicza przepustowość, a zwiększenie ponad nią wydłuża czas RTT, zachowując przepustowość łącza.

2.4. Wnioski

Otrzymaliśmy bardzo podobne wyniki do tych z laboratorium 1 - dla wszystkich sprawdzanych parametrów, co potwierdza prawidłową konfigurację ścieżek za pomocą interfejsu REST. W obu przypadkach dane zostały przesłane przez te same węzły i te same łącza o wartościach parametrów identycznych z tymi z poprzedniego laboratorium. Ostatecznie, potwierdzamy możliwość realizacji sesji transportowych zdefiniowanych w ramach punktu 4 poprzedniego ćwiczenia, a także stwierdzamy, że wartości uzyskiwanych parametrów przesyłu danych były bardzo zbliżone.

3. Zadanie 2

3.1. Przygotowanie

W celu skonfigurowania węzłów sieci, by uzyskać poprawę wartości parametrów przesyłu danych przy realizacji zbiorów jednoczesnych sesji transportowych rozpatrywanych w poprzednim ćwiczeniu, stworzyliśmy nowe pliki konfiguracyjne kontrolera Onos w formacie .json. Przepustowość łączy między wszystkimi hostami w sieci wynosiła 10 Mbit/s.

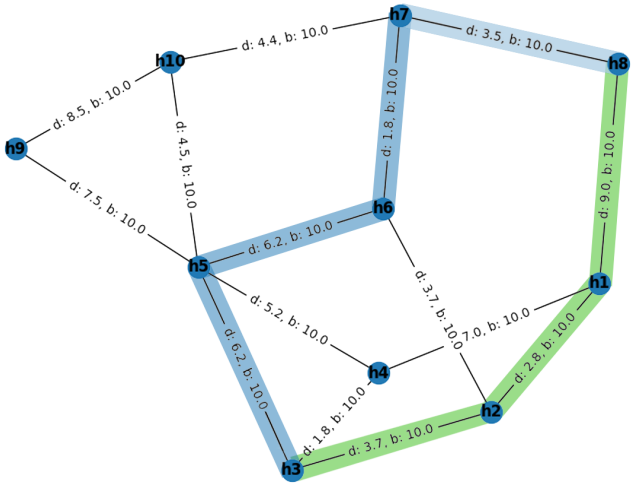
3.2. Dwie sesje UDP

Uruchomiliśmy dwie sesje UDP między tymi samymi hostami co w poprzednim laboratorium, lecz teraz wzięliśmy pod uwagę nowe drogi przesyłu danych, które dobraliśmy w potencjalnie lepszy sposób.

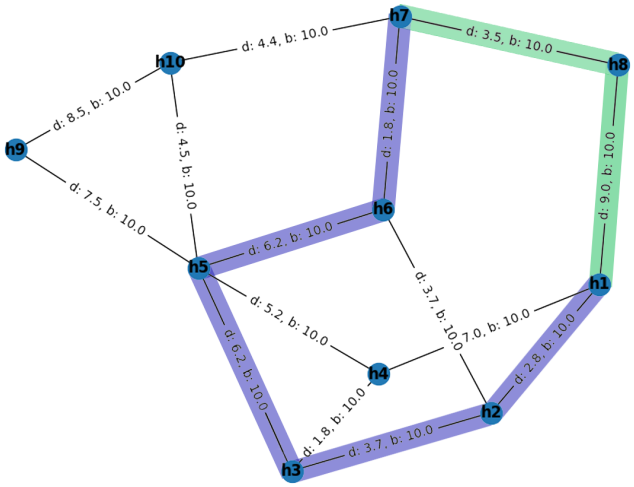
```
h3 iperf -e -u -i 1 -s -p 5001 >/home/mininet/UDP2server1.txt&
h8 iperf -e -i 1 -c h3 -p 5001 -u -S 0x10 -t 20 -b 420pps -l 1470 -w 1M
>/home/mininet/UDP2client1.txt&

h1 iperf -e -u -i 1 -s -p 5001 >/home/mininet/UDP2server2.txt&
h7 iperf -e -i 1 -c h1 -p 5001 -u -S 0x10 -t 20 -b 420pps -l 1470 -w 1M
>/home/mininet/UDP2client2.txt&
```

Wcześniej przesył danych między hostami h8 a h3 odbywał się według trasy: **h8 - s8 - s7 - s6 - s5 - s3 - h3**. Łączny delay na tej trasie wynosił 17.9ms. Teraz zdecydowaliśmy się na trasę: **h8 - s8 - s1 - s2 - s3 - h3**, gdzie łączne opóźnienie wynosi 15.7 ms, czyli jest nieco mniejsze. Dodatkowo przemieszczając się tą trasą, dane muszą przejść przez mniejszą liczbę switchy, aby dotrzeć do docelowego hosta.

