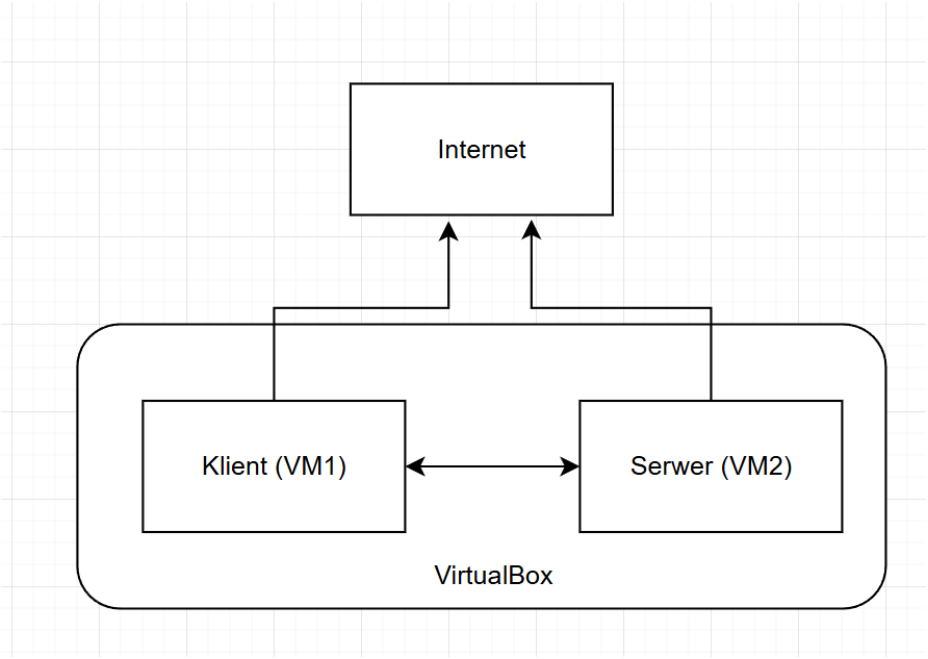
Sprawozdanie, W6.2 – TCP i UDP, netem

Uczestnicy zespołu: Tymon Zadara, Kinga Konieczna, Jan Czechowski

# Zadanie 0 – Adresacja środowiska

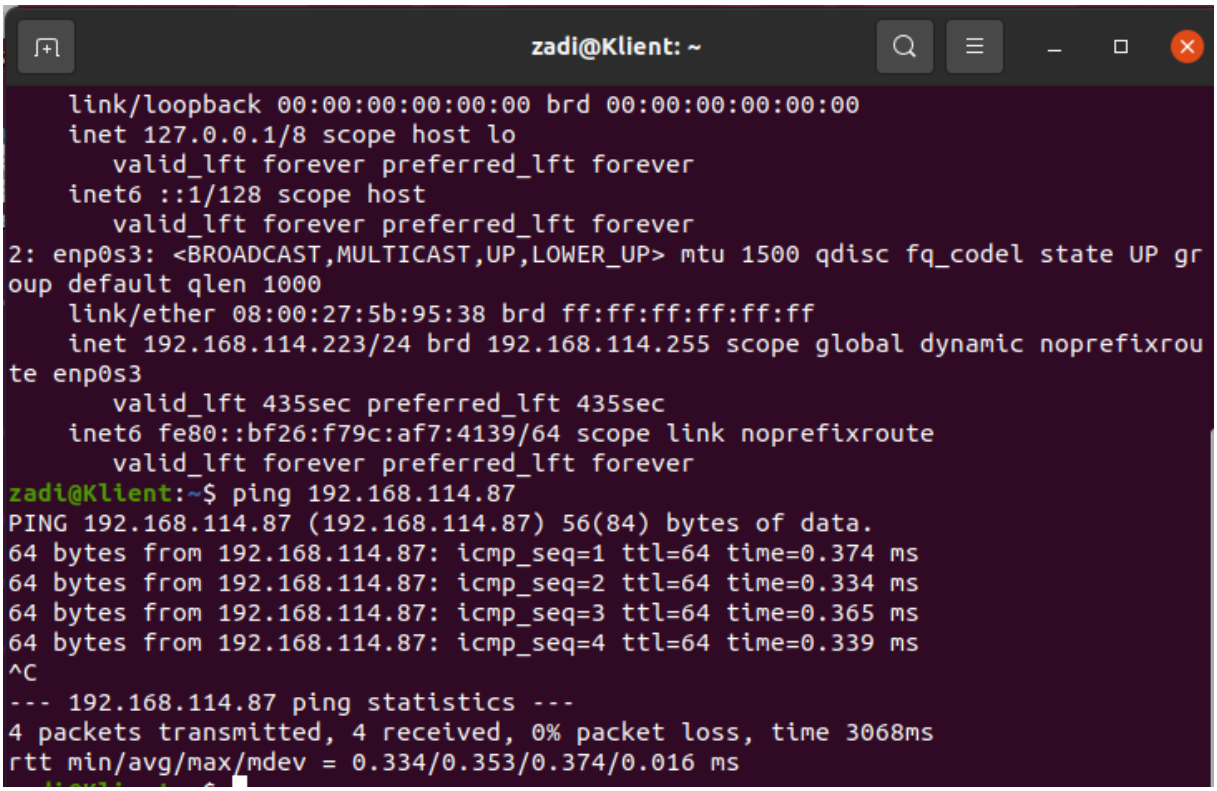
Ćwiczenia należy wykonywać w środowisku zwirtualizowanym, przygotowanym na poprzednich warsztatach.

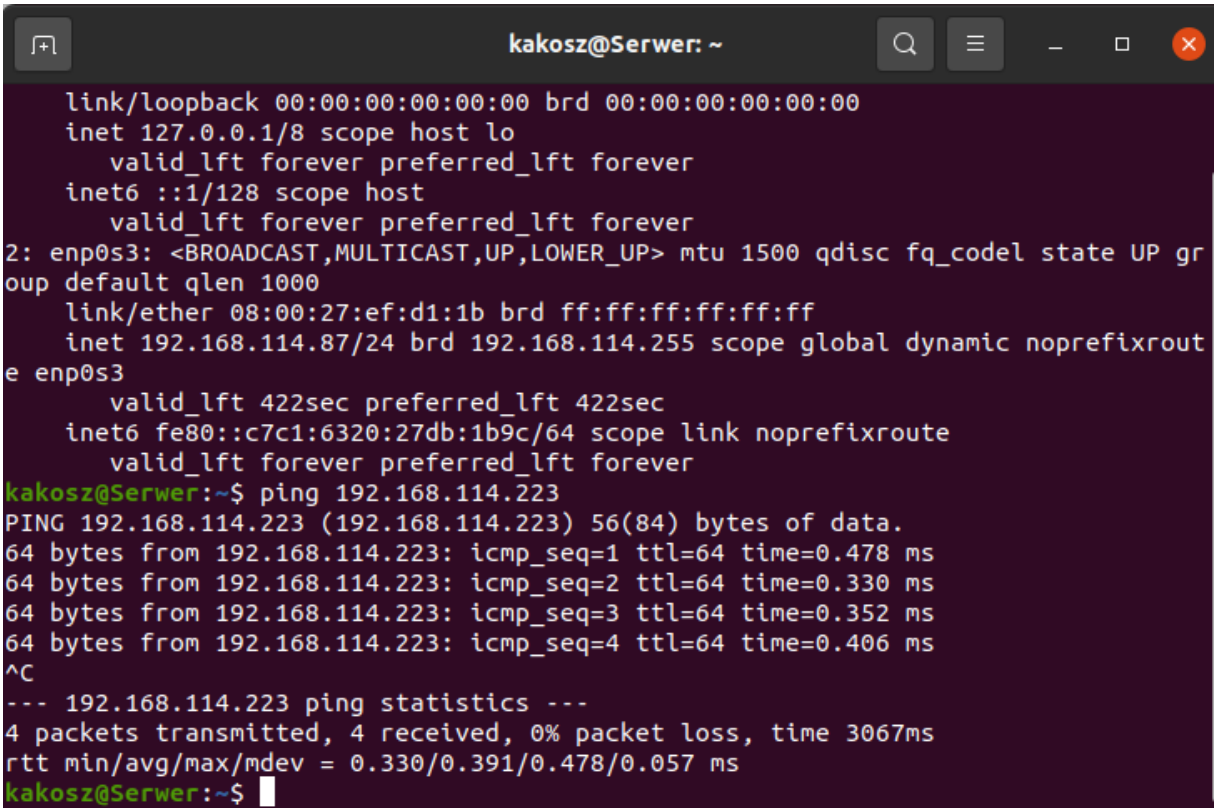
Zalecane jest robienie zadań w trybie Nat network (tutorial w poprzedniej instrukcji), zasada działania jest analogiczna.




Obydwie maszyny muszą zarówno mieć kontakt ze sobą nawzajem, oraz z siecią Internet (np. ping 8.8.8.8). Prawidłową wstępną weryfikację można sprawdzić za pomocą polecenia *ping*.

