SCHT - Lab1

Stanisław Kwiatkowski, Bartosz Ziemba

Spis treści

1. Wprowadzenie

Głównym celem tego laboratorium było zrozumienie sposobu działania sieci internetowej, dlatego też posłużyliśmy się programem mininet na którym zdefiniowaliśmy własny model imitujący sieć pomiędzy miastami na kontynencie Europejskim. Następnie przeprowadziliśmy na niem szereg testów symulujących prawdziwy ruch sieciowy przy użyciu narzędzia iperf, dzięki czemu mogliśmy zaobserwować zachodzące w niej procesy, przede wszystkim te występujące przy zbliżaniu się oraz przekraczaniu zasobów udostępnianych przez sieć. Całość laboratorium była podzielona na dwie części.

2. Część pierwsza

By przygotować się do tej części stworzyliśmy mapę połączeń między dziesięcioma europejskimi stolicami oraz wyznaczyliśmy odległość między nimi.

Rys. 1: Mapa Sieci

2.1. Tworzenie sieci

W kolejnym kroku zapoznaliśmy się z narzędziem mininet oraz przy użyciu Pythona zdefiniowaliśmy topologie sieci dla wcześniej opracowanej mapy uwzględniając rzeczywiste odległości i opóźnienia z nimi związane. Dla wszystkich relacji sieci, czyli połączeń pomiędzy poszczególnymi węzłami ustawiliśmy jednakową przepustowość sieci równą 80 Mb/s.

```
h1 = self.addHost('Londyn')
```

Rys. 2: Sposób inicjalizacji Hosta

```
self.addLink(s1,s2,delay='2ms', bw=80)
```

Rys. 3: Sposób tworzenia połączenia między switchami

2.2. Sprawdzanie działania sieci

Następnie podjęliśmy się prostego sprawdzania poszczególnych połączeń przy użyciu protokołu ICMP. Użycie komedny *pingall* dało oczekiwane rezultaty, ponieważ każdy z hostów ma połączenie ze wszystkimi innymi co oznacza że sieć jest spójna.

```
mininet> pingall
*** Ping: testing ping reachability
Ateny -> Berlin Londyn Madryt Paryz Praga Rzym Warszawa Zagrzeb Zurych
Berlin -> Ateny Londyn Madryt Paryz Praga Rzym Warszawa Zagrzeb Zurych
Londyn -> Ateny Berlin Madryt Paryz Praga Rzym Warszawa Zagrzeb Zurych
Madryt -> Ateny Berlin Londyn Paryz Praga Rzym Warszawa Zagrzeb Zurych
Paryz -> Ateny Berlin Londyn Madryt Praga Rzym Warszawa Zagrzeb Zurych
Praga -> Ateny Berlin Londyn Madryt Paryz Rzym Warszawa Zagrzeb Zurych
Rzym -> Ateny Berlin Londyn Madryt Paryz Praga Warszawa Zagrzeb Zurych
Warszawa -> Ateny Berlin Londyn Madryt Paryz Praga Rzym Zagrzeb Zurych
Zagrzeb -> Ateny Berlin Londyn Madryt Paryz Praga Rzym Warszawa Zurych
Zurych -> Ateny Berlin Londyn Madryt Paryz Praga Rzym Warszawa Zagrzeb
*** Results: 0% dropped (90/90 received)
```

Rys. 4: Potwierdzenie spójności sieci

Kolejnym krokiem było sprawdzanie czy wcześniej utworzona sieć faktycznie zachowuje nadane jej parametry opóźnień. Co uzyskaliśmy poprzez pingowanie odpowiednich węzłów i porównywanie rzeczywistego opóźnienia z teoretycznym.

2.2.1. Połączenie Berlin - Warszawa

Połączenie między tymi miastami jest bezpośrednie, a odległość w lini prostej wynosi około 500 kilometrów, co w naszym modelu skutkuje 4 [ms] opóźnienia, tak więc odpowiedź na wysłanego pinga powinniśmy otrzymać po ok 8 [ms]. Jak widać na zamieszczonym poniżej screenie rzeczywiste opóźnienia są poprawne i niewiele odbiegają od zakładanej wartości.

```
mininet> Berlin ping Warszawa -c4
PING 10.0.0.8 (10.0.0.8) 56(84) bytes of data.
64 bytes from 10.0.0.8: icmp_seq=1 ttl=64 time=8.83 ms
64 bytes from 10.0.0.8: icmp_seq=2 ttl=64 time=9.27 ms
64 bytes from 10.0.0.8: icmp_seq=3 ttl=64 time=8.20 ms
64 bytes from 10.0.0.8: icmp_seq=4 ttl=64 time=8.88 ms

--- 10.0.0.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3033ms
rtt min/avg/max/mdev = 8.199/8.794/9.267/0.383 ms
```