W przypadku trasy między hostami h1 a h7 pierwotnie przebiegała ona przez: **h7 - s7 - s6 - s5 - s3 - s2 - s1 - h1** (łączne opóźnienie - 20.9 ms). Nowa trasa obejmuje: **h7 - s7 - s8 - s1 - h1**, gdzie łączny delay na łączach wynosi 12.7 ms. Według nas są to scieżki, dla których można uzyskać poprawę wartości przesyłu danych.



Rysunek 3. Drogi przesyłu dla połączenia h8 - h3



Rysunek 4. Drogi przesyłu dla połączenia h7 - h1

TEST 1 Dla parametru -b równego 420pps oraz -l 1470 bajtów obaj klienci przesyłali dane z przepływnością **4.94Mbit/s**. Straty pakietów wynosiły około **0.6% (0.5%)**, średnia wartość latency dla połączenia h8-h3 **170ms (184ms)**, dla połączenia h7-h1 **166ms (179ms)**. Następnie zwiększyliśmy wartość parametru -b dla obu sesji na 470pps. W przypadku pierwszego połaczenia straty pakietów wzrosły do około **24% (15%)**, a w przypadku drugiego zmalały do **6.6% (8.8%)**. Wartość latency dla połączenia h8-h3 wyniosła **199ms (206ms)**, dla połączenia h7-h1 **191ms (202ms)**. Patrząc na logi po stronie serwera odczytaliśmy przepustowość, która wynosiła odpowiednio **4.19Mbit/s (4.67Mbit/s)** oraz **5.13Mbit/s (5.00Mbit/s)** W porównaniu do pierwszego laboratorium zmniejszyły się średnie wartości opóźnienia.

TEST 2 Przesyłamy taką samą ilosć pakietów lecz o różnych rozmiarach. Parametr -b został ustawiony na 700pps, a rozmiary pakietów wynosiły odpowiednio 1470B oraz 100B. Dla połączenia wysyłającego większe pakiety straty były znacznie większe niż w przypadku przesyłania mniejszych pakietów. Wynosiły **73% (70%)**. Zatem w tym przypadku dla większych pakietów straty były nieznacznie mniejsze niż poprzednio.

TEST 3 W kolejnym teście dla h1-h7 ustawiliśmy -b na 1000pps, a dla h3-h8 na 100pps. Host h1 wysyłał dane z przepływnością **11.8Mbit/s**, a h3 **1.18Mbit/s**. Przy pierwszym uruchomieniu straty pakietów wynosiły odpowiednio **50% (25%)** oraz **19% (8.8%)**. Kolejne uruchomienie skryptu spowodowało następujące straty **61% (65%)** oraz **20% (27%)**.

Zatem widzimy na przykładzie połączenia UDP, iż pomimo, że chcieliśmy wybrać potencjalnie lepsze trasy, nie zawsze otrzymujemy lepsze parametry przesyłu danych, ponieważ zależą one także od aktualnego obiążenia sieci. W większości otrzymywaliśmy lepsze wyniki, natomiast np. podczas testu 3. otrzymaliśmy najpierw większe straty pakietów niż te dla testu z poprzedniego laboratorium. Aby dobrze dobrać takie trasy, nie tylko na podstawie samej wartości zsumowanych opóźnień na łączach, moglibyśmy napisać odpowiedni algorytm, który bierze pod uwagę aktualne obciążenie łączy.

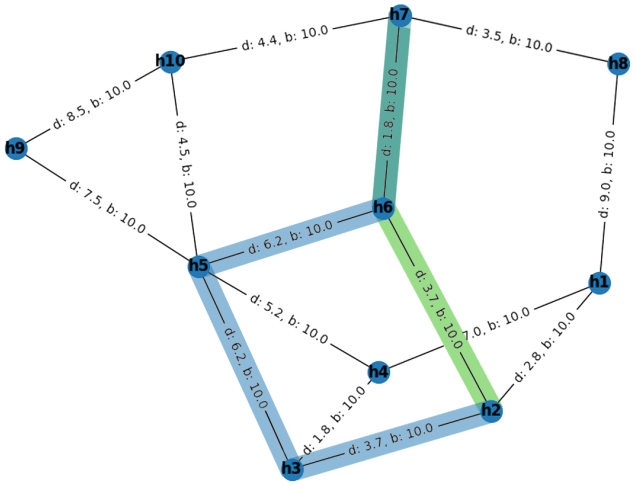
3.3. Dwie sesje TCP

Następnie uruchomiliśmy dwie sesje TCP tak jak w poprzednim laboratorium.