**Zrzuty ekranu z ping między maszynami (2 zrzuty ekranu):**


zadi@Klient: ~
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP gr
oup default qlen 1000
link/ether 08:00:27:5b:95:38 brd ff:ff:ff:ff:ff:ff
inet 192.168.114.223/24 brd 192.168.114.255 scope global dynamic noprefixrou
te enp0s3
valid_lft 435sec preferred_lft 435sec
inet6 fe80::bf26:f79c:af7:4139/64 scope link noprefixroute
valid_lft forever preferred_lft forever
zadi@Klient:~$ ping 192.168.114.87
PING 192.168.114.87 (192.168.114.87) 56(84) bytes of data.
64 bytes from 192.168.114.87: icmp_seq=1 ttl=64 time=0.374 ms
64 bytes from 192.168.114.87: icmp_seq=2 ttl=64 time=0.334 ms
64 bytes from 192.168.114.87: icmp_seq=3 ttl=64 time=0.365 ms
64 bytes from 192.168.114.87: icmp_seq=4 ttl=64 time=0.339 ms
^C
--- 192.168.114.87 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3068ms
rtt min/avg/max/mdev = 0.334/0.353/0.374/0.016 ms


kakosz@Serwer: ~
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP gr
oup default qlen 1000
link/ether 08:00:27:ef:d1:1b brd ff:ff:ff:ff:ff:ff
inet 192.168.114.87/24 brd 192.168.114.255 scope global dynamic noprefixrout
e enp0s3
valid_lft 422sec preferred_lft 422sec
inet6 fe80::c7c1:6320:27db:1b9c/64 scope link noprefixroute
valid_lft forever preferred_lft forever
kakosz@Serwer:~$ ping 192.168.114.223
PING 192.168.114.223 (192.168.114.223) 56(84) bytes of data.
64 bytes from 192.168.114.223: icmp_seq=1 ttl=64 time=0.478 ms
64 bytes from 192.168.114.223: icmp_seq=2 ttl=64 time=0.330 ms
64 bytes from 192.168.114.223: icmp_seq=3 ttl=64 time=0.352 ms
64 bytes from 192.168.114.223: icmp_seq=4 ttl=64 time=0.406 ms
^C
--- 192.168.114.223 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3067ms
rtt min/avg/max/mdev = 0.330/0.391/0.478/0.057 ms
kakosz@Serwer:~$

# Ćwiczenie 1 – Analiza ruchu sieciowego – TCP (Wireshark, Iperf), zakończenie połączenia

Podpowiedź – żeby nie zapełnić dysku serwera, po stronie klienta możemy korzystać z flag -t 50 (limit 50 sekund) i -b 1K (limit 1Kilobajt/sekundę)

Podpowiedź druga – zwalnianie miejsca na serwerze. Wireshark zapisuje pliki .pcap w /tmp/wireshark_<interfejs>_<data>.

Możemy je zlokalizować:

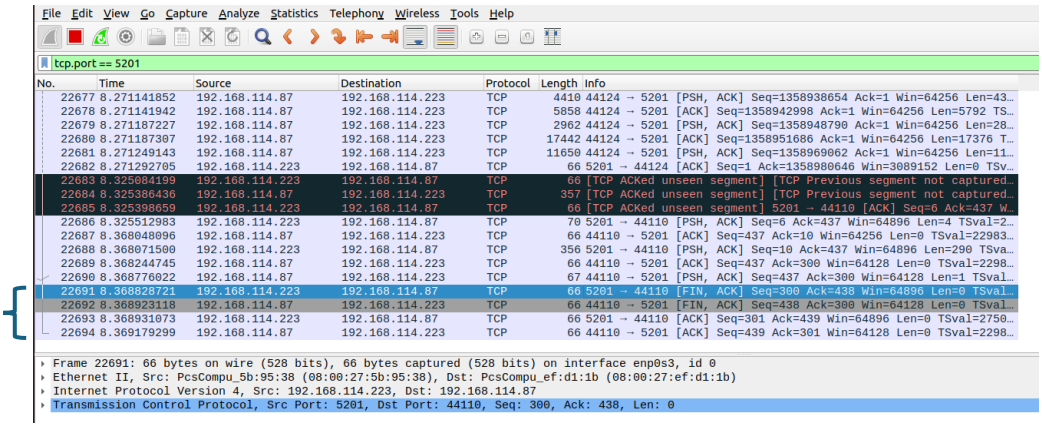
ls /tmp

Oraz usunąć wybrane (**ostrożnie – zawsze usuwamy konkretny folder wireshark**, jak usuniemy całe tmp – niedobrze…):

Rm -rf /tmp/wireshark….

1. Badanie zakończenia sesji:
    - Planowane (FIN):
        - Strona serwer uruchamia serwer iperf i Wireshark
        - Strona klient uruchamia klienta iperf (-t 5, czyli na 5 sekund)
        - Obserwujemy w Wireshark 4 krokowe zakończenie sesji

**Zrzut ekranu z Wireshark – FIN, 4 krokowe zakończenie sesji (wytłumaczcie wiadomość po wiadomości):**




W dzisiejszych warsztatach, maszyna „Klient” (zwana dalej Serwerem) będzie służyła nam jako serwer.

W wyznaczonym przez nas 4 krokowym zakończeniu sesji znajdują się 4 kroki zaznaczone niebieską klamrą. 4 – krokowe zakończenie składa się z następujących kroków:

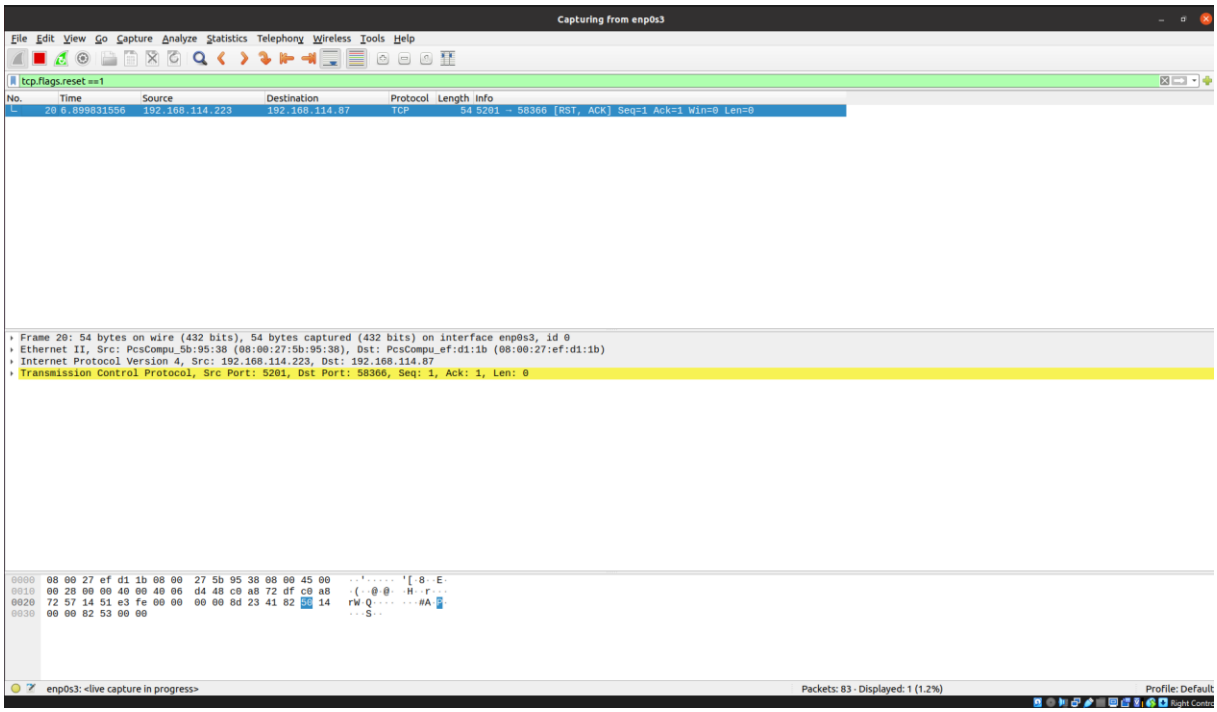
- Pierwszy pakiet [FIN, ACK] – wychodzi z serwera do klienta – Informuje, że serwer zakończył wysyłanie danych i chce zamknąć połączenie.

- Drugi pakiet [FIN, ACK] – od klienta do serwera - w normalnej sekwencji powinien wystąpić tu pakiet [ACK] natomiast tak się nie stało. Dlaczego? Najpewniej klient wysłał swój własny pakiet o fladze [FIN, ACK] w czasie bardzo podobnym do czasu wysłania pakietu przez serwer, zanim jeszcze dostał pakiet wysłany przez serwer. Nastąpiło wtedy jednoczesne zakończenie sesji (jednoczesne, natomiast nic nie jest faktycznie jednoczesne tylko bardzo blisko siebie w czasie).
- Pierwszy pakiet [ACK] – od serwera do klienta – potwierdza otrzymanie pakietu o fladze [FIN] i potwierdza zamknięcie połączenia.
- Drugi pakiet [ACK] – od klienta do serwera – potwierdza otrzymanie pakietu o fladze [FIN] i potwierdza zamknięcie połączenia.