Rys. 5: Rzeczywiste opóźnienia na połączeniu Berlin - Warszawa

2.2.2. Połączenie Berlin - Rzym

To połączenie nie jest bezpośrednie i przechodzi ono przez dwa inne miasta Paryż oraz Zurych. Dlatego też teoretyczna wartość opóźnienia wynosi 28 [ms]. Rzeczywiste opóźnienie wynosi około 32 [ms], co związane jest z dodatkowym czasem na przekierowywanie pakietu w switchach występujących na trasie. Bardzo ważnym mechanizmem, który występuje w protokole ICMP, a jeszcze nie został pokazany w naszym sprawozdaniu jest znaczne opóźnienie pierwszego pakietu, które związane jest z ustalaniem drogi do wyznaczonego hosta.

```
mininet> Berlin ping Rzym -c4
PING 10.0.0.7 (10.0.0.7) 56(84) bytes of data.
64 bytes from 10.0.0.7: icmp_seq=1 ttl=64 time=31.9 ms
64 bytes from 10.0.0.7: icmp_seq=2 ttl=64 time=33.5 ms
64 bytes from 10.0.0.7: icmp_seq=3 ttl=64 time=32.3 ms
64 bytes from 10.0.0.7: icmp_seq=4 ttl=64 time=30.6 ms

--- 10.0.0.7 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3054ms
rtt min/avg/max/mdev = 30.642/32.092/33.527/1.027 ms
```

Rys. 6: Rzeczywiste opóźnienia na połączeniu Berlin - Rzym

By lepiej zobrazować wpływ parametrów opóźnienia na czas przesyłu pakietów postanowiliśmy zwiększyć opóźnienie na drodze z Paryża do Zurychu z 3 [ms] na 30 [ms]. Co rzeczywiście zwiększyło czas o około 60 [ms]

```
mininet> Berlin ping Rzym -c4
PING 10.0.0.7 (10.0.0.7) 56(84) bytes of data.
64 bytes from 10.0.0.7: icmp_seq=1 ttl=64 time=95.6 ms
64 bytes from 10.0.0.7: icmp_seq=2 ttl=64 time=93.3 ms
64 bytes from 10.0.0.7: icmp_seq=3 ttl=64 time=98.1 ms
64 bytes from 10.0.0.7: icmp_seq=4 ttl=64 time=90.2 ms

--- 10.0.0.7 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3097ms
rtt min/avg/max/mdev = 90.180/94.287/98.082/2.914 ms
```

Rys. 7: Wpływ zmiany parametrów sieci na opóźnienie połączenia Berlin - Rzym

2.3. Zadania dodatkowe

By wykonać to zadanie najpierw przy użyciu komendy *Paryz xterm* oraz *Ateny xterm* dostaliśmy dostęp do wykonywania komend na obu tych hostach. Następnie przy użyciu python3 postawiliśmy w Paryżu prosty serwer http i dzięki aplikacji wget nawiązaliśmy z nim połączenie z Aten, w wyniku którego pobraliśmy plik "index.html". Całość przedstawiona na poniższym screenie.

Rys. 8: Działanie serwera HTTP

3. Część druga

Część druga obejmowała szereg testów o różnej charakterystyce przeprowadzanych przy użyciu narzędzia *iperf*. Głównym celem było zbadanie "rywalizacji" pomiędzy strumieniami danych o ograniczone zasoby sieci, testy przeprowadzaliśmy z różnymi typami strumieni oraz w różnych konfiguracjach. Do każdego testu dołączona jest tabelka z jego konfiguracją.

Co ważne, gdy inicjalizujemy strumień UDP to określamy prędkość z którą host wysyła dane na serwer (jako procent dostępnej przepustowości określonej w Mb/s) oraz czas trwania połączenia. Natomiast w przypadku połączenia TCP określamy rozmiar pliku, który chcemy przesłać, wynika to z zmieniającej się w czasie prędkości przesyłu zależnej od warunków i przepustowości sieci.

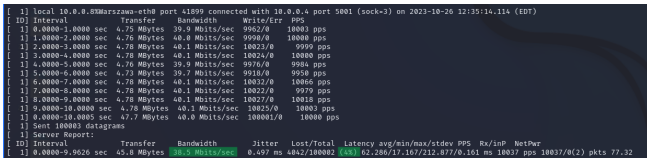
3.1. UDP 50%

Jako pierwszy test sprawdziliśmy zachowanie strumienia UDP w konfiguracji, która nie przekraczała dostępnych zasobów.

Nr.	Połączenie 1
Węzły sieci	Warszawa - Madryt
Typ strumienia	UDP
Wykorzystane zasoby	50%

```
Madryt iperf -s -u -e > MadrytServer &
Warszawa iperf -c Madryt -u -i 1 -e -b 10000pps -l 500 >  WarszawaHost &
```

Rys. 9: Skrypt testowy



Rys. 10: Wynik testu

Oczywiście w tej konfiguracji proces transferu danych przeszedł pomyślnie, packet loss wyniósł 4%, co wynika z natury sieci.

3.2. UDP 100%

Ten test jest małą modyfikacją testu 1 i bada on zachowanie strumienia UDP, gdy jesteśmy na granicy wyczerpania zasobów sieciowych.