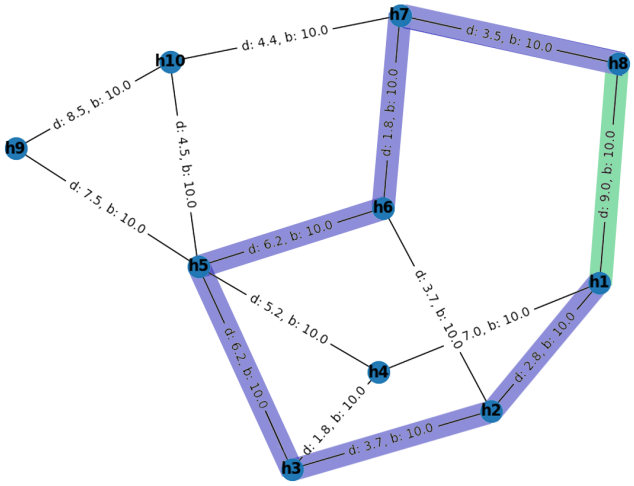
```
h2 iperf -e -i 1 -s -p 5001 >/home/mininet/TCP2server1.txt&
h7 iperf -e -i 1 -c h2 -p 5001 -N -S 0x10 -n 15M >/home/mininet/TCP2client1.txt&

h8 iperf -e -i 1 -s -p 5001 >/home/mininet/TCP2server2.txt&
h1 iperf -e -i 1 -c h8 -p 5001 -N -S 0x10 -n 15M >/home/mininet/TCP2client2.txt
```

Wcześniej przesył danych między hostami h7 a h2 odbywał się według trasy: **h7 - s7 - s6 - s5 - s3 - s2 - h2**. Łączny delay na tej trasie wynosił 18.1ms. Teraz zdecydowaliśmy się na trasę: **h7 - s7 - s6 - s2 - h2**, gdzie łączne opóźnienie wynosi 5.7 ms, czyli jest znacznie mniejsze. Ponadto, liczba switchy, przez które dane muszą przejść, aby dotrzeć do końcowego hosta, została zredukowana.

W przypadku trasy między hostami h1 a h8 pierwotnie przebiegała ona przez: **h1 - s1 - s2 - s3 -s5- s6 -s7 - s8 -h8** (łączne opóźnienie - 24.4 ms). Nowa trasa obejmuje: **h1 - s1 - s8 - h8**, gdzie łączny delay na łączach wynosi 9.2 ms. Uważamy, że dzięki tym ścieżkom będziemy mogli poprawić parametry przesyłu danych, gdyż łączny czas opóźnień na łączach jest mniejszy. Nasze nowo zdefiniowane ścieżki (zaznaczone na zielono) są "rozłączne", czyli dane przesyłane za pomocą oddzielnych połączeń nie przechodzą przez te same łącza.



Rysunek 5. Drogi przesyłu dla połączenia h2 - h7



Rysunek 6. Drogi przesyłu dla połączenia h8 - h1

TEST 1 Przesyłaliśmy dane między hostami o wielkości 15MB, przy użyciu domyślnego okna. W poprzednim ćwiczeniu TCP rozdzielił dostępną przepustowość po równo między dwie sesje tzn. dostępna przepustowość na łączach wynosiła 10Mbit/s i ponieważ dane od hosta h1 do h8 i od h2 do h7 były przesyłane w większości przez te same łącza to całkowity transfer danych w przypadku obu połączeń wynosił po 15 MB przy średniej przepustowości zbliżonej do 5 Mbps. Teraz przy nowych trasach pakiety przemieszczają się zupełnie innymi łączami. Mogą wykorzystywać w pełni przepustowość łącza. Zarówno dla połączenia pomiędzy h7 a h2 oraz h1 a h7 średnia przepustowość połączenia wyniosła 10.6Mbps.

TEST 2 Następnie rozpoczęliśmy uruchamianie sesji klientów dla kilku równoczesnych połączeń, używając parametru -P. Po zestawieniu kilku równoczesnych połączeń między h7 a h2 zauważyliśmy, że wyniki były zbliżone do tych z TESTU 1. Dostępna przepustowość została praktycznie równo podzielona między te kilka połączeń. Innymi słowy, przy jednoczesnym działaniu kilku połączeń, każde z nich otrzymywało niemal identyczną część dostępnej przepustowości. Wyniki były takie same jak w poprzednim laboratorium.