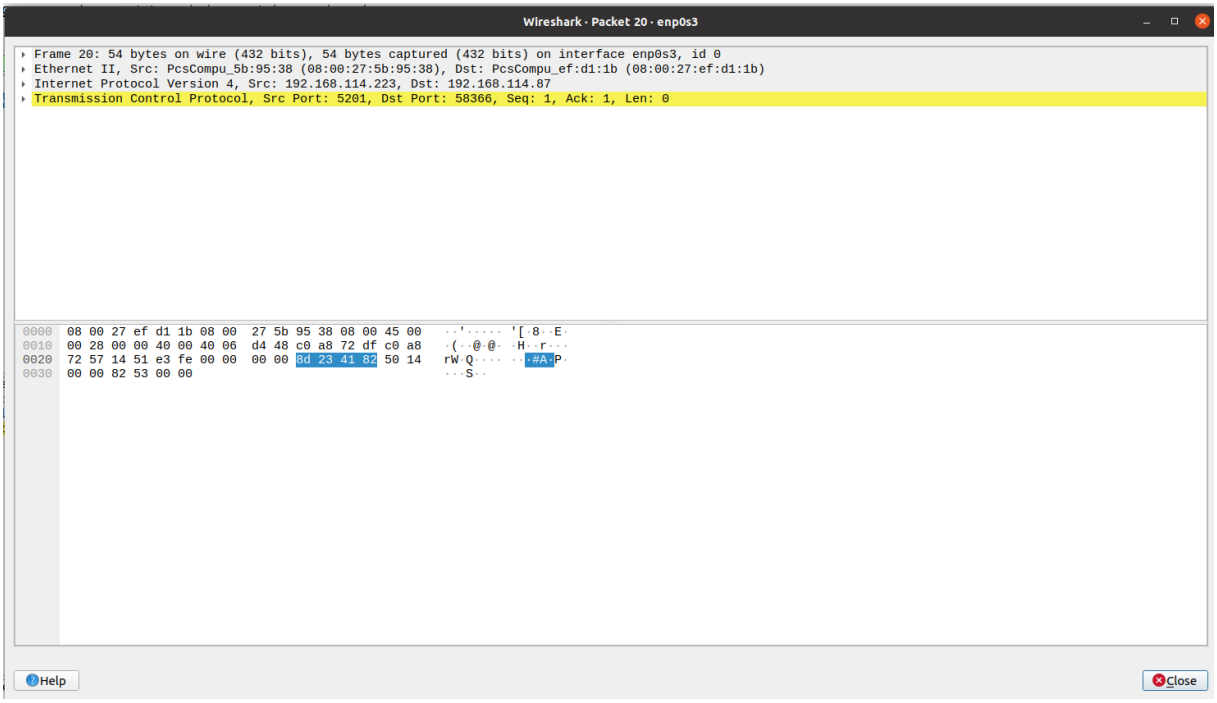
Po wysłaniu pakietu [ACK] zarówno klient, jak i serwer odczekują pewien okres czasu w celu upewnienia się, że druga strona otrzymała pakiet [ACK] i można poprawnie i bezpiecznie zakończyć połączenie.

- Nieplanowane(RST):
  - Wyłączamy proces **serwera** iperf (ctrl + c w terminalu)
  - Włączamy wireshark po stronie serwera
  - Ponownie inicjujemy iperf po stronie klienta na teraz już zamknięty port serwera

**Zrzut ekranu z Wireshark z filtrem wykorzystanym do pokazania tylko pakietów TCP RST:**



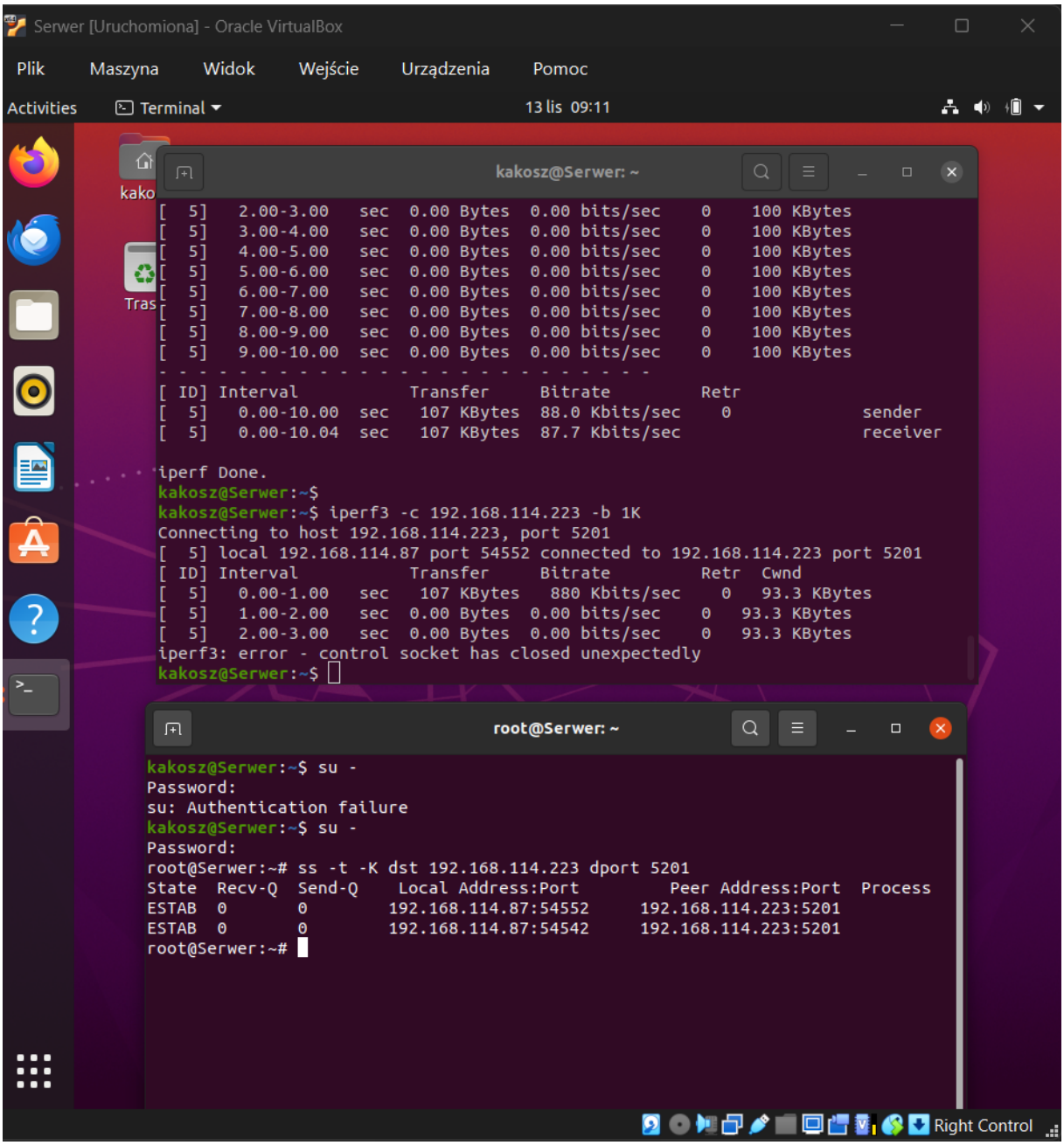

**Zrzut ekranu z Wireshark – RST, efekt nieudanego połączenia (dlaczego otrzymujemy RST?):**



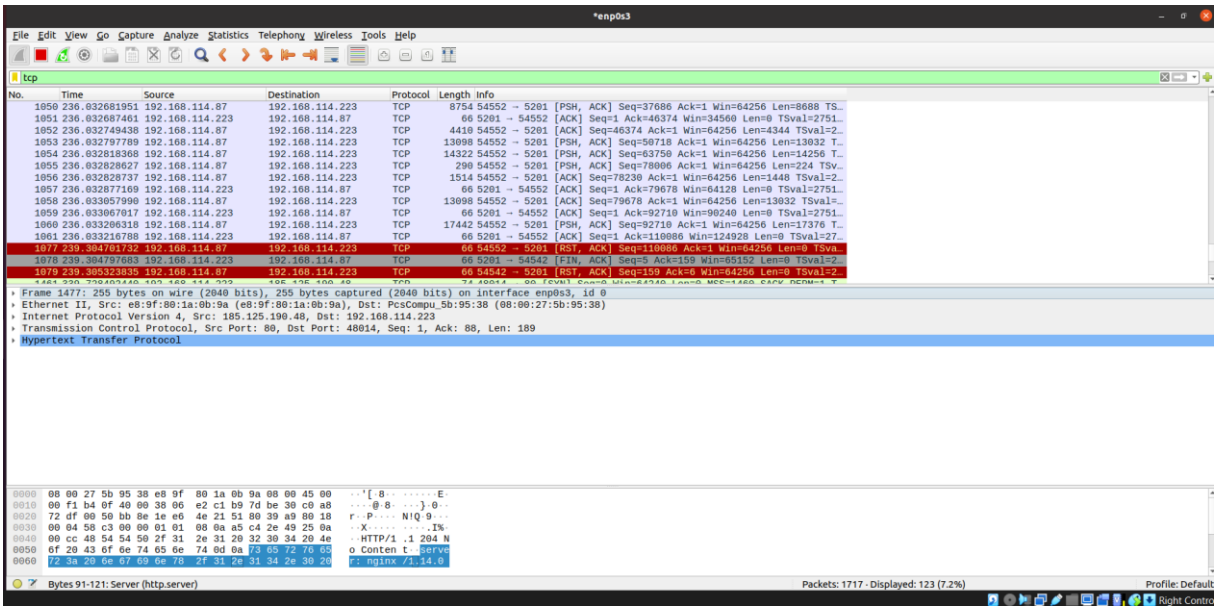

Otrzymanie pakietu o fladze [RST] jest normalną i oczekiwaną transmisją pojawiającą się w sytuacji gwałtownego zamknięcia procesu serwera. Pakiet o fladze [RST] informuje, że połączenie zostało natychmiastowo przerwane, a nie przeszło przez four-way handshake opisany wcześniej. Połączenie zostaje przerwane natychmiastowo. Strona otrzymująca pakiet o fladze [RST] jest poinformowana, o niepoprawnym zamknięciu sesji przez drugą stronę i sama również zakończy sesje.

- Nieplanowane – zabicie procesu (RST):
  - Włączamy proces serwera
  - Włączamy Wireshark po stronie serwera
  - Ponownie inicjujemy iperf po stronie klienta (z limitem -b 1K)
  - Po stronie klienta, w **drugim terminalu** robimy „su –„
  - Korzystamy z polecenia „ss -t -K” do zabicia połączenia TCP przez socket. Musimy podać również dst (adres docelowy) oraz dport (destination port).

