SCHT - Laboratorium 2

Stanisław Ciszkiewicz (324 906), Jakub Kusznier (324 924)

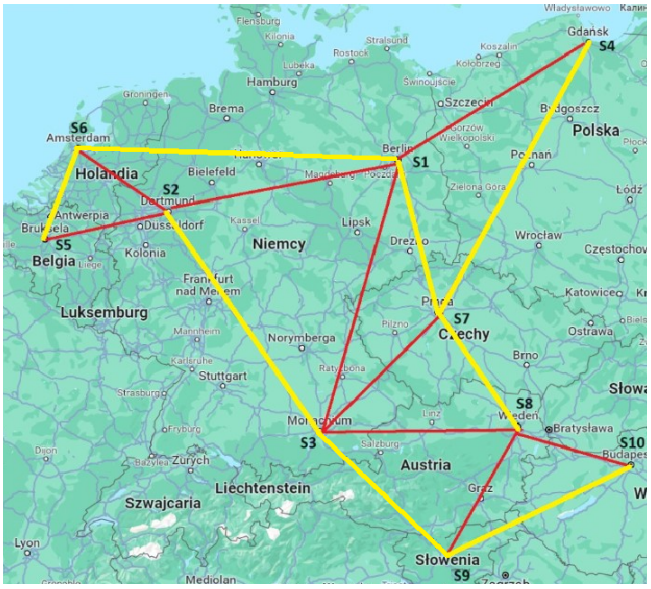
Politechnika Warszawska, Cyberbezpieczeństwo

19 listopada 2023

Spis treści

1. Zadanie 1

Do realizacji drugiego laboratorium dodaliśmy nowe połączenia pomiędzy węzłami, tworząc cykle w grafie, który teraz prezentuje się następująco:



Rys. 1. Mapa sieci z nowymi połączeniami (na czerwono połączenia z I laboratorium, na żółto nowe połączenia)

Tak jak widać na powyższej mapie, dodaliśmy połączenia s5-s6, s2-s3, s1-s7,s7-s8, s4-s7, s9-s10, s1-s6, s3-s9. W pierwszym zadaniu skonfigurowaliśmy węzły pełnej sieci z poprzedniego ćwiczenia przy użyciu styku REST kontrolera ONOS, w taki sposób, żeby wywołane pakiety w ramach przeprowadzonych testów, szły dokładnie tymi samymi drogami, co w początkowej topologii sieci. Należało w pliku .json odpowiednio skonfigurować źródłowy i docelowy adres IP, deviceId oraz numery portów. Fragment jednego z plików wygląda następująco:

```
{
  {
  "flows": [
    {
      "priority": 40000,
      "timeout": 1500,
      "isPermanent": true,
      "deviceId": "of:0000000000000006",
      "treatment": {
        "instructions": [
          {
            "type": "OUTPUT",
            "port": "1"
          }
        ],
        "deferred": []
      },
      "selector": {
        "criteria": [
          {
            "type": "IN_PORT",
            "port": "3"
          },
          {
            "type": "ETH_TYPE",
```

```
            "ethType": "0x0800"
          },
          {
            "type": "IPV4_DST",
            "ip": "10.0.0.1/32"
          }
        ]
      }
    },

}
```

Pliki .json wysłaliśmy do kontrolera skryptem zawierającym kilka komend curl. Przykładowa komenda wygląda następująco:

```
curl --user karaf:karaf -X POST "http://192.168.55.103:8181/onos/v1/flows?appId=0000000000000006" -d @Onos2s6.json
-H "Content-Type: application/json" -H "Accept: application/json"
```

Następnie przeprowadziliśmy analogiczne testy jak w poprzednim laboratorium dla nowo skonfigurowanej sieci.

1.1. TCP

1.1.1. Zależność między wielkością okna a przepustowością i RTT

Tak jak w pierwszym laboratorium, testy rozpoczęliśmy od badania wpływu okna na RTT oraz przepustowość sieci. Przy ustawieniu parametru -w na 70KB, mając ustawioną przepustowość sieci na 10 Mb/s, otrzymaliśmy czas RTT 58,480 ms (w pierwszym laboratorium wyniósł on 57,615 ms) oraz przepustowość 9,37 Mb/s (w pierwszym laboratorium wyniosła ona 9,55 Mb/s). Następnie zmniejszaliśmy parametr -w o 10KB i tak jak robiąc ćwiczenie za pierwszym razem, otrzymywaliśmy podobną przepustowość, zaś czas RTT malał. Przy parametrze -w ustawionym na 30 KB, przepustowość wyniosła 8,85 Mb/s (w pierwszym laboratorium było to 7,3 Mb/s). Dalsze zmniejszanie wielkości okna dawało tak jak w pierwszym laboratorium coraz mniejszy czas RTT, ale również spadała przepustowość. Wartością przy której przepustowość była wykorzystywana maksymalnie jest wartość -w 40KB. Wykonanie tego testu utwierdziło nas w przekonaniu, że prawidłowo skonfigurowaliśmy sieć oraz pliki .json, czyli, że połączenia idą tymi samymi trasami co w poprzednim laboratorium.

1.1.2. Zależność między wielkością bufora a przepustowością

W tym eksperymencie badaliśmy parametr -l, odpowiadający za wielkość bufora. Powtórzyliśmy testy dla tych samych ustawień sieci oraz iperf, czyli zwiększając parametr -l o 1, zaczynając od wartości 1 w pierwszym teście i otrzymaliśmy prawie identyczne rezultaty, których różnice są tak nieduże, że pozwalają dojść do tych samych wniosków, które mówią, że wielkość bufora ma znaczący wpływ na jakość transmisji danych. Za mały bufor znacząco spowalnia łącze i nie pozwala wykorzystać w pełni łącza. Wyniki testów znajdują się w tabeli poniżej.