TEST 3 Poprzednio uruchomiliśmy jednoczesne sesje między h1-h3 oraz h5-h2, lecz wcześniej zmieniliśmy wartość przepustowości łączy pomiędzy switchem 5 a switchem 3 na 3Mbit/s. Przepsutowości połączeń zostały podzielone następująco: h1-h3: **7Mbit/s**, h5-h2: **3Mbit/s**. Pakiety pokonując drogę od h1 do h3 przechodziły przez switche **s1 - s2 - s3**, a od h5 do h2 przez **s5 - s3 - s2**. My postanowiliśmy zbadać parametry przesyłu danych, gdy zmienimy drugą trasę na taką, która przechodzi przez switche **s5 - s6 - s2**. Wówczas pakiety nie przechodzą przez łącze o przepustowości 3Mbit/s a także jedna ścieżka nie zawiera się w drugiej. Dla połączenia między h1 a h3 klient przesyłał dane z średnią przepływnością 10.5Mbps, a dla h2 - h5: 10.6Mbps. Pozwoliło nam to w efektywniejszy sposób przesyłać dane korzystając z lepszej przepustowości.

TEST 4 Zestawiliśmy dwa połączenia między h2 a h7 charakteryzujące się innym rozmiarem okien. W pierwszym połączeniu wynosił 50kB, a w drugim 100kB. Aktualny wynik tak samo jak w laboratorium pierwszym pokazał, że dla drugiego połączenia o bardziej optymalnym rozmiarze okna równym 100kB przepustowość była większa.

Zdołaliśmy dostosować konfigurację węzłów sieci, korzystając z interfejsu REST kontrolera ONOS, w taki sposób, który pozwolił na poprawę parametrów przesyłu danych podczas obsługi równoczesnych sesji transportowych TCP.

3.4. Sesja UDP i TCP

Zestawiliśmy jednocześnie połączenia TCP i UDP między hostami h2 a h4. W poprzedniej topologii dane były przesyłane ścieżką **h2 - s2 - s3 - s4 - h4**. My umożliwimy przesył dwoma ścieżkami, tą z poprzedniej topologii oraz drugą przechodzącą przez switche **s2 - s1 - s4**. Dla powyższych połączeń skonfigurowaliśmy odpowiednie pliki w formacie .json, zawierające wszystkie niezbędne ustawienia umożliwiające przesył danych przez dwie trasy. Następnie przesłaliśmy ten plik konfiguracyjny do kontrolera ONOS przy użyciu żądania POST protokołu HTTP.

```
h4 iperf -e -u -i 1 -s -p 5001 >/home/mininet/UDPserver.txt&
h2 iperf -e -i 1 -c h4 -p 5001 -u -S 0x10 >/home/mininet/UDPclient.txt&

h4 iperf -e -i 1 -s -p 5001 >/home/mininet/TCPserver.txt&
h2 iperf -e -i 1 -c h4 -p 5001 -N -S 0x10 -n 20M >/home/mininet/TCPclient.txt
```

TEST 1 Podczas pierwszego testu TCP osiągnęło przepustowość **10Mbit/s (9Mbit/s)**, a UDP około **1.25Mbit/s (1Mbit/s)**. Przepustowość nieco wzrosła, gdyż dane były przesyłane przez inne łącza, a w poprzednim przypadku transfer danych odbywał się poprzez jedną trasę.

TEST 2 Następnie zdecydowaliśmy się zwiększyć parametr -b w przypadku połączenia UDP do 2000pps. Przepustowość TCP pozostała na poziomie **10 Mbit/s (5 Mbit/s)**, ponieważ nie wprowadziliśmy żadnych zmian w parametrach, natomiast przepustowość UDP wzrosła do **9,6 Mbit/s (9,53 Mbit/s)**. Wartość przepustowości TCP równa 5 Mbit/s z poprzedniego pomiaru wynikała z równoczesnego korzystania z tych samych łącz przez połączenie UDP. W przypadku protokołu TCP konieczne było dostosowanie przepływności, ponieważ jest to protokół gwarantujący niezawodność tranasmisji danych. Ze względu na przeciążenie łączy dla UDP straciliśmy **58% (59%)** pakietów. To oznacza, że h2 chciał przesyłać zbyt dużą ilość danych biorąc pod uwagę przepustowości łączy, które wynoszą 10Mbit/s (2000pps*1470B = 23.5Mbps > 10Mbps).

TEST 3 Po zmniejszeniu rozmiaru pakietów dla połączenia UDP na 100B nasze przespustowości wyniosły: dla TCP **10-Mbit/s (7.10Mbit/s)**, dla UDP **1.55Mbit/s (1.55Mbit/s)**, a straty pakietów były minimalne. Dla TCP nie zmienialismy parametrów, więc parametru przesyłu danych były identyczne jak w teście 1. i 2., natomiast dla UDP zmieniła się wartość przepływności - przesyłano mniejszą ilość danych w jednostce czasu.