**Zrzut ekranu z  klienta – komendy do iperf klient oraz ss (socket state) w dwóch terminalach:**




**Zrzut ekranu z  serwera Wireshark – RST jako wynik „ss -K”:**



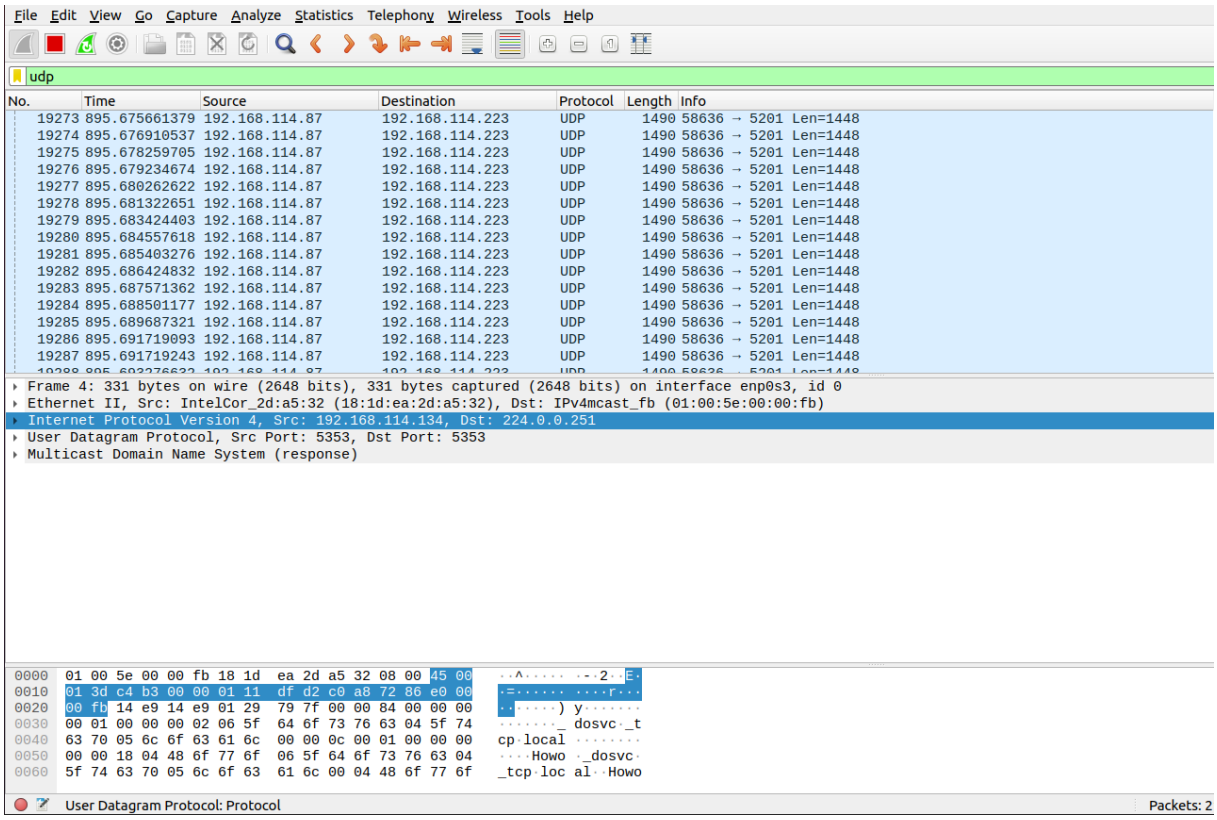

# Ćwiczenie 2 – Analiza ruchu sieciowego – warunki nominalne TCP (Wireshark, Iperf)

Podpowiedź:

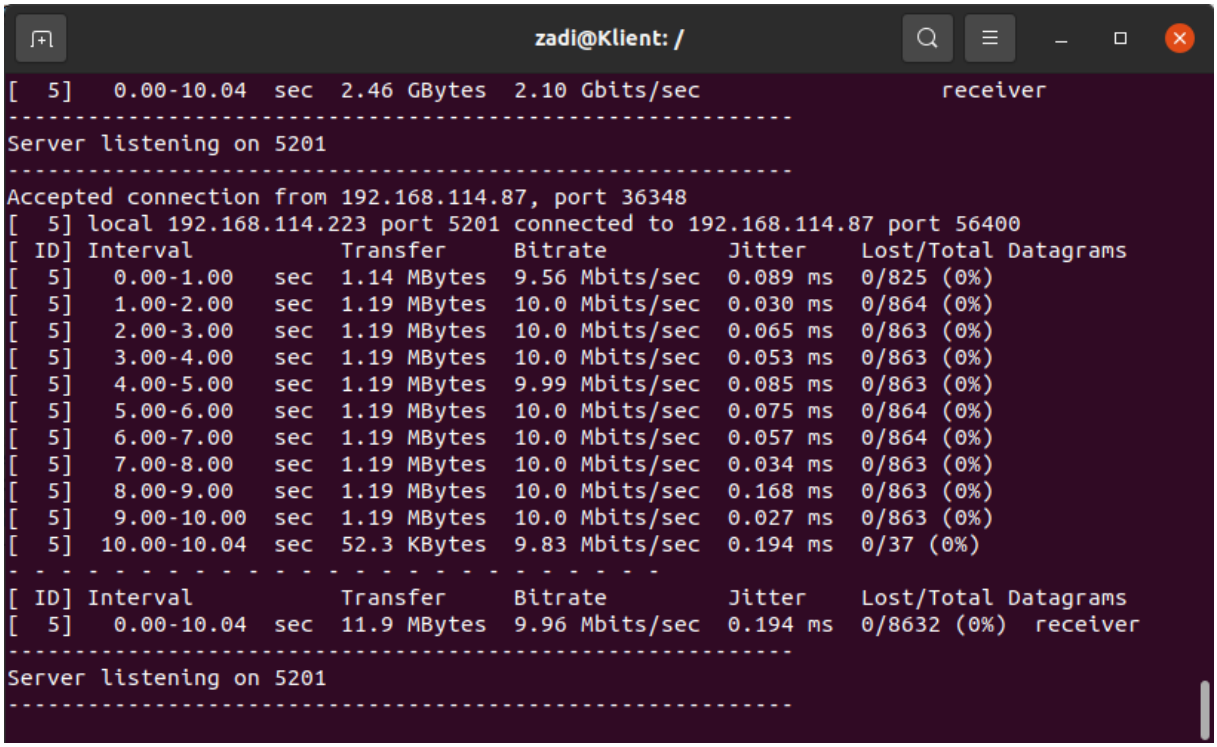
Dla TCP nie ograniczamy prędkości nadawania, badamy maksymalną przepływność łącza.

2. Powtarzamy badanie przepływności łącza  dla UDP oraz dla TCP (opcje klienta iperf)
    - Dla UDP Uruchamiamy serwer i klient z odpowiednią flagą (serwer – flaga dla udp, klient – flagi dla udp oraz do przepustowości, bandwidth (wybieramy sami))

**Zrzut ekranu wireshark z segmentami UDP**




**Czy po stronie serwera iperf widzimy dodatkowe wyniki dla UDP? Dlaczego?**




Ustawiliśmy przepustowość równą 10 Mbit\s.

Po stronie serwera iperf widzimy dodatkowe wyniki pojawiające się dla UDP. Są to: Jitter oraz Lost/Total Datagrams. Są to kolumny unikalne dla UDP (nie pojawiają się przy analizie TCP). UDP w przeciwieństwie do TCP po wysłaniu pakietu nie czeka na

informację powrotną, czy pakiet został otrzymany. Dlatego też, w UDP pojawiły się dwie nowe sekcje informujące nas o jakości połączenia. Jitter informuje nas o różnicy w odstępach czasu między pakietami wysyłanymi a pakietami otrzymanymi (jeżeli pakiety zostały otrzymane w większych odstępach niż zostały wysłane Jitter jest większy, informacja o czasie wysłana jest zawarta w pakiecie, co pozwala na kalkulacje Jitter). Lost/Total Datagrams informuje nas, ile pakietów zostało utraconych w transmisji. Bazuje to na zliczaniu numerów sekwencyjnych.
**Otrzymane przez nas 0% utraty pakietów oraz Jitter równy 0,194 ms świadczy o bardzo dobrym i stabilnym połączeniu.**