Wartość z pierwszego laboratorium	3,87 Mb/s	7,56 Mb/s	11,2 Mb/s	13,2 Mb/s
Nowa wartość	3,4 Mb/s	7,53 Mb/s	10,6 Mb/s	10,6 Mb/s

Tabela 1. Porównanie wyników eksperymentu z pierwszego laboratorium z tymi z drugiego

1.1.3. Badanie zmian przepustowości danych po dodaniu flagi -b

W tym teście, tak jak wykonując go za pierwszym razem, badaliśmy parametr -b. Gdy ustawiliśmy parametr na wartość niższą niż przepustowość łącza (która wynosiła nadal 10 Mb/s), zaobserwowaliśmy ograniczenie przepustowości zgodnie z ustawieniem flagi -b. Przeprowadziliśmy testy przy wartościach 8 Mb/s i 3 Mb/s. Przy ustawieniu wartości -b na większą niż przepustowość sieci, klient na początku wysyłał dane z zadaną wielkością, ale po odnotowaniu strat na łączu, zmniejszał szybkość wysyłania. Aby potwierdzić poprawność interpretacji

tego parametru, zmieniliśmy wartości przepustowości dla wszystkich łączy pomiędzy h5 a h10 na 5 Mb/s. Wyniki potwierdziły nasze przekonanie - dla wartości bit rate -b ustawionej na 3 Mb/s przepustowość była ograniczona do 3 Mb/s, podobnie jak wcześniej, natomiast dla wartości ustawionej na 8 Mb/s przepustowość była ograniczona do możliwości sieci, czyli do 5 Mb/s. Drugi eksperyment dając te same rezultaty co wykonując go za pierwszym razem, potwierdził poprawność skonfigurowania plików .json.

1.1.4. Zmiana przepustowości na jednym z łączy

W tym teście zmniejszyliśmy przepustowość na łączu s1-s2 na 5 Mb/s. Po uruchomieniu skryptu z pozostałymi parametrami ustawionymi na domyślne, zauważyliśmy, że średnia przepustowość po stronie klienta wynosiła 5,31 Mb/s (poprzednio 6,67 Mb/s). Analizując logi z kolejnych sekund trwania sesji, zauważyliśmy występowanie ponownych przesłań utraconych pakietów. Łącze było przeciążone, co skutkowało znacznymi spadkami chwilowej przepustowości. W tym samym czasie przepustowość po stronie serwera utrzymywała się na poziomie około 5 Mb/s (średnia wartość z trwania sesji to 4,64 Mb/s). Otrzymane rezultaty dają takie same wnioski jak te z pierwszego laboratorium.

1.2. Wnioski

Rezultaty otrzymane po wykonaniu testów na nowej sieci, nie są identyczne jak w poprzednim laboratorium, jednak różnice są niewielkie i pozwalają wysnuć te same wnioski. Warto zaznaczyć, że przy każdej próbie odpalenia iperf otrzymywaliśmy trochę inne wyniki, jednak niewiele się różniące (wynika to z działania minineta).

1.3. UDP

1.3.1. Badanie block_rate na jakość transmisji

Pierwszy test dla UDP przeprowadziliśmy dla relacji h1-h2 (oraz h3-h9). Składał się on z 3 etapów: pierwszego dla wartości block_rate ustawionej na wartość mniejszą od optymalnej, potem równą wartości optymalnej oraz na większą wartość. Dla mniejszej wartości nie odnotowaliśmy strat i zwiększając ją rosło wykorzystanie przepustowości. Dla wartości block_rate ustawionej na wartość przepustowości (w pierwszym laboratorium wyniosła ona około 720 pps, a w drugim laboratorium około 800pps), zaobserwowaliśmy jej maksymalne wykorzystanie oraz brak strat (w pierwszym laboratorium straty wyniosły 3,8%, ale warto dodać, że przy każdym odpaleniu iperf były trochę inne, a nawet wynosiły 0). Dla coraz większych wartości block_rate przepustowość dalej pozostawała na podobnym poziomie co dla wartości optymalnej (czyli wynosiła około 9,47 Mb/s), jednak zwiększały się straty. Otrzymane rezultaty są prawie identyczne jak te z 1 laboratorium i pozwalają sformułować identyczne wnioski mówiące, że parametr bandwidth wzrasta wprost proporcjonalnie do wzrostu ilości wysyłanych pakietów aż do maksymalnego wykorzystania przepustowości sieci

1.3.2. Badanie wpływu wielkości bloku na jakość transmisji

W tym teście sprawdzaliśmy wpływ wielkości bloku na jakość transmisji.

-l	50	100	200	1300	1500
Laboratorium 1(% strat;przepustowość Mb/s)	82%;1,79	71%;3	54%;4,78	0%; 9,41	7,7%;9,41
Laboratorium 2(%strat;przepustowość Mb/s)	83%;1,76	72%;2,97	52%;4,81	0%;9,42	7,5%;9,41

Tabela 2. Tabela porównująca wyniku testu wpływu wielkości bloku na QoS dla 1 i 2 laboratorium

Tak jak w poprzednich testach, otrzymane rezultaty są niemal identyczne i formułujemy z nich te same wnioski, czyli że przy przesyłaniu danych protokołem UDP, ważne jest, żeby dobrze dobrać wielkość obsługiwanych bloków. Ustawienie za małej wartości skutkuje przepełnieniem się bufora i pojawieniem się strat oraz spadkiem przepustowości. Ustawienie za dużej wartości również nie jest pożądane, ponieważ proces fragmentacji IP wymaga dodatkowego czasu, co poskutkuje obniżeniem się jakości świadczonej usługi.