Nr.	Połączenie 1
Węzły sieci	Warszawa - Madryt
Typ strumienia	UDP
Wykorzystane zasoby	100%

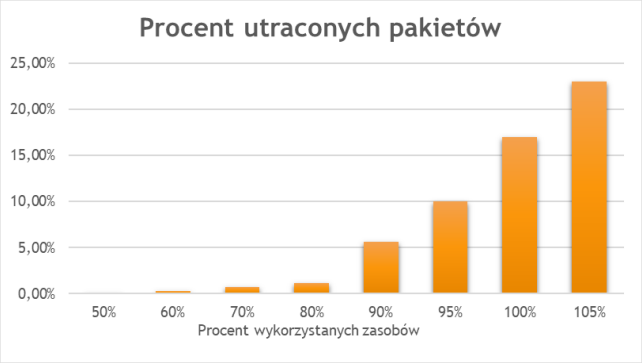
```
Madryt iperf -s -u -e > MadrytServer &
Warszawa iperf -c Madryt -u -i 1 -e -b 10000pps -l 1000 > WarszawaHost &
```

Rys. 11: Skrypt testowy

```
[  1] local 10.0.0.8%Warszawa-eth0 port 50444 connected with 10.0.0.4 port 5001 (sock-3) on 2023-10-26 12:42:22.317 (EDT)
[ ID] Interval       Transfer     Bandwidth       Write/Err  PPS
[  1] 0.0000-1.0000 sec  9.53 MBytes  79.9 Mbits/sec  9988/0     9992 pps
[  1] 1.0000-2.0000 sec  9.54 MBytes  80.0 Mbits/sec  10000/0    10010 pps
[  1] 2.0000-3.0000 sec  9.55 MBytes  80.1 Mbits/sec  10011/0    10000 pps
[  1] 3.0000-4.0000 sec  9.53 MBytes  80.0 Mbits/sec  9998/0    10008 pps
[  1] 4.0000-5.0000 sec  9.54 MBytes  80.0 Mbits/sec  10001/0    10000 pps
[  1] 5.0000-6.0000 sec  9.54 MBytes  80.0 Mbits/sec  10002/0    10001 pps
[  1] 6.0000-7.0000 sec  9.53 MBytes  80.0 Mbits/sec  9995/0     9996 pps
[  1] 7.0000-8.0000 sec  9.54 MBytes  80.0 Mbits/sec  10006/0    10004 pps
[  1] 8.0000-9.0000 sec  9.53 MBytes  80.0 Mbits/sec  9997/0     9997 pps
[  1] 9.0000-10.0000 sec  9.52 MBytes  79.9 Mbits/sec  9987/0     9993 pps
[  1] 0.0000-10.0012 sec  95.4 MBytes  80.0 Mbits/sec  99987/0    9998 pps
[  1] Sent 99989 datagrams
[  1] Server Report:
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total  Latency avg/min/max/stdev PPS  Rx/inP  NetPwr
[  1] 0.0000-10.1653 sec  77.4 MBytes  63.9 Mbits/sec  0.189 ms 10831/99988 (19%) 159.230/20.069/237.135/8.220 ms 9836 pps 9836/0(2) pkts 50.14
```

Rys. 12: Wynik testu

Jak widać w tej sytuacji sieć miał już znacznie większe problemy w obsłużeniu strumienia i prawie 1/5 ze wszystkich pakietów została utracona (19%). Co oczywiście wynika z przeciążenia sieci, warto zwrócić uwagę, że problemy z odebraniem wszystkich pakietów zaczynają się pojawiać około 85% i gwałtownie rosną w miarę zbliżania się do maksymalnej przepustowości sieci. Co na pewno trzeba mieć na uwadze podczas planowania własnej sieci. Opisaną zależność co pokazuje poniższy wykres.



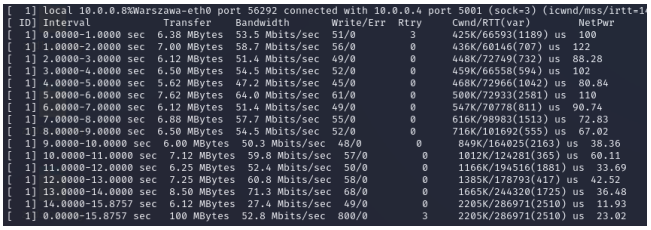
Rys. 13: Procent utraconych danych od obciążenia sieci

3.3. TCP

Nr.	Połączenie 1
Węzły sieci	Warszawa - Madryt
Typ strumienia	TCP
Rozmiar pliku	100 MB

```
Madryt iperf -s -e -i 1 > MadrytServer &
Warszawa iperf -c Madryt -i 1 -e -n 100M > WarszawaHost &
```

Rys. 14: Skrypt testowy



Rys. 15: Wynik testu dla jednego połączenia TCP

Ten prosty test pokazał nam, że mimo szerokości łącza wynoszącej 80 Mb/s, średnia przepustowość jakiegokolwiek połączenia będzie o kilkanaście procent niższa. Jest to spowodowane zasadą działania algorytmu *congestion control*, który mimo braku znajomości maksymalnej szerokości łącza, stara się zachować najlepszą średnią prędkość połączenia.