TEST 4 Następnie na koniec zbadaliśmy wpływ rozmiaru okna TCP na przepustowość.
Rozmiar okna: 10kB — Przepustowość: **9.1Mbit/s (6.42Mbit/s)**
Rozmiar okna: 100kB — Przepustowość: **9.65Mbit/s (7.41Mbit/s)**
Rozmiar okna: 300kB — Przepustowość: **9.78Mbit/s (7Mbit/s)**

Dobraliśmy takie ścieżki które prowadzą przez inne łącza, zatem zmiana parametrów połączenia UDP teraz już nie ma wpływu na zmianę przepustowości TCP. Poprzednio TCP dostosowało się, aby uniknąć utraty pakietów, gdyż ten protokół gwarantuje niezawodną transmisję danych. W tym przypadku otrzymaliśmy lepsze parametry przesyłu danych dla tych samych parametrów połączenia. Ponadto możemy jeszcze raz potwierdzić, że rozmiar okna w protokole TCP ma wpływ na jakość usług (QoS) w przypadku nawiązanych połączeń.

3.5. Wnioski

Skonfigurowaliśmy węzły w sieci za pomocą kontrolera ONOS poprzez interfejs REST w celu optymalizacji parametrów przesyłu danych podczas obsługi równoczesnych sesji transportowych. Udało się poprawić wartości przepustowości i opóźnień w dwóch sesjach UDP. W przypadku sesji TCP, konfiguracja także przyczyniła się do wzrostu przepustowości. Ostatecznie, wyniki eksperymentów podkreślają znaczenie odpowiedniej konfiguracji sieci w zależności od zmieniających się warunków.

4. Zadanie 3

4.1. Wprowadzanie informacji o topologii

Napisaliśmy program w Pythonie sterujący siecią, który za pośrednictwem kontrolera ONOS dostosowuje zasoby emulowanej sieci przy użyciu emulatora sieci Mininet. Najpierw opisaliśmy naszą topologię w pliku tekstowym. W pierwszej linii zdefiniowaliśmy wszystkie hosty, a następnie w każdych kolejnych wierszach definiowaliśmy dane według schematu: [**host 1, host 2, delay, bandwidth, port 1, port 2**], gdzie delay, bandwidth oznaczającą odpowiednio opóźnienie i przepustowość na łączu pomiędzy hostem 1 i hostem 2. Port 1 oznacza port, którym host 1 wysyła dane do hosta 2 (analogicznie dla port 2).

Listing 3. topology.txt

```
h1,h2,h3,h4,h5,h6,h7,h8,h9,h10
h1,h2,9,2,2,2
h1,h4,7,10,3,4
h1,h8,9,5,4,3
h2,h3,3.7,10,3,2
h2,h6,1,10,4,4
h3,h4,1.8,10,3,2
h3,h5,6.2,10,4,2
h4,h5,5.2,10,3,6
h5,h6,6.2,10,3,2
h5,h9,7.5,10,4,2
h5,h10,4.5,10,5,3
h6,h7,2.5,10,3,2
h7,h8,3.5,10,3,2
h7,h10,4.4,10,4,4
h9,h10,8.5,10,3,2
```

4.2. Funkcjonalność aplikacji sterującej siecią

Podzieliliśmy kod aplikacji na kilka funkcji w celu poprawy czytelności oraz ułatwienia zarządzania kodem, każda funkcja spełniała określone zadanie w naszym programie:

– **create_graph(my_file)**
Jako parametr przyjmuje ona ścieżkę do pliku napisanego w naszym własnym formacie - w tym przypadku "topology.txt". Wykorzystuje bibliotekę networkx do utworzenia grafu reprezentującego topologię sieci. Odczytuje nazwy hostów z pierwszej linii pliku i dodaje je jako węzły do grafu. Przetwarza każdą kolejną linię, aby wyodrębnić informacje o połączeniach między hostami, w tym opóźnienie, przepustowość oraz szczegóły portów. Zapisuje je w formacie słownika, a następnie tworzy połączenie pomiędzy zdefiniowanymi hostami biorąc pod uwagę uwzględnione parametry łączy. Graf jest następnie konstruowany poprzez dodawanie węzłów i połączeń na podstawie dostarczonych informacji. Funkcja także generuje wizualizację topologii sieci przy użyciu matplotlib, rysując graf z etykietami węzłów oraz etykietami krawędzi wskazującymi opóźnienie i przepustowość. Na koniec zwraca graf topologii sieci (topo).

– **find_all_paths(first_host, second_host)**
Znajduje wszystkie ścieżki między dwoma hostami w topologii sieci. Korzystając z funkcji calculate_total_delay(path), która iteruje przez węzły ścieżki, sumując opóźnienia każdej krawędzi, zwraca posortowaną listę ścieżek. Wykorzystuje funkcję nx.all_simple_paths z biblioteki networkx do znalezienia wszystkich ścieżek między dwoma hostami w sieci.