1.3.3. Zmiana przepustowości na jednym z łączy

Analogicznie jak w 1 laboratorium zdecydowaliśmy się zbadać, co się wydarzy, gdy ograniczymy przepustowość na jednym z połączeń, przez które odbywa się transfer danych. Wybraliśmy połączenie między h1 (serwerem) a h9 (klientem). Na łączu między s3 a s8 zmniejszyliśmy parametr bw o połowę, co spowodowało

obniżenie go do 5 Mb/s. Zanotowaliśmy straty na łączu na poziomie 45% (w pierwszym laboratorium 40 %), a przepustowość łącza spadła do 4,82 Mb/s (w pierwszym laboratorium 4,83 Mb/s). Wyciągnęliśmy wniosek, że przesył danych musi odbywać się jedną drogą oraz że ograniczenie przepustowości na jednym łączu wpłynie negatywnie na jakość transmisji dla wszystkich relacji przechodzących przez to połączenie. Jest to identyczny wniosek jak ten z pierwszego laboratorium.

Tak jak w pierwszym laboratorium, powtórzyliśmy testy dla relacji h3-h9. Otrzymaliśmy bardzo podobne rezultaty, z których wyciągamy te same wnioski.

1.3.4. Wnioski

Rezultaty otrzymane w wszystkich testach są prawie identyczne (różnice wynikają z działania minineta) i wyciągamy z nich te same wnioski. Dowiodło to, że prawidłowo skonfigurowaliśmy ścieżki przy pomocy styku REST, czyli tak, żeby przesył danych odbywał się dokładnie tymi samymi ścieżkami co w laboratorium numer 1 pomimo dodania nowych ścieżek, które mogą być bardziej optymalne.

2. Zadanie 2

W testach w zadaniu 2 wykonywaliśmy testy puszczając jednocześnie kilka połączeń zarówno TCP, jak i UDP. Testy miały pokazać, że dobierając lepsze ścieżki w grafie, jesteśmy w stanie uzyskać lepszą przepustowość na łączach.W testach, w których z różnych klientów puszczaliśmy połączenia do serwera znajdującego się na tym samym hoscie, połączenia i tak na samym końcu szły tą samą drogą, drogą od switcha do docelowego hosta, powodując tym samym straty na łączu oraz obniżenie przepustowości. Aby zniwelować ten problem, zwiększyliśmy przepustowość na ostatnim odcinku trasy, który jest zarazem jedynym współdzielonym odcinkiem (odcinek pomiędzy ostatnim switchem i docelowym hostem) tak, żeby ten odcinek nie wpływał negatywnie na wynik testów. Dzięki temu uzyskaliśmy lepsze rezultaty i mogliśmy potwierdzić, że połączenia idą innymi drogami.

2.1. TCP

Na początku zrobiliśmy test tylko dla połączenia h10-h1. Otrzymaliśmy przepustowość 9,53 MB/s (przepustowość łącza w wszystkich testach wynosi 10 MB/s). Następnie puściliśmy tylko połączenie h9-h1. Tutaj przepustowośc wyniosła 9,49 Mb/s. Następnie puściliśmy 2 połączenia równocześnie; dla obu relacji otrzymaliśmy równe wartości, 4,76 Mb/s. W kolejnym teście puściliśmy jednocześnie 4 połączenia, które współdzieliły odcinek na trasie, były to połączenia z h7, h8, h9, h10 do h1. Otrzymaliśmy przepustowość dla h10 2,33 Mb/s (poprzednio 2,21 Mb/s), dla h9 2,41 Mb/s (poprzednio 2,22 Mb/s), dla h8 2,44 Mb/s (poprzednio 2,15 Mb/s) i dla h7 2,46 Mb/s (poprzednio 2,2 Mb/s). W 2 poprzednich testach widać, że hosty podzieliły się mniej więcej po równo dostępną przepustowością, czyli jest to taki sam wniosek jak w laboratorium 1.

W kolejnym teście wróciliśmy do 2 hostów, h9 i h10 i ustawiliśmy różne wielkości okna, 30 KB dla h9 i 5 KB dla h10. Dla większej wartości okna otrzymaliśmy przepustowość 8,46 MB/s (poprzednio 8,3 MB/s), zaś dla mniejszej 1,17 MB/s (poprzednio 1,14 MB/s). Tak jak w pierwszym laboratorium, testy nie trwały tyle samo czasu i ten z większym oknem skończył się szybciej i w tym momencie nastąpił wzrost przepustowości dla testu z mniejszym oknem.

Zrobiliśmy jeszcze test, w którym zmniejszyliśmy przepustowość na jednym łączu (h1-h3) z 10 MB/s na 5 MB/s. Tak jak w poprzednim laboratorium nastąpił spadek przepustowości tak, żeby nie przekraczać ustawionej wartości. Puszczając połączenia z h9 i h10 otrzymaliśmy kolejno 2,3 MB/s (poprzednio 2,36 MB/s) oraz 2,52 MB/s (poprzednio 2,58 MB/s). Test powtórzyliśmy dla hostów h2-h7-h8 (Dortmund - Praga - Wiedeń), a uzyskane wyniki utwierdziły nas w przekonaniu, że rezultaty uzyskane w eksperymentach na relacji h1-h9-h10 nie były przypadkiem.

2.1.1. Zmienione drogi

W tym teście puściliśmy połączenia h10-h1 oraz h9-h1 innymi ścieżkami. Pierwsza ścieżka idzie następująco s10-s8-s7-s1, a druga s9-s3-s1. Puszczając po jednym połączeniu, przepustowość była maksymalnie wykorzystywana, dla relacji h10-h1 otrzymaliśmy 9,22 Mb/s, a dla relacji h9-h1 9,29 Mb/s. Puszczając oba połączenia naraz, otrzymaliśmy podobną przepustowość, dla h10 otrzymaliśmy 9, 33 Mb/s, a dla h9 9,3 Mb/s. Dowodzi to, że połączenia idą nowymi ścieżkami, ponieważ nie odnotowujemy już zmniejszonej przepustowości (w testach w 1 laboartorium wyszło po 5,2 Mb/s dla obu połączeń jak współdzieliły trasę).