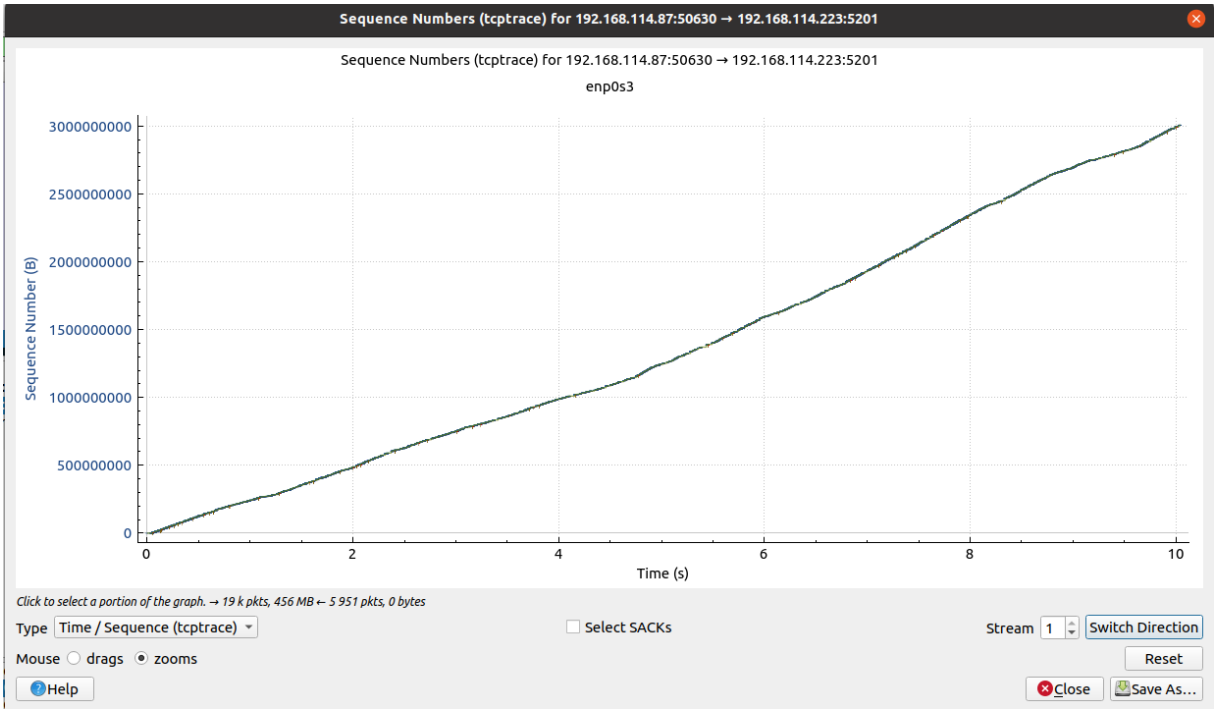
- Uruchamiamy pomiar dla TCP wraz z Wireshark (po stronie serwera) – analogicznie jak poprzednio (nie specyfikujemy maksymalnej przepustowości)
    - Jedna strona wywołuje komendę umożliwiającą odbieranie i analizę przychodzących pakietów: iperf3 -s
    - Druga strona wywołuje komendę pomiarową, która uruchamia całą procedurę testową iperf3 -c <adresIP> -t 10
    - Koniecznie łapiemy ruch TCP (uruchamiamy wireshark przed klientem)
    - Po zakończeniu wysyłania ruchu przez klienta (10 sekund), wpisujemy w Wireshark filtr „tcp”, oraz za pomocą pola „statistics, TCP stream graphs” generujemy wykresy (opisane poniżej)