– **find_path(all_paths, flow_rate)**
Funkcja find_path służy do znalezienia optymalnej ścieżki na podstawie dostępnej przepustowości z listy ścieżek między dwoma hostami. Parametr flow_rate określa wielkość strumienia przesyłanego w jednej sekundzie [Mbps]. Funkcja analizuje wszystkie ścieżki między dwoma hostami, uporządkowane według najniższego łącznego opóźnienia. Najpierw poszukuje ścieżki, dla której minimalna przepustowość łącza jest większa niż przepustowość połączenia (flow_rate) i jednocześnie opóźnienie jest możliwie najmniejsze. Jeśli taka ścieżka nie istnieje, wybierana jest ścieżka o największej możliwej przepustowości, a w przypadku równych przepustowości wybierana jest ta o najmniejszym opóźnieniu.

– **change_topology(found_path, band)**
Ta funkcja modyfikuje wartości w grafie sieci w wyniku przepływu połączenia przez znalezioną ścieżkę. Poprzez aktualizację przepustowości na łączach - ich zmniejszanie o wartość przepustowości połączenia zwiększonej o 0.5Mbit/s podanej przez użytkownika, odzwierciedla zmniejszenie dostępnej przepustowości na tej trasie. Jeśli użytkownik poda przepustowość większą niż dostępna na łączach, spowoduje to pojawienie się "ujemnych przepustowości", co sygnalizuje obciążenie łącza.

– **user_input()**
Funkcja umożliwiająca użytkownikowi wprowadzanie danych dotyczących połączeń sieciowych. Iteracyjnie zbiera informacje o hostach oraz wielkości strumienia danych. W przypadku braku podanej wielkości, przyjmuje domyślną wartość 1 Mbps. Użytkownik decyduje, czy kontynuować dodawanie połączeń. Obsługuje błędy, szczególnie związane z nieprawidłowymi danymi liczbowymi dla wielkości strumienia. Po zakończeniu prezentuje wprowadzone połączenia i zwraca słownik z danymi połączeń.

– **create_json(min_path, i)**
Funkcja create_json tworzy plik JSON konfigurujący węzły w sieci na podstawie zadanej ścieżki minimalnej. Podczas tego procesu, zamienia w pliku JSON wartości takie jak %1, %2, itp., na konkretne wartości związane z pierwszym i ostatnim węzłem na ścieżce minimalnej oraz portami, przez które połączone są kolejne węzły na tej ścieżce. Następnie, zapisuje uzyskany JSON do pliku.

– **post_json_files()**
Korzysta z biblioteki requests do obsługi zapytań HTTP. Wykorzystuje zapytanie POST, aby wysłać wygenerowane za pomocą funkcji create_json(min_path, i) pliki do kontrolera ONOS, który zarządza siecią. Nagłówki Content-Type i Accept są ustawione na "application/json" w celu obsługi danych w formacie JSON podczas komunikacji. Gdy pliki zostaną poprawnie przesłane, użytkownik otrzymuje informację o poprawnej transmisji.

W pliku zip umieściliśmy kod aplikacji wraz z adekwatną dokumentacją opisującą każdą funkcję za pomocą formatu docstring. Dodatkowo dołączyliśmy plik z definicją topologii oraz wszystkie inne niezbędne pliki, konieczne do poprawnego funkcjonowania aplikacji.