Puściliśmy jeszcze 4 połączenia jednocześnie, z h10, h9, h8, h7 do h1. Znaleźliśmy 4 oddzielne trasy, takie, żeby połączenia nie współdzieliły odcinków (poza odcinkiem s1-h1, na którym zwiększyliśmy przepustowość na potrzeby testów). Trasy wyglądały następująco: h9-S9-S3-S1-h1, h7-S7-S1-h1, h8-S8-S7-S4-S1-h1, h10-S10-S8-S3-S2-S1-h1. W testach, przepustowość była maksymalnie wykorzystana, co dowiodło, że połączenia szły nowymi, lepszymi trasami (lepszymi, czyli takimi, które pozwalają dla każdego połączenia wykorzystać maksymalnie możliwości sieci).

2.2. UDP

W tym ćwiczeniu na początku testowaliśmy te same drogi, co w laboratorium 1. Sprawdziliśmy pojedyncze połączenia h6-h1 oraz h5-h1. Dla puszczonej samej relacji h6-h1 otrzymaliśmy przepustowość 9,29 Mb/s, a dla puszczonej samej relacji h5-h1 9,44 Mb/s. następnie puściliśmy 2 te same połączenia jednocześnie, co spowodowało przechodzenie przez te samo łącze w tym samym czasie. Dla wartości sumarycznie poniżej przepustowości łącza nie odnotowaliśmy spadku przepustowości po stronie serwera (nawet jak jedno połączenie wymagało większej przepustowości niż drugie, ale razem nie przekraczało 10 Mb/s np. 3Mb/s i 5Mb/s), zaś dla wartości większej nastąpił spadek przepustowości. Obaj klienci próbowali wysyłać dane z szybkością około 9Mb/s, zaś serwer od h6 otrzymywał dane z przepustowością 4,77 Mb/s co dało straty 51%, a od h5 3,76 Mb/s co dało straty 57%. Dowodzi to, że przesył danych szedł tą sama ścieżką.

W tym teście ustawiliśmy parametr -b na 5 Mb/s dla relacji h5-h1 oraz 15 Mb/s dla relacji h6-h1. Uzyskaliśmy przepustowość 8,21 Mb/s (w pierwszym laboratorium 6,6 Mb/s) oraz straty na poziomie 44% (w pierwszym laboratorium 56%) dla relacji h6-h1, a dla relacji h5-h1 przepustowość wyniosła 1,4 Mb/s (w 1 laboratorium 3,08 Mb/s), a straty 72 % (w pierwszym laboratorium 39%). Z tych testów wyciągamy taki sam wniosek jak z tych z pierwszego laboratorium, czyli, że UDP nie dzieli dostępnej przepustowości sprawiedliwie, h5 żądało 3 razy mniejszej przepustowości, a otrzymało ponad 5 razy mniejszą.

Uruchomiliśmy 4 sesje jednocześnie. Hosty h7, h8, h9, h10 były klientami, a serwer postawiliśmy na h10. Puszczając wszystkie połączenia jednocześnie, sesje rywalizowały o łącze. Host h10 uzyskał przepustowość 1,47 Mb/s oraz straty 84%, h9 1,39 Mb/s oraz straty 85%, h8 1,92 Mb/s oraz straty 80%, h7 4,71 Mb/s oraz straty 52%. Tak jak w 1 laboratorium, w przypadku UDP podział nie był równy (sprawiedliwy).

2.2.1. Zmiana drogi

Dla poprawy rezultatów ustanowiliśmy nową ścieżkę dla relacji h6-h1; jest to bezpośrednie połączenie pomiędzy switchami s6 i s1. Połączenie h5-h1 puszczaliśmy tą samą ścieżką co w poprzednim teście, czyli połączenia szło s5-s2-s1. Uzyskaliśmy poprawę rezultatów, każde z łączy było wykorzystywane maksymalnie, łącza nie rywalizowały ze sobą; dla h5-h1 otrzymaliśmy 9,41 Mb/s, a dla h6-h1 9,26 Mb/s.

Postanowiliśmy puścić jednocześnie 2 połączenia, każde wysyłające z prędkością 10 Mb/s. Serwer dla relacji h6-h1 otrzymywał dane z przepustowością 9,56 Mb/s, zaś dla relacji h5-h1 9,58 Mb/s. Klient wysyłał zgodnie z ustawioną przepustowością, jednak serwer odbierał dane z zmniejszoną przepustowością tak, żeby nie przekraczać przepustowości sieci. Spowodowało to straty na łączu, jednak wynikały one z za dużego przesyłu ze strony klienta w stosunku do możliwości łącza, a nie z współdzielenia odcinka na trasie.

Puszczając jednocześnie sesje h5-h1, która żąda przepustowości 5 Mb/s oraz h6-h1, która żąda 15 Mb/s, łącza nie musiały z sobą rywalizować i relacja h5-h1 nie uzyskała start, zaś relacja h6-h1 odnotowała straty, jednak wynikały one z tego, że sieć ma ustawiona przepustowość na 10 Mb/s, więc nie można w niej puścić tak dużego połączenia. Lepsze trasy dla tego połączenia, które zastosowaliśmy to s6-s1 oraz s5-s2-s1.

W celu uzyskania lepszych rezultatów dla puszczonych 4 sesji jednocześnie, ustanowiliśmy nowe drogi. Wyglądały one teraz następująco: s9-s3-s1, s7-s1, s8-s7-s4-s1, s10-s8-s3-s2-s1. Dla nowych tras, sesje nie musiały rywalizować o łącze, dzięki czemu każda sesja wykorzystywała przepustowość maksymalnie, a starty były niewielkie.