**Wynik pomiaru iperf (serwer) dla TCP (będziemy porównywać z opóźnieniami, stratami pakietu):**

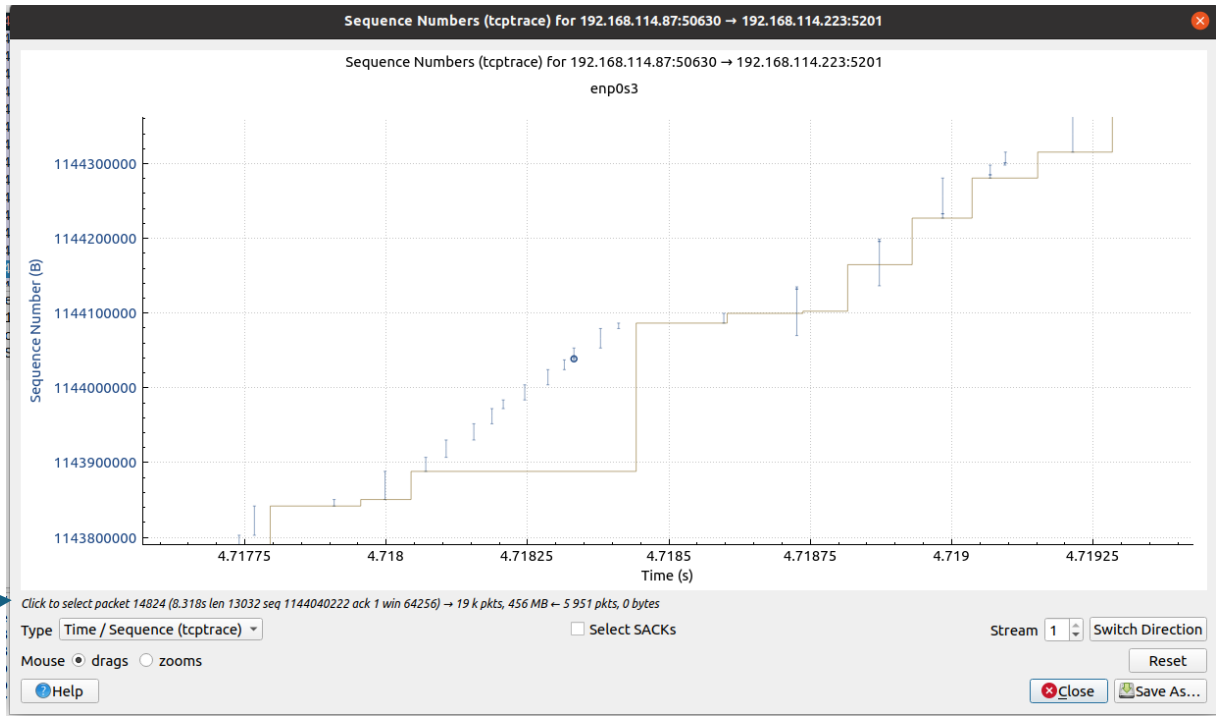
```
zadi@Klient:/$ iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 192.168.114.87, port 37730
[  5] local 192.168.114.223 port 5201 connected to 192.168.114.87 port 37746
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-1.00   sec   239 MBytes  2.00 Gbits/sec
[  5]   1.00-2.00   sec   236 MBytes  1.98 Gbits/sec
[  5]   2.00-3.00   sec   260 MBytes  2.18 Gbits/sec
[  5]   3.00-4.00   sec   271 MBytes  2.27 Gbits/sec
[  5]   4.00-5.00   sec   265 MBytes  2.23 Gbits/sec
[  5]   5.00-6.00   sec   231 MBytes  1.94 Gbits/sec
[  5]   6.00-7.00   sec   266 MBytes  2.23 Gbits/sec
[  5]   7.00-8.00   sec   304 MBytes  2.55 Gbits/sec
[  5]   8.00-9.00   sec   295 MBytes  2.47 Gbits/sec
[  5]   9.00-10.00  sec   313 MBytes  2.63 Gbits/sec
[  5]  10.00-10.05  sec  9.46 MBytes  1.64 Gbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-10.05  sec  2.63 GBytes  2.25 Gbits/sec                  receiver
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

W TCP nie ma Jittera oraz Lost/Total Datagrams. TCP polega na wysyłaniu pakietów oraz potwierdzaniu otrzymania ich. Stąd te 2 kategorie tutaj się nie pojawiają.

**Wykres TCP: Time/Sequence tcptrace <zrzut ekranu>:**

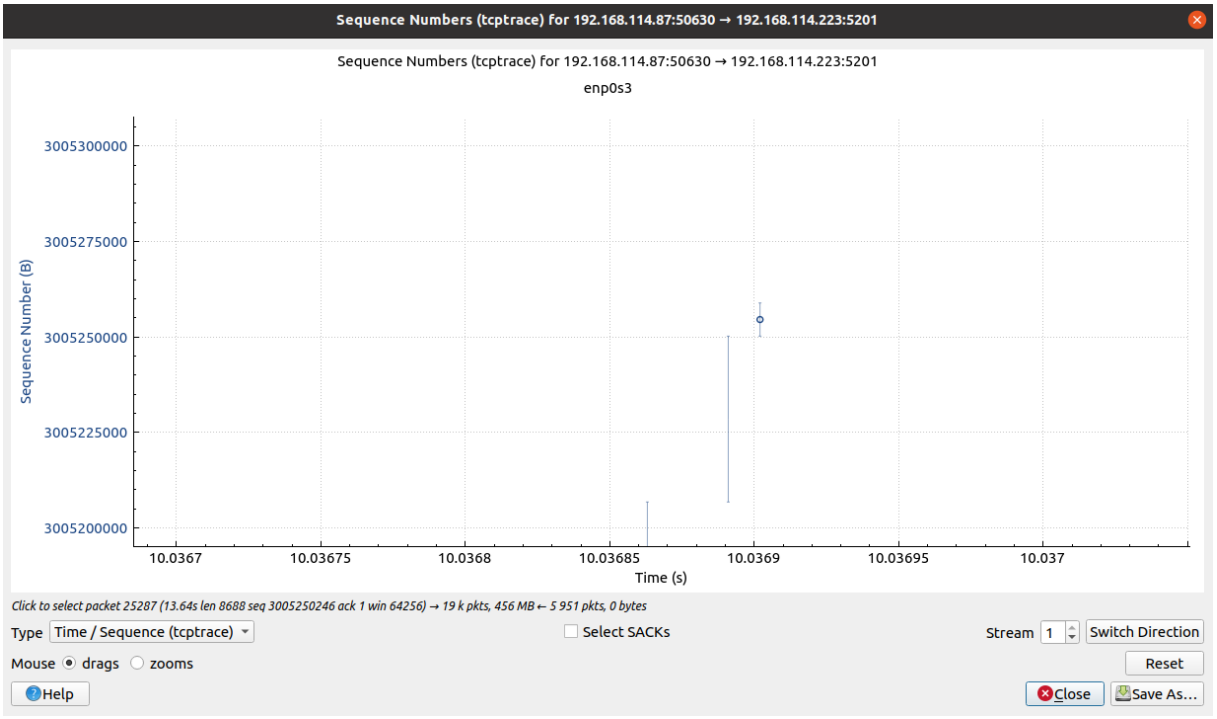



- **Czego możemy dowiedzieć się o indywidualnych pakietach (najedź myszką na wykres, przybliż wykres)?**



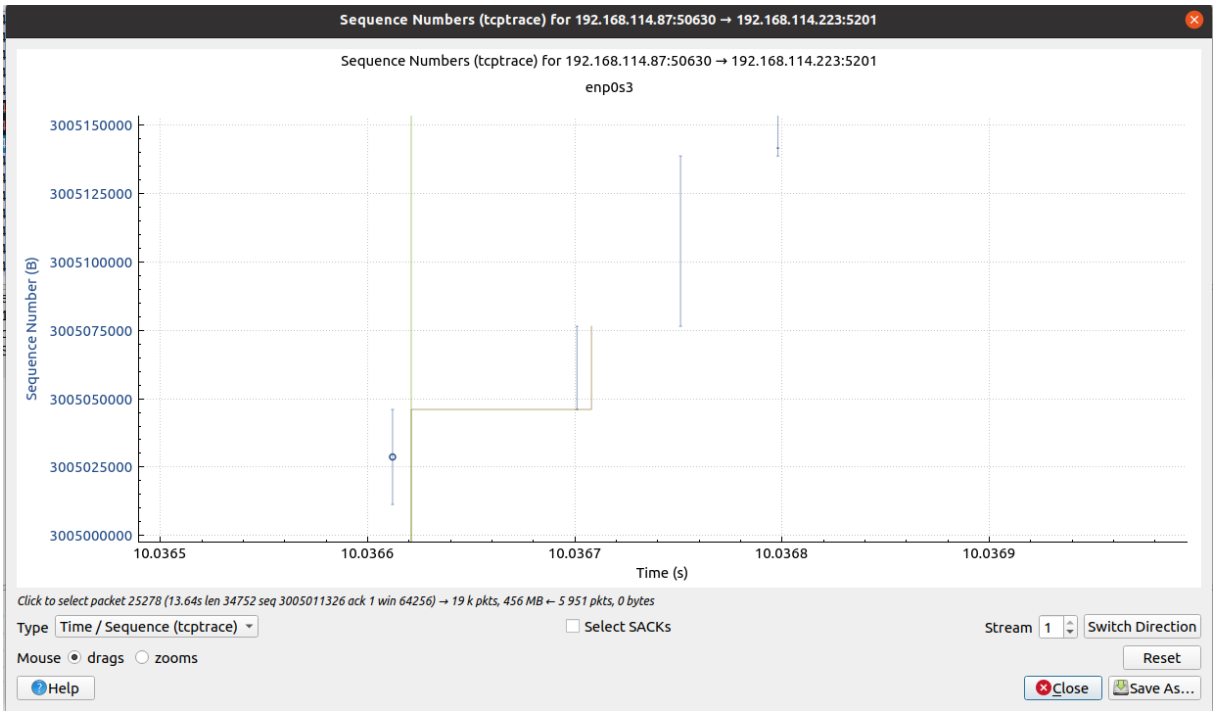