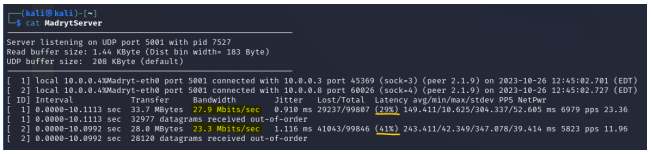
3.4. UDP 50% + UDP 50%

W teście tym przesyłamy dane strumieniem UDP z dwóch źródeł jednocześnie, tak że każdy z nich wykorzystuje po 50% dostępnych zasobów.

Nr.	Połączenie 1	Połączenie 2
Węzły sieci	Warszawa - Madryt	Londyn - Madryt
Typ strumienia	UDP	UDP
Wykorzystane zasoby	50%	50%

```
Madryt iperf -s -u -e > MadrytServer &
Warszawa iperf -c Madryt -u -i 1 -e -b 10000pps -l 500 > WarszawaHost &
Londyn iperf -c Madryt -u -i 1 -e -b 10000pps -l 500 > LondynHost &
```

Rys. 16: Skrypt testowy



Rys. 17: Wynik testu dla Serwera

Czytając raport wygenerowany przez serwer z łatwością możemy zauważyć proces rywalizacji o zasoby zachodzący między dwoma strumieniami UDP. Jego pochodną jest ilość straconych pakietów. Ponieważ oba strumienie spotykały się w Paryżu to Lodnyn z racji znacznie mniejszej odległości, a co za tym idzie opóźnieniom, był w stanie lepiej zagospodarować zasoby niż oddalona Warszawa.

Co warte podkreślenia to fakt iż oba te strumienie w teorii zajmujące 100% dostępnych zasobów na trasie z Paryża do Madrytu w sumie straciły około 35% pakietów co jest znacznie gorszym wynikiem od jednego strumienia wykorzystującego tą samą ilość zasobów na trasie z Warszawy do Madrytu.

3.5. UDP 100% + UDP 100%

Test ten jest modyfikacją poprzedniego testu, ustawiając szybkość wysyłania pakietów z Warszawy na 80 Mb/s, świadomie przekraczamy maksymalną przepustowość sieci pomiędzy węzłami Paryż-Madryt.

Nr.	Połączenie 1	Połączenie 2
Węzły sieci	Warszawa - Madryt	Londyn - Madryt
Typ strumienia	UDP	UDP
Wykorzystane zasoby	100%	50%

```
Madryt iperf -s -u -e > MadrytServer &
Warszawa iperf -c Madryt -u -i 1 -e -b 10000pps -l 1000 > WarszawaHost &
Londyn iperf -c Madryt -u -i 1 -e -b 10000pps -l 500 > LondynHost &
```

Rys. 18: Skrypt testowy

```
┌──(kali㉿kali)-[~]
└─$ cat MadrytServer
------------------------------------------------------------
Server listening on UDP port 5001 with pid 10911
Read buffer size: 1.44 KByte (Dist bin width= 183 Byte)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  1] local 10.0.0.4%Madryt-eth0 port 5001 connected with 10.0.0.3 port 43734 (sock=3) (peer 2.1.9) on 2023-10-26 12:54:36.351 (EDT)
[  2] local 10.0.0.4%Madryt-eth0 port 5001 connected with 10.0.0.8 port 42740 (sock=4) (peer 2.1.9) on 2023-10-26 12:54:36.381 (EDT)
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total  Latency avg/min/max/stdev PPS NetPwr
[  1] 0.0000-10.0517 sec  28.1 MBytes  23.5 Mbits/sec   0.237 ms 41013/100003 (41%) 95.934/32.170/160.245/11.864 ms 5869 pps 30.59
[  1] 0.0000-10.0517 sec  3175 datagrams received out-of-order
[  2] 0.0000-10.1284 sec  47.7 MBytes  39.5 Mbits/sec   0.357 ms 49989/100000 (50%) 257.033/38.665/345.230/27.745 ms 4938 pps 19.21
[  2] 0.0000-10.1284 sec  3180 datagrams received out-of-order
```

Rys. 19: Wynik testu dla Serwera

Zapoznając się z raportem widzimy, że zalanie sieci pakietami z Warszawy spowodowało znaczy spadek efektywności połączenia zarówno na trasie Warszawa-Madryt (wcześniej 41% teraz 50%) jak i Londyn-Madryt (wcześniej 29% teraz 41%). Co oznacza, że wzmożony ruch sieciowy ma wpływ na jakość połączwenia wszystkich hostów niezależnie od tego jak bardzo wykorzystują oni jej zasoby.

3.6. UDP 25% + UDP 25% + UDP 25%

Test ukazujący wpływ wielu połączeń utrzymywanych przez jeden serwer na efektywność i jakość każdego z nich.