2.3. TCP vs UDP

Odpalenie jednocześnie połączeń UDP oraz TCP, w naszym przypadku z hostów h9 i h10 do h1, dało dokładnie te same rezultaty co poprzednie testy i poprzednie laboratorium. Jeżeli ścieżka była współdzielona, to przy ustawieniu wartości block size na małą wartość dla TCP, UDP dominowało TCP do czasu zakończenia sesji UDP, po zakończeniu sesji UDP, TCP miało większa przepustowość. Można było zwiększyć przesył dla TCP zwiększając parametr block size, wtedy uzyskiwało ono większe wartości. Żeby uzyskać maksymalną prze-

pustowość łącza dla obu połączeń należało rozdzielić ich trasy. Z h9 połączenie szło s9-s3-s1, a z h10 s10-s8-s7-s1. Wtedy łącza nie rywalizowały ze sobą i serwer otrzymywał dane z obu hostów z maksymalną przepustowością.

2.4. Wnioski

Testy udowodniły, że jak puszczane relacje na łączach współdzieliły odcinki na łączu to odnotowywaliśmy spadek przepustowości po stronie serwera. Poprawę uzyskaliśmy wyznaczając oddzielne trasy dla poszczególnych połączeń tak, żeby nie współdzieliły odcinków na trasie. Wtedy przepustowość była wykorzystywana maksymalnie przez każde poszczególne połączenie. warto dodać, że w przypadku testów dla TCP, dla których wyznaczyliśmy nową trasę na odcinku s9-s3-s1, czas był minimalnie dłuższy niż dla poprzedniej trasy, czyli tej, która współdzieliła odcinek, jednak minimalnie większy delay na łączu nie wpłynął na wyniki testu, a udało nam się pokazać, że dla połączeń idących innymi trasami mamy większą przepustowość na poszczególnych łączach.

3. Zadanie 3

W tym zadaniu w skrócie opiszemy najważniejsze elementy charakterystyki działania oraz implementacji naszej aplikacji sterującej siecią. Jak mogliśmy zauważyć w poprzednich zadaniach jednoczesne puszczenie wielu strumieni jednym łączem może skutkować przeciążeniem i spadkiem jakości oferowanej usługi transportowej. Tym samym, konieczne stało się dostosowywanie dróg do obecnych warunków obserwowanych na łączach. Głównym zadaniem naszej aplikacji było obliczenie najbardziej optymalnej ścieżki, stworzenie plików .json i wysyłanie ich na serwer onosowy.

3.1. Wczytanie grafu z pliku

Pierwszym wykonanym przez nas krokiem było wczytanie informacji o naszej sieci z przygotowanego przez nas pliku txt. Jak widać poniżej, informacje o węzłach i łączach zapisane są w następującym formacie:

- łącza między switchami - każda z kolejnych danych oddzielona spacją (a b delay bw porta portb)
  - a - switch, pierwszy z dwóch między którymi występuje połączenie
  - b - switch, drugi z dwóch między którymi występuje połączenie
  - delay - opóźnienie występujące na łączu między switchami a-b
  - bw - przepustowość łącza między switchami a-b
  - porta - port na switchu a, do którego podłączone jest łącze a-b
  - portb - port na switchu b, do którego podłączone jest łącze a-b
- numer portu (p), na którym do n-tego switcha podłączony jest jego host (format %n p)

Poniżej prezentujemy fragment pliku graf.txt:

```
1 2 2.97 10 1 1
1 3 3.57 10 2 1
1 4 2.85 10 3 1
2 5 1.61 10 2 1
2 6 1.43 10 3 1
3 7 2.09 10 2 1

.......

%1 5
%2 5
%3 5
%4 3
%5 3
%6 3
```

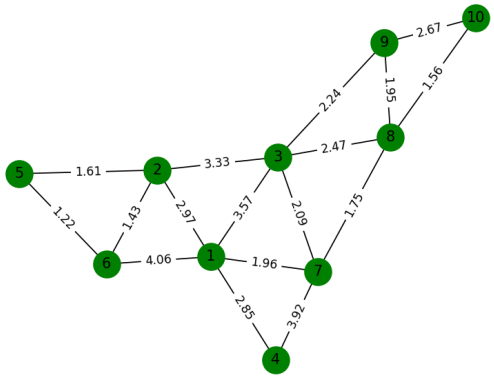
Dane w pliku graf.txt zapisane były w taki sposób, który umożliwiał nam dalsze swobodne użycie ich w obliczeniach i nie ograniczał uniwersalności naszej aplikacji (jeśli charakterystyka innej sieci zostanie podana

w tym formacie aplikacja wyznaczy najbardziej optymalne ścieżki). W tym pliku znajdują się także wszystkie informacje potrzebne do dalszego przesłania żądania na serwer (chociażby numery portów wejściowych i wyjściowych). Wczytania grafu z pliku dokonuje metoda read_graph_from_file(), która: tworzy graf i dodaje do niego krawędzi - metoda graph.add_edge(a,b,delay), tworzy kolekcję edge, w której przechowujemy switche jako klucze wraz z listą wszystkich ich połączeń wraz z portami, do których podłączony jest link po obu stronach (ports i portd) oraz przepustowością, i tablicę hostsports, w której zapisane są numery portów na switchach, do których podpięte są hosty.

3.2. Klasa main

W klasie main działanie rozpoczyna nasza aplikacja - tworzony jest obiekt klasy Controller. Tutaj wywoływane są m.in. funkcje wczytania z pliku, rysowania grafu (dzięki bibliotece networkX) czy na końcu wysłania plików .json na serwer onosowy. Ponadto wywoływana jest metoda input panel, w której klient wprowadza kolejne połączenia i czeka na wyliczenie najlepszej ścieżki.