Najeżdżając myszą na pojedynczy pakiet jesteśmy w stanie dowiedzieć się o nim następujące informacje: ID pakietu (14824); Długość danych (13032); numer sekwencyjny (1144040222); numer ack (1); win – wolne miejsce w buforze (64256 bajt)

- **Jaki jest maksymalny numer sekwencji?**




Maksymalny numer sekwencji to: 3005250246

- **Co oznaczają kolory (zielony, szary, niebieski) po przybliżeniu wykresu?**

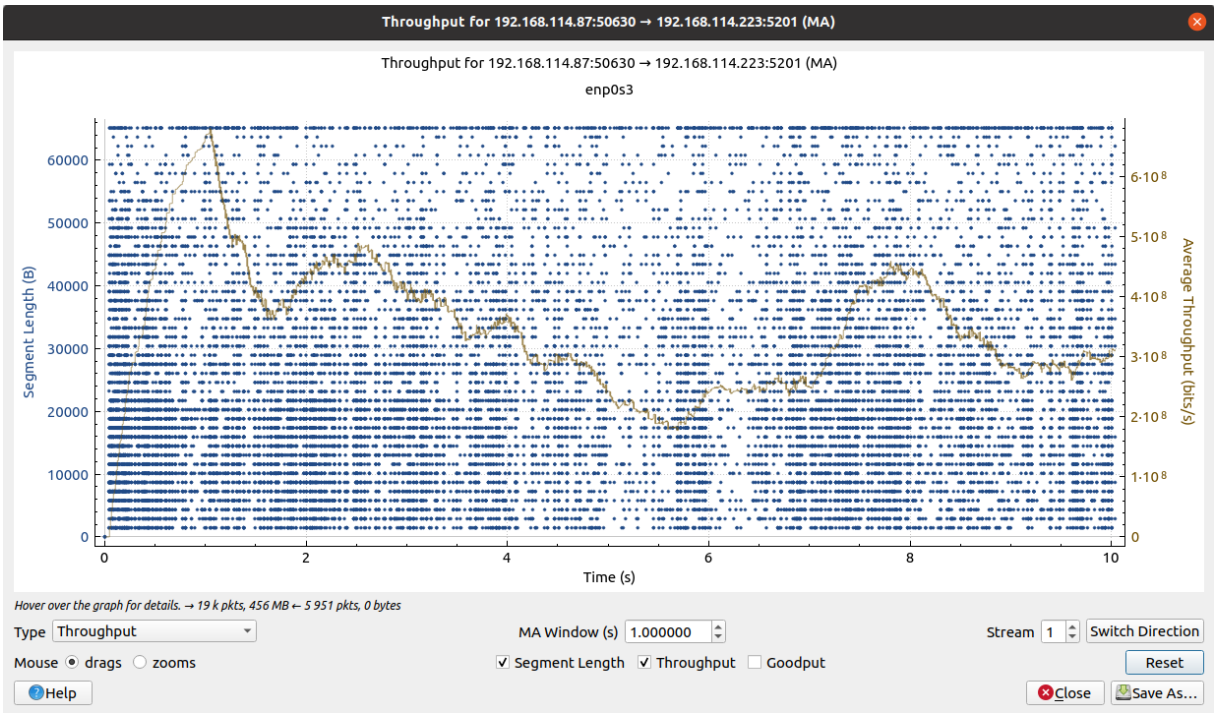



Niebieską linią, w formie schodków (na zdjęciu nie widać - za bardzo przybliżone), zaznaczone są dane w locie. Niebieska linia reprezentuje numery sekwencyjne (OY) wysyłanych danych w czasie (OX). Podniesienie się tej linii informuje nas o tym, że dane zostały wysłane.

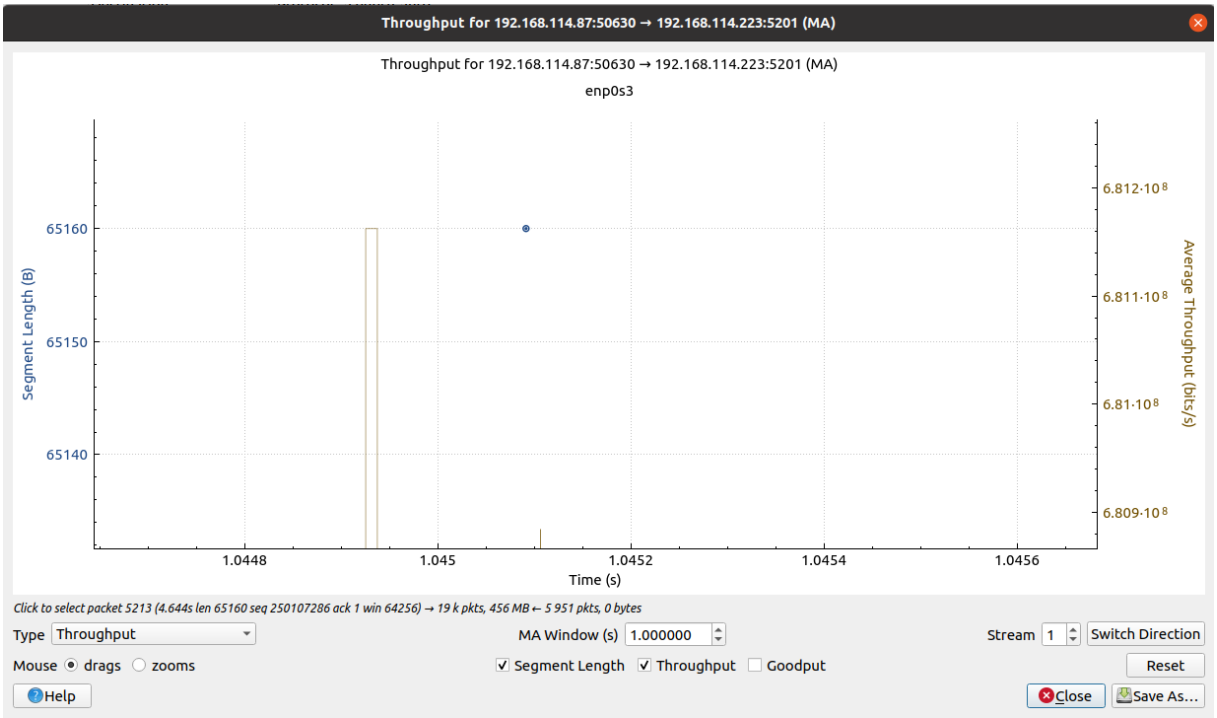
Zieloną linią zaznaczone jest okno odbioru danych. Pokazuje to, ile jest wolnego miejsca w danym momencie w buforze odbiornika. Nadawca nie może wysłać danych które wyszłyby poza zieloną linię (niebieska linia nie może przekroczyć zielonej).

Szarą/Brązową linią zaznaczone są potwierdzone dane. Ta linia wskazuje, które bajty danych zostały zatwierdzone pakietem oznaczonym flagą [ACK] przez odbiornik. Podniesienie się tej linii oznacza, że dane zostały dostarczone.

**Wykres TCP: Throughput <zrzut ekranu>:**

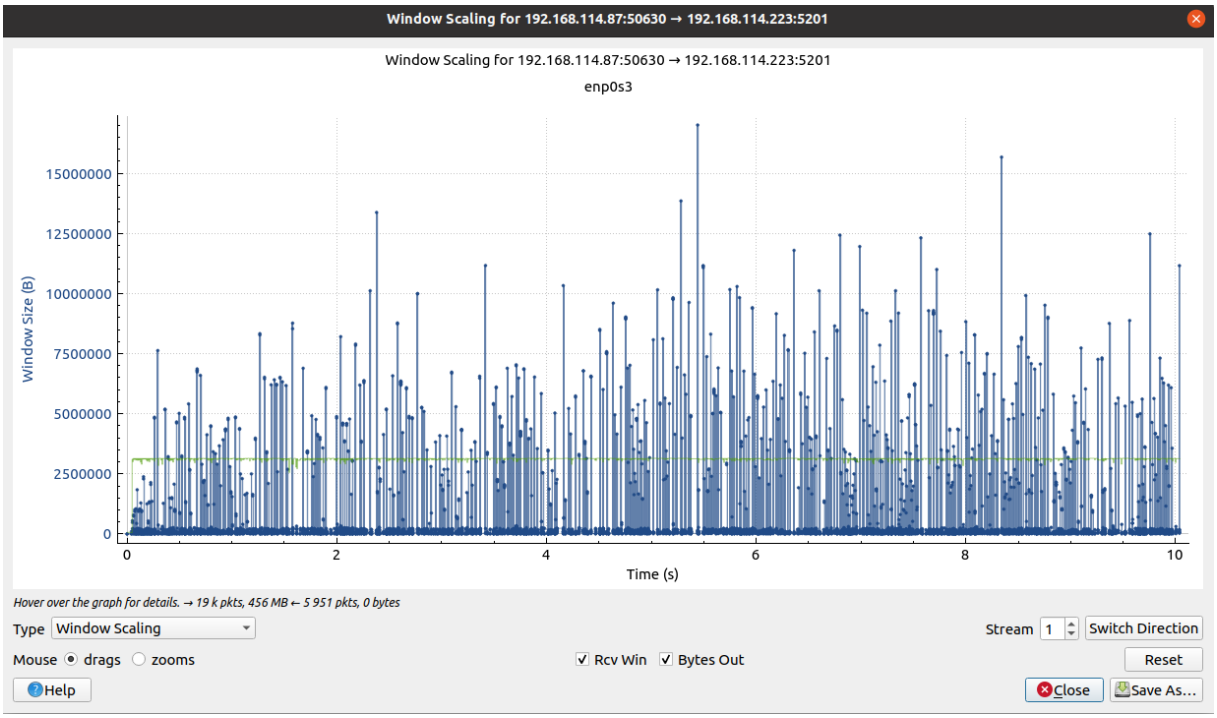



- **Ile wynosił maksymalny throughput?**




Maksymalny throughput wynosił około: $6{,}8116 \cdot 10^8$.

**Wykres TCP: Window Scaling<zrzut ekranu>:**

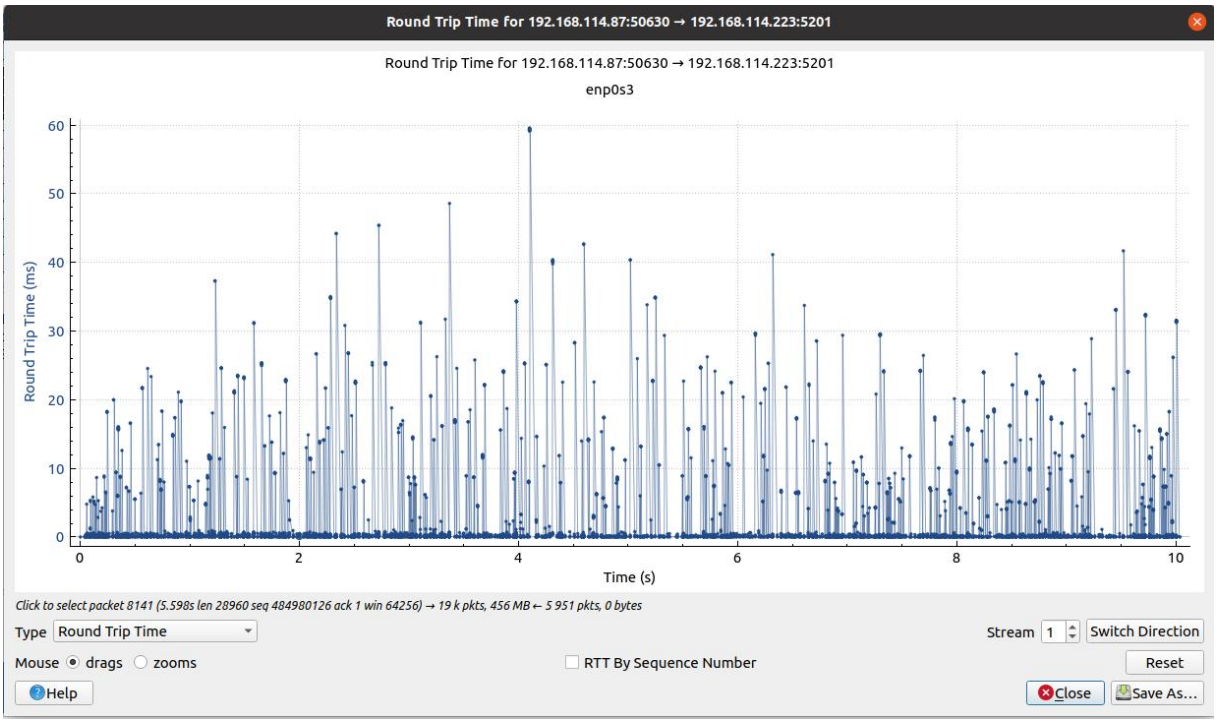



- **Co oznaczają wykresy na niebiesko i zielono?**

Niebieska linia tak jak wcześniej, informuje o bitach w locie, wysłanych bitach transmisji.