Nr.	Połączenie 1	Połączenie 2	Połączenie 3
Węzły sieci	Warszawa - Madryt	Londyn -Madryt	Ateny-Madryt
Typ strumienia	UDP	UDP	UDP
Wykorzystane zasoby	25%	25%	25%

```
1 Madryt iperf -s -u -e > MadrytServer &
2 Warszawa iperf -c Madryt -u -i 1 -e -b 10000pps -l 250 &
3 Londyn iperf -c Madryt -u -i 1 -e -b 10000pps -l 250 &
4 Ateny iperf -c Madryt -u -i 1 -e -b 10000pps -l 250 &
```

Rys. 20: Skrypt testowy

```
0neone@s0neone-VirtualBox:~/sinthector/sintest$ cat MadrytServer
------------------------------------------------------------
Server listening on UDP port 5001 with pid 2658
Read buffer size: 1.44 KByte (Dist bin width= 183 Byte)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  1] local 10.0.0.4%Madryt-eth0 port 5001 connected with 10.0.0.3 port 53960 (sock=3) (peer 2.1.5) on 2023-10-28 21:48:14 (CEST)
[  2] local 10.0.0.4%Madryt-eth0 port 5001 connected with 10.0.0.8 port 60002 (sock=4) (peer 2.1.5) on 2023-10-28 21:48:14 (CEST)
[  3] local 10.0.0.4%Madryt-eth0 port 5001 connected with 10.0.0.1 port 39976 (sock=5) (peer 2.1.5) on 2023-10-28 21:48:14 (CEST)
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total  Latency avg/min/max/stdev PPS NetPwr
[  1] 0.0000-10.0142 sec  22.3 MBytes  18.7 Mbits/sec   0.065 ms 6486/100003 (6.5%) 30.681/9.023/72.613/11.317 ms 9338 pps 76
[  1] 0.0000-10.0142 sec  3373 datagrams received out-of-order
[  3] 0.0000-9.9361 sec  22.0 MBytes  18.6 Mbits/sec   0.149 ms 7511/99994 (7.5%) 44.425/22.070/86.198/11.423 ms 9308 pps 52
[  3] 0.0000-9.9361 sec  3911 datagrams received out-of-order
[  2] 0.0000-9.9805 sec  22.0 MBytes  18.5 Mbits/sec   0.091 ms 7692/100004 (7.7%) 38.951/17.037/81.093/11.351 ms 9249 pps 59
[  2] 0.0000-9.9805 sec  3325 datagrams received out-of-order
```

Rys. 21: Wynik testu dla Serwera

Jak widać pomimo tego iż jesteśmy daleko od wykorzystania pełnej przepustowości łącza (75% na odcinku Paryz-Madryt) to samo uruchomienie trzech strumieni wpływa na zwiększoną utratę pakietów każdego z nich. Dla jednego strumienia UDP zajmującego 80% przepustowości strata wynosi lekko powyżej 1%

3.7. TCP + TCP

Nr.	Połączenie 1	Połączenie 2
Węzły sieci	Warszawa - Madryt	Londyn - Madryt
Typ strumienia	TCP	TCP
Rozmiar pliku	500 MB	500 MB

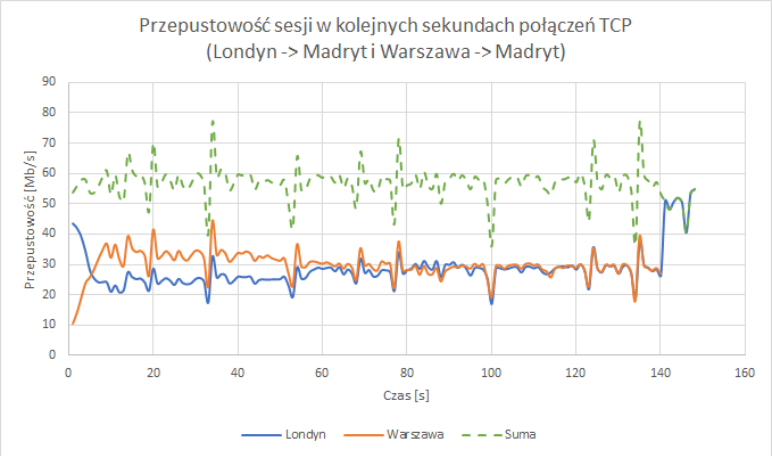
```
Madryt iperf -s -e -i 1 > MadrytServer &
Warszawa iperf -c Madryt -i 1 -e -n 500M > WarszawaHost &
Londyn iperf -c Madryt -i 1 -e -n 500M > LondynHost &
```