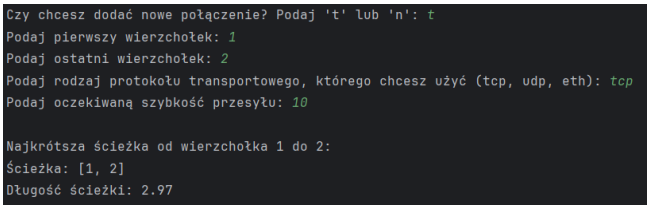
```
class Main:
    if __name__ == "__main__":
        filename = "graf.txt"
        controller = Controller(filename)
        controller.draw()
        controller.copy_and_save("template.json", "tosend.json")
        controller.input_panel()

        curl_command = (
            'curl --user karaf:karaf -X POST "http://192.168.0.143:8181/onos/v1/flows" '
            '-d @tosend.json -H "Content-Type: application/json" -H "Accept: application/json"'
        )

        result = subprocess.run(curl_command, capture_output=True, text=True, shell=True)
        print(result.returncode)
        print(result.stdout)
        print(result.stderr)
```



Rys. 2. Rysunek grafu wygenerowany metodą nx.draw z pakietu networkX

3.3. Metoda input_panel

Metoda klasy Controller, w jej ciele następuje interakcja aplikacji z użytkownikiem - zapytanie o parę hostów między którymi chcemy nawiązać połączenie oraz przepustowość z jaką chcielibyśmy przesyłać dane. Po sprawdzeniu poprawności wpisanych danych (w razie błędnego wpisania użytkownik musi poprawić wpis) następuje uruchomienie głównej metody wyliczającej ścieżkę - finding_path. Po wyliczeniu i wyświetleniu wyników w terminalu przechodzimy do przygotowania plików .json do przesłania na serwer - metoda file_builder. W momencie gdy, użytkownik chce zakończyć wpisywanie połączeń (na zapytanie o dodaniu nowego połączenia odpowiada "n") program opuszcza funkcję.



Rys. 3. Zrzut ekranu z terminala, pokazujący kolejne kroki ustawiania żądania w aplikacji

3.4. Opis wybranego algorytmu, metoda finding_path

Najważniejszym i najtrudniejszym etapem tego ćwiczenia było wymyślenie najbardziej efektywnego sposobu znajdywania ścieżek w grafie, który spowoduje maksymalne poprawienie jakości oferowanej usługi. Po dogłębnej analizie doszliśmy do wniosku, że żadne z rozwiązań nie będzie idealne - każde będzie posiadać swoje wady i zalety. Jednocześnie nasza aplikacja miała obserwować obecny stan łączy i brać go pod uwagę podczas wyliczania ścieżki. Ostatecznie postawiliśmy na rozwiązanie, które:

- w pierwszym kroku szuka najkrótszej ścieżki w grafie (wagą jest opóźnienie) oraz sprawdza czy ta ścieżka posiada wystarczająco dużo wolnej przepustowości do obsłużenia żądania
- jeśli wyliczona ścieżka nie daje szans obsłużenia żądania, aplikacja usuwa przeciążone łącza z grafu i następnie po raz kolejny wylicza najkrótszą ścieżkę (do momentu znalezienia takiej, która będzie w stanie ją obsłużyć)
- w przypadku braku możliwości znalezienia najkrótszej ścieżki, program odmawia obsłużenia klienta, wysyłając mu stosowny komunikat.

Za powyżej opisane kroki w naszym kodzie odpowiada metoda finding_path, która zwraca optymalną ścieżkę, zaktualizowaną listę łączy oraz długość najkrótszej ścieżki. Dzięki wykorzystaniu biblioteki networkX możemy w bardzo prosty sposób metodą nx.shortest_path() znaleźć najkrótszą ścieżkę w grafie. W metodzie left_bandwidth_check() następuje sprawdzenie, czy w wybranej ścieżce zmieści się strumień (jeśli nie, usuwane są zbyt obciążone krawędzi - metoda remove_edge() z biblioteki networkX i poszukiwania trwają nadal). Na samym końcu po wyliczeniu ścieżki modyfikowana jest lista edges (metoda decrease_bw), a konkretnie liczba i rodzaje strumieni znajdujących się na łączu.

```
def finding_path(self, start, end, stream_bw,type):
        is_possible = False
        copy_graph = self.graph.copy()

        while (is_possible==False):
            try:
                is_possible = True
                path = nx.shortest_path(self.graph, start, end, weight='weight')
            except NetworkXNoPath:
                print("Brak mozliwosci realizacji twojego zadania, sprobuj pozniej.")
                path=None
                path_length=0
                self.graph = copy_graph
                return (path, path_length)
            self.graph, is_possible = self.left_bandwidth_check(path, stream_bw, is_possible)
        self.decrease_bw(path, stream_bw,type)
        path_length = nx.shortest_path_length(self.graph, start, end, weight='weight')
        self.graph = copy_graph
        return path, path_length
```

3.5. Obserwacja stanu łączy

Istotnym był również sposób obserwacji i aktualizacji obecnego stanu łączy. Założyliśmy, że żądana przepustowość podawana przez klienta będzie minimalną, jaką ostatecznie powinien dostać strumień. Tym samym, pierwszeństwo w dostępie do łącza miały wcześniej wprowadzone strumienie. Analizując zachowanie naszej sieci aktualną ilość dostępnej przepustowości określiliśmy wzorem: $possible\_bw = \frac{link_{bw} - udp}{how\_many\_tcp+1}$. Wzór ten sprawdzał jak dużo mamy dostępnej przepustowości, natomiast nawet gdy dostępna przepustowość była wystarczająca przechodził do sprawdzenia tablicy tcp w słowniku danego łącza - musiał sprawdzić czy nie ograniczy już wcześniej znajdujących się tam strumieni.