4.3. Przykład działania

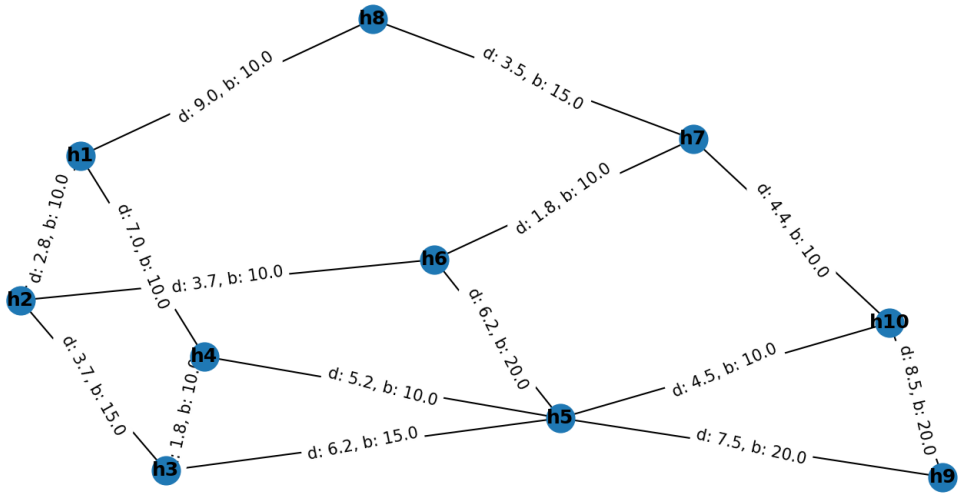
Przedstawiamy widok z terminala środowiska programistycznego Pycharm przedstawiający przykładowy wynik działania naszej aplikacji.

```
Podaj pierwszego hosta: h1
Podaj drugiego hosta: h7
Podaj wielkość strumienia danych [Mbit/s]: 5
Czy zakończyć wprowadzanie połączeń ['y', 'n']: n
Podaj pierwszego hosta: h2
Podaj drugiego hosta: h8
Podaj wielkość strumienia danych [Mbit/s]: 6
Czy zakończyć wprowadzanie połączeń ['y', 'n']: y
Wprowadzone połączenia:
1. Host h1 <-> Host h7, Wielkość strumienia danych: 5.0 Mbit/s
2. Host h2 <-> Host h8, Wielkość strumienia danych: 6.0 Mbit/s
Wszystkie pliki pomyślnie przesłane do kontrolera.
```

Rysunek 7. Widok z terminala

Aplikacja pokazuje, który ścieżkę znalazł algorytm, a także informację o pomyślnym przesłaniu plików konfiguracyjnych w formacie json do kontrolera ONOS.

Ponadto dzięki niej możemy wygenerować graf naszej sieci z wykorzystaniem biblioteki networkx, przykładowy prezentujemy poniżej. Na schemacie są przedstawione zarówno wartości parametru delay (d:) oraz przepustowości łączy (b:).



Rysunek 8. Graf sieci

5. Zadanie 4

5.1. Dwie sesje UDP - zoptymalizowane

Zgodnie z instrukcją przeprowadzimy eksperymenty na jednym ze zbiorów jednocześnie realizowanych sesji transportowych zdefiniowanych w ramach punktu 5 poprzedniego ćwiczenia. Do badań wybraliśmy jednoczesne sesje UDP. Porównamy także wartości uzyskane w zadaniu drugim tego laboratorium.

Zaczęliśmy od wprowadzenia informacji o strumieniach danych oraz odpowiednich hostach do naszej aplikacji przedstawionej w zadaniu nr 3. Ta następnie przygotowała odpowiednie pliki konfiguracyjne i przesłała je do kontrolera Onos zarządzającego naszą siecią. W ten sposób ponownie zbadaliśmy połączenia między hostami h3-h8 oraz h1-h7. Tym razem droga została wybrana nie tylko na podstawie najmniejszego opóźnienia, lecz brała również pod uwagę aktualne obciążenia sieci. W naszej sieci przepustowość wszystkich łączy wynosi po 10Mbit/s, a dane chcemy przesyłać z przepływnością 4.94Mbit/s. Poniżej przedstawiamy wynik z terminala naszej aplikacji:

```
Podaj pierwszego hosta: h7
Podaj drugiego hosta: h1
Podaj wielkosc strumienia danych [Mbit/s]: 4.94
Czy zakonczyc wprowadzanie polaczen ['y', 'n']: n
Podaj pierwszego hosta: h8
Podaj drugiego hosta: h3
Podaj wielkosc strumienia danych [Mbit/s]: 4.94
Czy zakonczyc wprowadzanie polaczen ['y', 'n']: y
Wprowadzone polaczenia:
1. Host h7 <-> Host h1, Wielkosc strumienia danych: 4.94 Mbit/s
2. Host h8 <-> Host h3, Wielkosc strumienia danych: 4.94 Mbit/s
Znaleziona sciezka: ['h7', 'h6', 'h2', 'h1']
Znaleziona sciezka: ['h8', 'h1', 'h4', 'h3']
Wszystkie pliki zostaly pomyslnie przeslane do kontrolera.
```

W celu dosyć przejrzystego przedstawienia porównania wyników, będziemy używać następującej notacji: aktualny wynik (wynik z zadania drugiego, wynik z poprzedniego laboratorium z zadania 5.)