Rys. 22: Skrypt testowy

```
[  1] local 10.0.0.4%Madryt-eth0 port 5001 connected with 10.0.0.3 port 56446 (sock=4) (peer 2.1.9)
[  2] local 10.0.0.4%Madryt-eth0 port 5001 connected with 10.0.0.8 port 48872 (sock=5) (peer 2.1.9)
[ ID] Interval       Transfer     Bandwidth       Reads=Dist
[  1] 0.0000-1.0000 sec  5.19 MBytes  43.5 Mbits/sec  760=735:20:5:0:0:0:0:0
[  2] 0.0000-1.0000 sec  1.22 MBytes  10.3 Mbits/sec  410=410:0:0:0:0:0:0:0
[  1] 1.0000-2.0000 sec  4.99 MBytes  41.9 Mbits/sec  930=929:1:0:0:0:0:0:0
[  2] 1.0000-2.0000 sec  1.68 MBytes  14.1 Mbits/sec  568=568:0:0:0:0:0:0:0
[  1] 2.0000-3.0000 sec  4.65 MBytes  39.0 Mbits/sec  890=889:1:0:0:0:0:0:0
[  2] 2.0000-3.0000 sec  2.27 MBytes  19.0 Mbits/sec  736=736:0:0:0:0:0:0:0
[  1] 3.0000-4.0000 sec  4.08 MBytes  34.2 Mbits/sec  879=878:1:0:0:0:0:0:0
[  2] 3.0000-4.0000 sec  2.84 MBytes  23.8 Mbits/sec  873=873:0:0:0:0:0:0:0
[  1] 4.0000-5.0000 sec  3.35 MBytes  28.1 Mbits/sec  836=836:0:0:0:0:0:0:0
[  2] 4.0000-5.0000 sec  3.07 MBytes  25.7 Mbits/sec  887=884:3:0:0:0:0:0:0
[  1] 5.0000-6.0000 sec  3.03 MBytes  25.4 Mbits/sec  874=874:0:0:0:0:0:0:0
```

Rys. 23: Wynik testu dla Serwera

W tym teście jesteśmy w stanie zaobserwować mechanizm *congestion control* na dwóch połączeniach TCP. Przesyłając małe pliki, średnia przepustowość sieci będzie wyższa dla hosta znajdującego się geograficznie bliżej serwera, ponieważ to on jako pierwszy zawiążę połączenie. Jednak w przypadku większych plików i dłuższego czasu połączenia, przepustowości te wyrównają się (żółty kolor podkreślenia na zrzucie ekranu), co pokazuje wykres nr 24



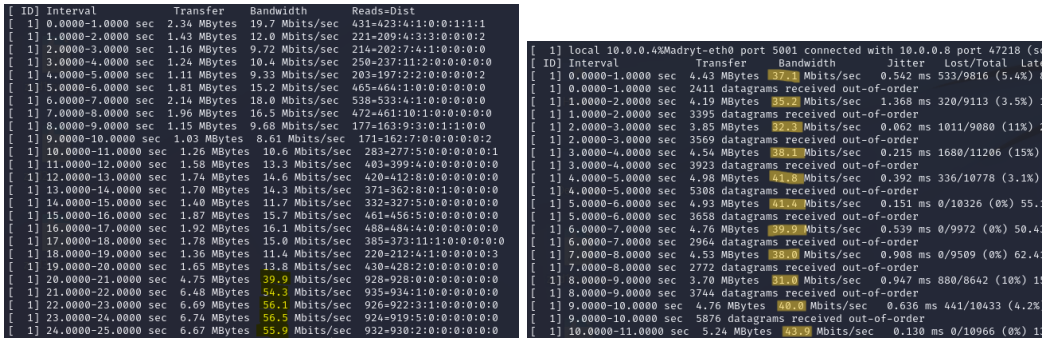
Rys. 24: Wykres przepustowości dwóch symetrycznych połączeń TCP

3.8. UDP 50% + TCP

Nr.	Połączenie 1	Połączenie 2
Węzły sieci	Warszawa - Madryt	Londyn - Madryt
Typ strumienia	UDP	TCP
Wykorzystane zasoby / Rozmiar pliku	50%	200 MB

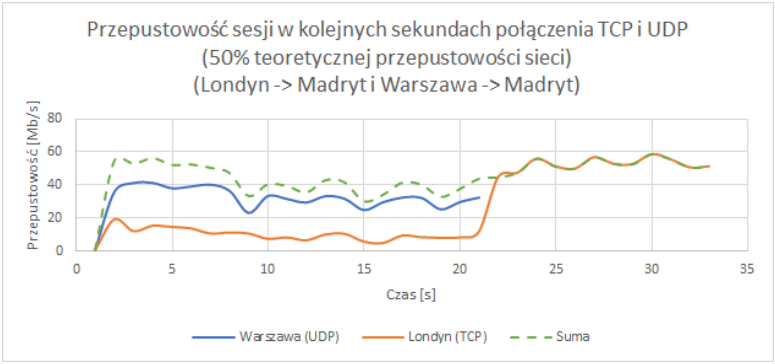
```
Madryt iperf -s -e -i 1 > MadrytServerTCP &
Madryt iperf -s -u -e -i 1 > MadrytServerUDP &
Warszawa iperf -c Madryt -u -i 1 -e -b 10000pps -l 500 -t 20 > WarszawaHost &
Londyn iperf -c Madryt -i 1 -e -n 200M > LondynHost &
```

Rys. 25: Skrypt testowy



Rys. 26: Wynik testu dla Serwera TCP i UDP

Ten test pokazał zależność między przepustowościami w przypadku rywalizacji połączenia TCP i UDP na jednym łączu. Jak widać na wykresie 27, to połączenie TCP dostosowuje swoją przepustowość zmniejszając szerokość swojego okna tak, aby razem ze strumieniem protokołu UDP nie przekroczyć całej pojemności łącza. Dopiero po zakończeniu strumienia UDP, przepustowość połączenia TCP znacznie się zwiększyła, pozwalając na przesłanie reszty pliku.