3.6. Metoda file_builder

W metodzie file_builder za pomocą doklejania kolejnych reguł tworzymy plik .json, który na koniec wysyłany jest na serwer onosowy. W tym celu przygotowaliśmy sobie "wzorcową regułę", w której kolejne parametry (deviceID, INPORT, OUTPORT itd.) zapisaliśmy jako kolejno %1, %2, %3 itd. Po doklejeniu takiej reguły szukaliśmy konkretnych fragmentów zawierających znak procent wraz z daną liczbą i podmienialiśmy na wartości znajdujące się w np. kolekcji edge. Poniżej prezentujemy fragment kodu zapisującego jedną z reguł:

```
def file_builder(edges,filename, shortestpath,hostports,how_many_streams):
        with open(filename, "r") as file_to_send:
            content = file_to_send.read()
            with open("szablondowzoru.json", "r") as file_template:
                new_content = file_template.read()
                if(how_many_streams>1):
                    new_content += ","
                line = content.split('\n')
                line.insert(2, new_content)
                content = '\n'.join(line)

                content = content.replace("%1", str(hex(int(shortestpath[0]))[2:]))
                content = content.replace("%2", str(int(get_edge(edges,shortestpath[0],shortestpath[1])[3])))
                content = content.replace("%3", hostports[shortestpath[0]-1])
                content = content.replace("%4", str(shortestpath[len(shortestpath)-1]))
                content = content.replace("%5", str(shortestpath[0]))

                ........

        with open(filename, 'w') as file_to_send:
        file_to_send.write(content)
```

4. Zadanie 4

W zadaniu 4 mieliśmy pokazać, że zaprojektowana przez nas aplikacja dobierze lepsze ścieżki dla testów realizowanych w zadaniu 5 w laboratorium 1.

4.1. TCP vs TCP

4.1.1. Badanie jakości połączenia przy kilku sesjach TCP przechodzących przez to samo łącze

W pierwszym kroku zbadaliśmy relację dwóch klientów (h10 i h9) wysyłających jednocześnie strumień na serwer znajdujący się na h1. Najpierw sprawdziliśmy, czy gdy obydwa hosty zażądają minimalnej przepustowości 3 Mb/s (suma będzie mniejsza od przepustowości wspólnego łącza), czy aplikacja przydzieli im ścieżkę, ze wspólnym łączem. Tak się stało (dla h10 - s10-**s8-s7**-s1), dla h9 - s9-**s8-s7**-s1, , a żaden z klientów nie stracił na takim rozwiązaniu (dostali kolejno: 5,12 i 4,53 Mb/s; zażądali w aplikacji minimum 3 Mb/s, czyli jak w sieci jest dostępne więcej, to mogą mieć więcej). W kolejnym teście każdy z użytkowników chciał wysłać strumień TCP z przepustowością 10 Mb/s. Zgodnie z przewidywaniami, aplikacja przydzieliła im całkowicie różniące się ścieżki, które nie miały żadnego wspólnego połączenia (dla h10 - s10, s8, s7, s1 i dla h9 - s9, s3, s1). Całkowicie rozdzielne ścieżki dały każdemu strumieniowi maksymalną przepustowość(kolejno 9.17 Mb/s i 9.14 Mb/s).

Dla utwierdzenia się w poprawności działania aplikacji przeprowadziliśmy test puszczający 4 strumienie z łączy: h10, h9, h8 i h7 na serwer znajdujący się na hoście h1. Najpierw każdy ze strumieni miał w aplikacji ustawioną żądaną przepustowość na 2 Mb/s (wszystkie powinny się zmieścić we wspólnym łączu). Aplikacja zwraca nam następujące ścieżki: dla h10 - s10, s8, **s7, s1**, dla h9 - s9, s8, **s7, s1**, dla h8 - s8, **s7, s1** i dla h7 - **s7, s1**. Ostatecznie każdy z strumieni otrzymał kolejno: 2,31 Mb/s, 2,29 Mb/s, 2,29 Mb/s i 2,26 Mb/s - przepustowość żadnego z żądań nie spadła poniżej minimalnej, oczekiwanej przez klienta. Podobnie jak dla dwóch połączeń, sprawdziliśmy co się wydarzy gdy żądana przepustowość będzie wynosiła 10 Mb/s. Program przydzielił każdemu z żądań różne ścieżki: h10 - s10, s8, s7, s1, h9 - s9, s3, s1, h8 - s8, s3, s2, s1 i h7 - s7, s4, s1. Każde dostało kolejno: 9,28 Mb/s, 9,27 Mb/s ,9,24 Mb/s i 9,30. Tym samym utwierdziliśmy się w przekonaniu, że nasza aplikacja sterująca siecią poprawia jakość wykonywanej usługi transportowej dla wielu równoległych połączeń TCP.

4.1.2. Ograniczenie przepustowości na jednym z łączy

Następnie zmniejszyliśmy przepustowość na łączu s1-s7 na 5 Mb/s. Celem było sprawdzenie jak zachowają się strumienie idące z hostów h10 i h9. Żądana przepustowość wynosiła 3 Mb/s - pierwsze żądanie z hosta h10 przeszło przez zwężoną ścieżkę (s10, s8, s7, s1) z przepustowością: 4,64 Mb/s, natomiast z h9 musiało ominąć (s9,s3, s1) z przepustowością:8,85 Mb/s. Ostatecznie sprawdziliśmy jak zachowają się strumienie, gdy żądana wartość przepustowości będzie większa od 5 Mb/s. Zgodnie z przewidywaniami aplikacja przekierowała strumień z h10 na inną ścieżkę niż poprzednio (s10, s8, s3, s1), tak żeby nie utracił na jakości (otrzymał przepustowość 9,28 Mb/s), natomiast h9 dostał ścieżkę: s9, s3, s2, s1 z przepustowością 9,29 Mb/s.