TEST 1 Dla połączeń z parametrami -b 420pps, -l 1470 bajtów nastąpił całkowity spadek strat pakietów do **0% (0.6%, 0.5%)**. Ponadto znacząco zmalała średnia wartość parametru latency dla obu połączeń: h8-h3: **18.294ms (170ms, 184ms)**:

```
[ ID] Interval           Transfer     Bandwidth       Jitter   Lost/Total  Latency avg/min/max/stdev PPS  NetPwr
...
[  3] 0.0000-19.9901 sec  11.8 MBytes  4.94 Mbits/sec   0.072 ms    0/ 8400 (0%) 18.294/18.058/32.151/ 0.976 ms  420 pps  33.77
```

Dla połączenia h7-h1 **8.672ms (166ms, 179ms)**:

```
[ ID] Interval           Transfer     Bandwidth        Jitter   Lost/Total  Latency avg/min/max/stdev PPS  NetPwr
[  3] 0.0000-20.0132 sec  11.8 MBytes  4.94 Mbits/sec   1.267 ms    0/ 8400 (0%)  8.762/ 8.560/27.377/ 0.857 ms  419 pps  70.42
```

Następnie zwiększyliśmy wartość parametru -b dla obu sesji na 470pps. Średni czas przesyłania danych oraz straty praktycznie się nie zmieniły - w przypadku pierwszego połączenia wartość parametru loss wyniosła **0% (24%, 15%)**, latecy avg: **18.329ms (199ms, 206ms)**, w przypadku drugiego połączenia - loss: **0% (6.6%, 8.8%)**, latency avg: **8.825ms (191ms, 202ms)**. Patrząc na logi po stronie serwera odczytaliśmy przepustowość, która wynosiła odpowiednio **5.53Mbit.s (4.19Mbit/s, 4.67Mbit/s)** oraz **5.52Mbit.s (5.13Mbit/s, 5.00Mbit/s)**).

Uzyskaliśmy lepsze parametry przesyłu danych, algorytm wybrał takie scieżki, które umożliwiają przesył danych w przypadku pierwszego i drugiego połączenia innymi łączami, w wyniku czego nie dochodzi do przeciążeń na łączach.

TEST 2 Przesyłamy taką samą ilosć pakietów lecz o różnych rozmiarach. Parametr -b został ustawiony na 700pps, a rozmiary pakietów wynosiły odpowiednio 1470B oraz 100B. W tym przypadku odpowiednia przepustowość łączy, z któych korzystają połączenia pozwoliła nam na bezstratny przesył danych.

```
[ ID] Interval           Transfer     Bandwidth        Jitter   Lost/Total  Latency avg/min/max/stdev PPS  NetPwr
[  3] 0.0000-19.9988 sec  19.6 MBytes  8.23 Mbits/sec   0.142 ms    0/14000 (0%) 18.626/18.056/40.360/ 2.091 ms  700 pps  55.25
```

Przypomnijmy, że dla wcześniejszych doświadczeń w przypadku połączenia, gdzie następował przesył większych pakietów straty były znacznie większe niż w przypadku przesyłania mniejszych pakietów. Natomiast wtedy obie scieżki po części korzystały z tych samych łączy.

TEST 3 W kolejnym teście dla h1-h7 ustawiliśmy -b na 1000pps, a dla h3-h8 na 100pps. Host h1 wysyłał dane z przepływnością **11.8Mbit/s**, a h3 **1.18Mbit/s**. Skonfigurowanie węzłów sieci za pomocą aplikacji, korzystając z interfejsu REST kontrolera ONOS, umożliwiło optymalizacji parametrów przesyłu danych, a zwłaszcza ograniczyło straty pakietów. Wynosiły one odpowiednio **11% (50%, 25%)** oraz **0% (19%, 8.8%)**

5.2. Wnioski

W trakcie eksperymentów z dwiema jednoczesnymi sesjami UDP, zoptymalizowanymi przy użyciu kontrolera ONOS oraz naszej aplikacji, zauważyliśmy istotne poprawy w wartościach parametrów przesyłu danych. Aplikacja wzięła pod uwagę, że przez poszczególne łącza przesyłane są dane, więc wskazała nam dla kolejnego połączenia inną ścieżkę, która składała się z nieobciążonych łączy.

6. Podsumowanie

Dzięki wykonanemu ćwiczeniu zapoznanaliśmy się z możliwością sterowania siecią przy użyciu kontrolera ONOS, a także napisaliśmy własną aplikację, która pozwala nam na wpływanie na charakterystyki ruchu przesyłanego przez sieć. Ponadto zapoznaliśmy się z biblioteką Pythona - networkx umożliwiającą badanie wykresów i sieci, a także biblioteką requests, która umożliwia w niezwykle prosty sposób wysyłanie zapytań HTTP. Nasza aplikacja daje nam możliwość dostosowywania parametrów przesyłu danych, co z kolei wpływa pozytywnie na jakość usług (QoS). To doświadczenie nie tylko poszerzyło naszą wiedzę na temat sterowania sieciami, ale także umożliwiło nam rozwinięcie umiejętności programistycznych w obszarze komunikacji sieciowej.