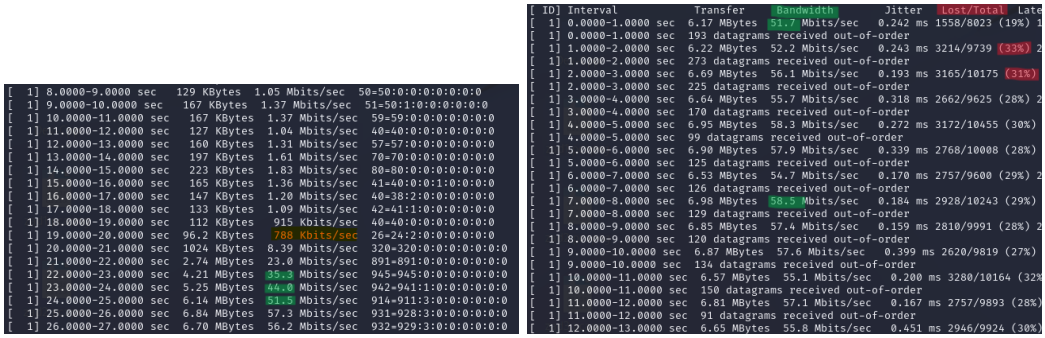
Rys. 27: Rywalizacja połączeń TCP i UDP

3.9. UDP 100% + TCP

Nr.	Połączenie 1	Połączenie 2
Węzły sieci	Warszawa - Madryt	Londyn - Madryt
Typ strumienia	UDP	TCP
Wykorzystane zasoby / Rozmiar pliku	100%	200 MB

```
Madryt iperf -s -e -i 1 > MadrytServerTCP &
Madryt iperf -s -u -e -i 1 > MadrytServerUDP &
Warszawa iperf -c Madryt -u -i 1 -e -b 10000pps -l 1000 -t 20 > WarszawaHost &
Londyn iperf -c Madryt -i 1 -e -n 200M > LondynHost &
```