4.2. UDP vs UDP

4.2.1. Badanie jakości połączenia przy kilku sesjach UDP przechodzących przez to samo łącze

W związku z tym, że w poprzednim laboratorium robiliśmy relację h5-h1 oraz h6-h1 dla współdzielonego łącza musieliśmy zmienić testowaną relacje (w nowej sieci nie mielibyśmy szansy na uzyskanie sytuacji, w której te dwie relacje dzieliłyby łącza - h6 ma bezpośrednią drogę do h1, więc zawsze wybierze inną ścieżkę). Zdecydowaliśmy tym samym powtórzyć testy z poprzedniego laboratorium dla relacji h5-h3 i h6-h3, co umożliwi nam zaobserwowanie zależności na nowej sieci. Na starej topologii zrobiliśmy wyżej opisany test: h6-h3 na współdzielonej ścieżce otrzymało 5,24 Mb/s przepustowości (obserwowane straty to 46%) natomiast h5-h3 otrzymało 4,2 Mb/s (straty 57%). Mając te wyniki przeszliśmy do analizy jakości przesyłu przy użyciu zaprojektowanej przez nas aplikacji. Obydwa strumienie oczekiwały przepustowości 10 Mb/s - aplikacja wydzieliła im rozdzielne ścieżki: h6 - s6, s2, s3, a h5 - s5, s2, s1, s3. Zaobserwowaliśmy tym samym znaczną poprawę jakości usługi: relacja h5-h3 zanotowała przepustowość wynoszącą 9,04 Mb/s (straty na poziomie 11%), a h6-h3 9,02 Mb/s (straty wynosiły 11%) - zauważamy znaczący wzrost jakościowy oferowanej usługi.

Dla 4 połączeń (h10-h1, h9-h1, h8-h1 i h7-h1) również zaobserwowaliśmy znaczącą poprawę, gdyż tym razem ścieżki były rozłączne dla każdej relacji. Każdy z kolejnych strumieni otrzymał następujące rezultaty: (h10 - przepustowość 9,56 Mb/s, straty 5,2%, h9 - przepustowość 9,59 Mb/s, straty 5,3%, h8 - 9,47 Mb/s, straty 5,1%, h7 -przepustowość 9,52 Mb/s, straty 5,1% strat.

4.2.2. Ograniczenie przepustowości na jednym z łączy

Podobnie jak w poprzednim laboratorium ograniczyliśmy przepustowość do 5 na łączu s1-s3 i puściliśmy połączenia w relacji h9-h1 i h7-h1 (poprzednio używały tej samej, zwężonej drogi). Tym razem dla zadanej przepustowości 10 Mb/s aplikacja przydzieliła im różne ścieżki, jednak obydwie omijające zwężony odcinek s1-s3: h9 - s9, s8, s7, s1 oraz h7 - s7, s4, s1. Ponownie uzyskane rezultaty były znacząco lepsze: przepustowość była praktycznie maksymalna (ok. 9,5 Mb/s), natomiast straty spadły do poziomu 5,6%.

4.3. UDP vs TCP

Puszczając oba połączenia jednocześnie, program dobrał trasy tak, żeby ścieżki nie były współdzielone. Połączenie TCP szło h1-S1-S8-S7-S1-h1, a połączenie UDP h9-S9-S3-S1-h1. Dzięki temu połączenia nie rywalizowały o łącze i nie zaobserwowaliśmy dominowania UDP przez TCP dla domyślnych wartości iperf. Zmienianie parametru block size w tym przypadku nie ma sensu, ponieważ nie uzyskamy dzięki temu lepszych parametrów dla TCP, tak jak miało to miejsce w pierwszym laboratorium, gdzie mogliśmy w ten sposób uzyskać większą przepustowość przy współdzieleniu trasy przez TCP i UDP.

4.4. Wnioski

Testy wykonane dla ścieżek wyznaczonych przy użyciu naszej naszej aplikacji dały lepsze rezultaty, niż te uzyskane w pierwszym laboratorium. Pozwoliło to na maksymalne wykorzystanie łącza przy uruchomionych kilku sesjach jednocześnie, dzięki temu trasa nie była współdzielona i można było uzyskać lepszy efekt. Co prawda podobne efekty uzyskaliśmy w drugim zadaniu, jednak tam był potrzebny człowiek, który sam musiał ręcznie konfigurować pliki .json i samemu zastanawiać się jaka trasa poprawi jakość transmisji. Aplikacja sama dobiera najlepszą możliwą ścieżkę i konstruuje plik .josn do zarządzania jej trasą. Dowodzi to, że nasza aplikacja spełnia swoje zadanie i poprawia jakość świadczonej usługi.

5. Podsumowanie

Praca nad laboratorium pokazała nam złożony problem wyszukiwania najbardziej optymalnej drogi dla strumieni w sieci - można zastosować całkowicie różne podejścia, przy czym dla każdego można znaleźć jego wady i zalety. Model sieci, na którym pracowaliśmy był znacznie mniej skomplikowany od topologii prawdziwych sieci, których projektowanie wymaga zdecydowanie bardziej dogłębnych analiz (przygotowanie na zmiany obciążenia sieci w czasie itp.). Mimo to, ćwiczenie w bardzo dobry sposób obrazuje konieczność obserwacji obciążenia sieci i dobierania w miarę możliwości takich rozwiązań, które poprawią jakość oferowanej usługi. Tematykę laboratorium uważamy za bardzo ciekawą, a uzyskany efekt końcowy pokazuje, że ostateczne rozwiązanie oferowane przez aplikacje sprawdza się zdecydowanie najlepiej - jest zautomatyzowane (nie trzeba wszystkich reguł aktualizować ręcznie), a po drugie tworzy ścieżki analizując obecny stan sieci. Wiedza zdobyta w trakcie wykonywania tego laboratorium ma zastosowanie praktyczne w zadaniach powiązanych z zarządzaniem ruchem w sieci, co jest niezbędne w pracy w cyberbezpieczeństwie.