Rys. 28: Skrypt testowy

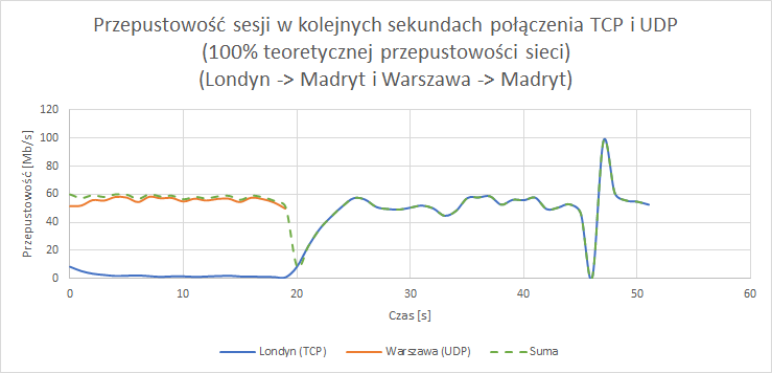


```
[  1] 8.0000-9.0000 sec   129 KBytes  1.05 Mbits/sec  50=50:0:0:0:0:0:0:0
[  1] 9.0000-10.0000 sec   167 KBytes  1.37 Mbits/sec  51=50:1:0:0:0:0:0:0
[  1] 10.0000-11.0000 sec   167 KBytes  1.37 Mbits/sec  59=59:0:0:0:0:0:0:0
[  1] 11.0000-12.0000 sec   127 KBytes  1.04 Mbits/sec  40=40:0:0:0:0:0:0:0
[  1] 12.0000-13.0000 sec   160 KBytes  1.31 Mbits/sec  57=57:0:0:0:0:0:0:0
[  1] 13.0000-14.0000 sec   197 KBytes  1.61 Mbits/sec  70=70:0:0:0:0:0:0:0
[  1] 14.0000-15.0000 sec   223 KBytes  1.83 Mbits/sec  80=80:0:0:0:0:0:0:0
[  1] 15.0000-16.0000 sec   165 KBytes  1.36 Mbits/sec  41=40:0:0:1:0:0:0:0
[  1] 16.0000-17.0000 sec   147 KBytes  1.20 Mbits/sec  40=38:2:0:0:0:0:0:0
[  1] 17.0000-18.0000 sec   133 KBytes  1.09 Mbits/sec  42=41:1:0:0:0:0:0:0
[  1] 18.0000-19.0000 sec   112 KBytes   915 Kbits/sec  40=40:0:0:0:0:0:0:0
[  1] 19.0000-20.0000 sec  96.2 KBytes   788 Kbits/sec  26=24:2:0:0:0:0:0:0
[  1] 20.0000-21.0000 sec  1024 KBytes  8.39 Mbits/sec  320=320:0:0:0:0:0:0:0
[  1] 21.0000-22.0000 sec  2.74 MBytes  23.0 Mbits/sec  891=891:0:0:0:0:0:0:0
[  1] 22.0000-23.0000 sec  4.21 MBytes  35.3 Mbits/sec  945=945:0:0:0:0:0:0:0
[  1] 23.0000-24.0000 sec  5.25 MBytes  44.0 Mbits/sec  942=941:1:0:0:0:0:0:0
[  1] 24.0000-25.0000 sec  6.14 MBytes  51.5 Mbits/sec  914=911:3:0:0:0:0:0:0
[  1] 25.0000-26.0000 sec  6.84 MBytes  57.3 Mbits/sec  931=928:3:0:0:0:0:0:0
[  1] 26.0000-27.0000 sec  6.70 MBytes  56.2 Mbits/sec  932=929:3:0:0:0:0:0:0
```

```
[ ID] Interval           Transfer     Bandwidth       Jitter   Lost/Total  Late
[  1] 0.0000-1.0000 sec  6.17 MBytes  51.7 Mbits/sec   0.242 ms 1558/8023 (19%) 1
[  1] 0.0000-1.0000 sec  193 datagrams received out-of-order
[  1] 1.0000-2.0000 sec  6.22 MBytes  52.2 Mbits/sec   0.243 ms 3214/9739 (33%) 2
[  1] 1.0000-2.0000 sec  273 datagrams received out-of-order
[  1] 2.0000-3.0000 sec  6.69 MBytes  56.1 Mbits/sec   0.193 ms 3165/10175 (31%)
[  1] 2.0000-3.0000 sec  225 datagrams received out-of-order
[  1] 3.0000-4.0000 sec  6.64 MBytes  55.7 Mbits/sec   0.318 ms 2662/9625 (28%) 2
[  1] 3.0000-4.0000 sec  170 datagrams received out-of-order
[  1] 4.0000-5.0000 sec  6.95 MBytes  58.3 Mbits/sec   0.272 ms 3172/10455 (30%)
[  1] 4.0000-5.0000 sec  99 datagrams received out-of-order
[  1] 5.0000-6.0000 sec  6.90 MBytes  57.9 Mbits/sec   0.339 ms 2768/10008 (28%)
[  1] 5.0000-6.0000 sec  125 datagrams received out-of-order
[  1] 6.0000-7.0000 sec  6.53 MBytes  54.7 Mbits/sec   0.170 ms 2757/9600 (29%) 2
[  1] 6.0000-7.0000 sec  126 datagrams received out-of-order
[  1] 7.0000-8.0000 sec  6.98 MBytes  58.5 Mbits/sec   0.184 ms 2928/10243 (29%)
[  1] 7.0000-8.0000 sec  129 datagrams received out-of-order
[  1] 8.0000-9.0000 sec  6.85 MBytes  57.4 Mbits/sec   0.159 ms 2810/9991 (28%) 2
[  1] 8.0000-9.0000 sec  120 datagrams received out-of-order
[  1] 9.0000-10.0000 sec  6.87 MBytes  57.6 Mbits/sec   0.399 ms 2620/9819 (27%)
[  1] 9.0000-10.0000 sec  134 datagrams received out-of-order
[  1] 10.0000-11.0000 sec  6.57 MBytes  55.1 Mbits/sec   0.200 ms 3280/10164 (32%
[  1] 10.0000-11.0000 sec  150 datagrams received out-of-order
[  1] 11.0000-12.0000 sec  6.81 MBytes  57.1 Mbits/sec   0.167 ms 2757/9893 (28%)
[  1] 11.0000-12.0000 sec  91 datagrams received out-of-order
[  1] 12.0000-13.0000 sec  6.65 MBytes  55.8 Mbits/sec   0.451 ms 2946/9924 (30%)
```

Rys. 29: Wynik testu dla Serwera TCP oraz UDP

Ta zmodyfikowana wersja poprzedniego testu, utwierdziła nas w przekonaniu, że to połączenie TCP dostosowuje swoje okno do aktualnej zajętości łącza. W tym przypadku konkurencyjne łącze UDP w teorii zajmuje 100% łącza, ale przez utracone pakiety faktyczna zajętość łącza jest minimalnie mniejsza, co pozwala połączeniu TCP przesyłać dane z prędkością momentami nie przekraczającą nawet 1% maksymalnej zajętości łącza (poniżej 1 Mb/s). Skok tej wartości nastąpił dopiero po zakończeniu strumienia UDP (wykres poniżej)



Rys. 30: Rywalizacja połączeń TCP i UDP

4. Podsumowanie

To laboratorium w praktyce pokazało nam, jak zachowują się protokoły TCP i UDP w przypadku ograniczonych zasobów łącza i w zależności od ich parametrów. Poznaliśmy emulator sieci Mininet, który dzięki prostej zasadzie działania i możliwości tworzenia skryptów, pozwala badać zachowanie sieci. Razem z narzędziem Iperf stanowią dobrą bazę do nauki i empirycznego zbadania jak działa sieć. Najbardziej zaskakujące okazało się zachowanie połączenia dwóch protokołów w przypadku ograniczonej pojemności sieci - okazuje się, że UDP ma pewien "priorytet" nad TCP, wynikający z mechanizmu *congestion control*. Przykładowo oznacza to, że gdy jeden z hostów streamuje treści z Internetu (protokołem UDP), a drugi próbuje pobrać jakiś plik (używając TCP), przy wysokim obciążeniu sieci nie będą obserwować takich samych efektów.