Zielona linia tak jak wcześniej, pokazuje limit bufora odbiornika informujący o możliwości oraz ilości danych jaką może odebrać.

**Wykres TCP: RTT <zrzut ekranu>:**



**- Spróbuj oszacować RTT na bazie skali wykresu**

Oszacowane przez nas RTT wynosi mniej niż 1ms. Analizując wykres widzimy nagromadzenie się punktów przy osi OX. Występujące na wykresie piki (punkty o dużym RTT) nie mają dużego wpływu na RTT ze względu na ich niedużą ilość w porównaniu do punktów znajdujących się tuż przy osi OX. Powyżej granicy 10 ms ilość punktów drastycznie spada.

# Ćwiczenie 2 – Analiza ruchu sieciowego w niekorzystnych warunkach – TCP oraz UDP (Wireshark, Iperf, netem)

Moduł netem powinien być już zainstalowany na VM. Instrukcja: https://manpages.ubuntu.com/manpages/noble/en/man8/tc-netem.8.html

Uwaga – pod koniec każdego „typu chaosu", usuwamy chaos z interfejsu – **chcemy mieć tylko jeden „chaos" jednocześnie, nie łączymy ich!**

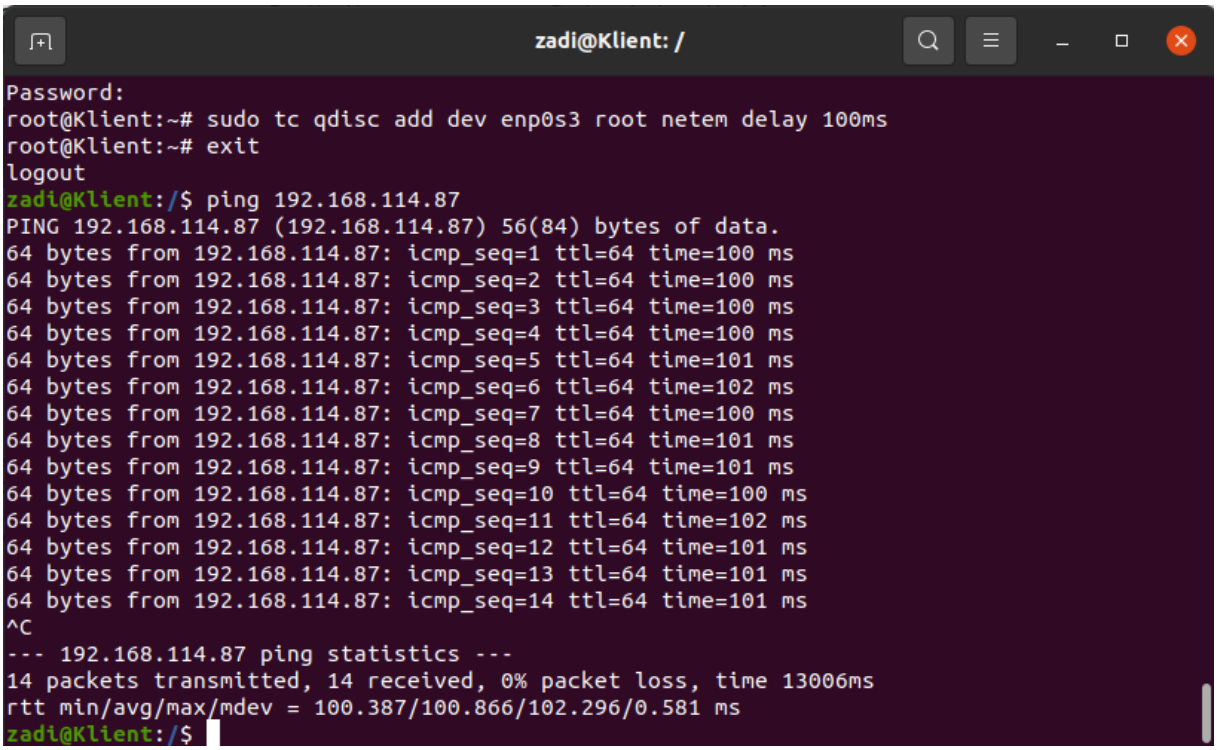
## Dodatkowe opóźnienie – UDP vs TCP

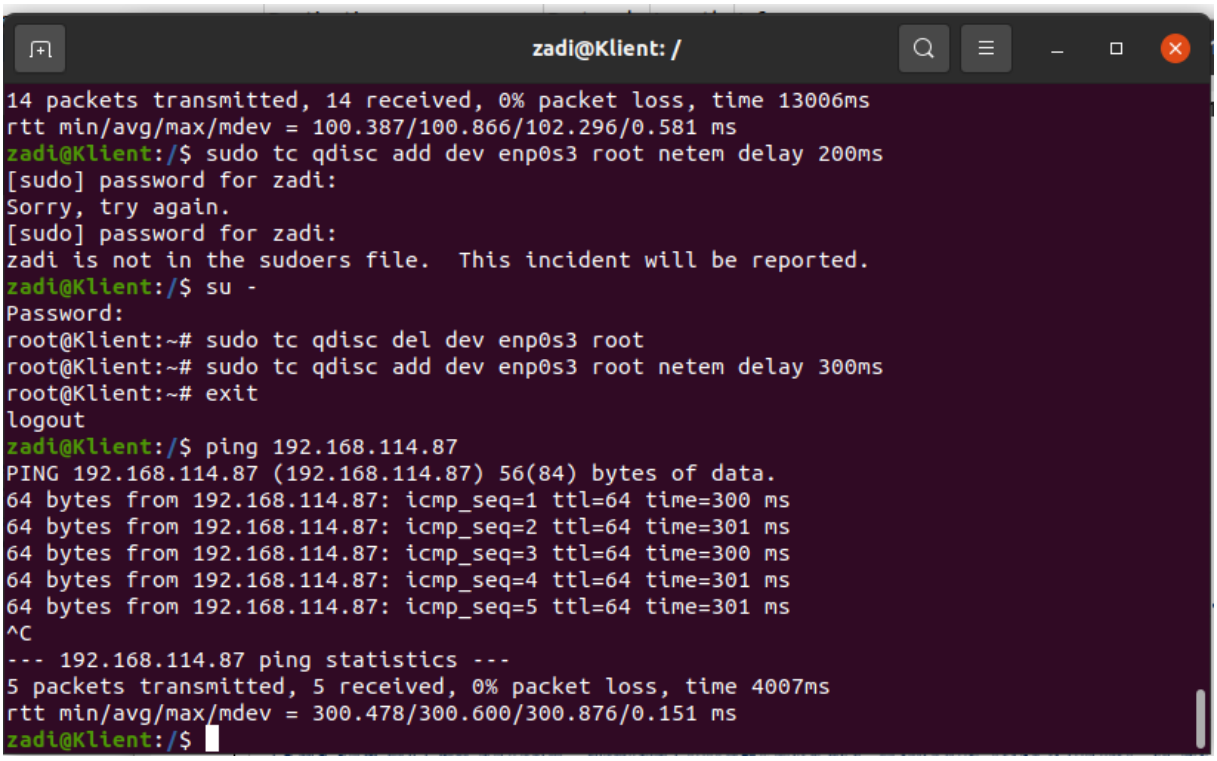
*W stronę interfejsu serwera wprowadzamy dodatkowe opóźnienie – np. 100 ms, (czyli RTT +200 ms).*

**Komenda do dodania chaosu dla interfejsu:**

*Sudo tc qdisc add dev enp0s3 root netem delay 100ms*

**Sprawdzamy Ping czy opóźnienie wzrosło <zrzut ekranu>:**

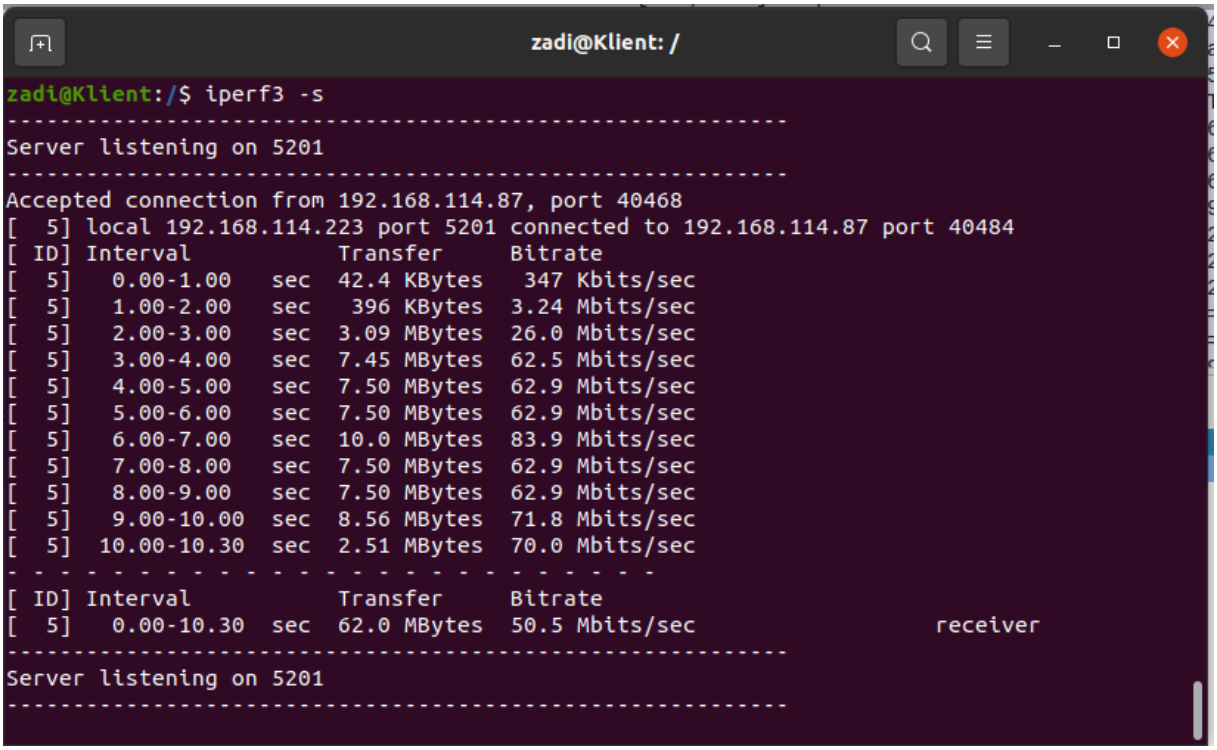


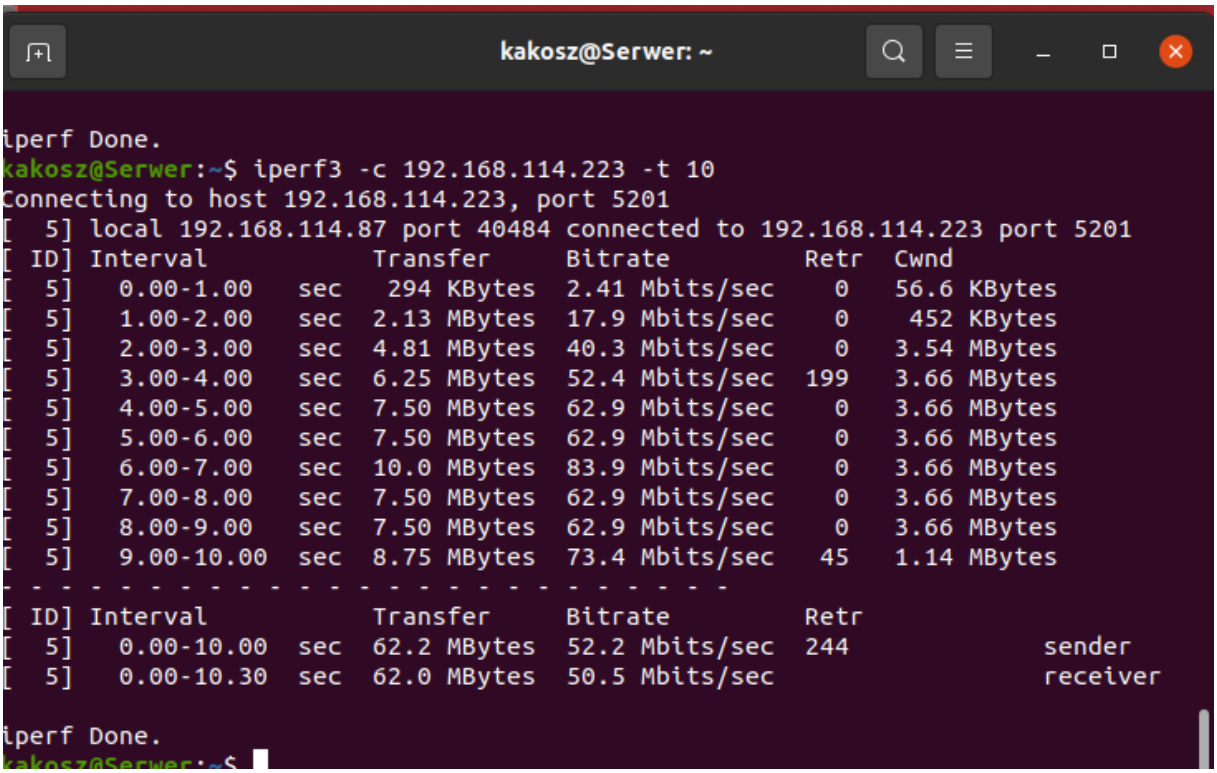




Powtarzamy pomiar dla TCP wraz z Wireshark (po stronie serwera) – analogicznie jak poprzednio (nie specyfikujemy maksymalnej przepustowości)

- Łapiemy Wireshark pakiety „TCP” oraz przygotowywujemy się do generowania wykresów.

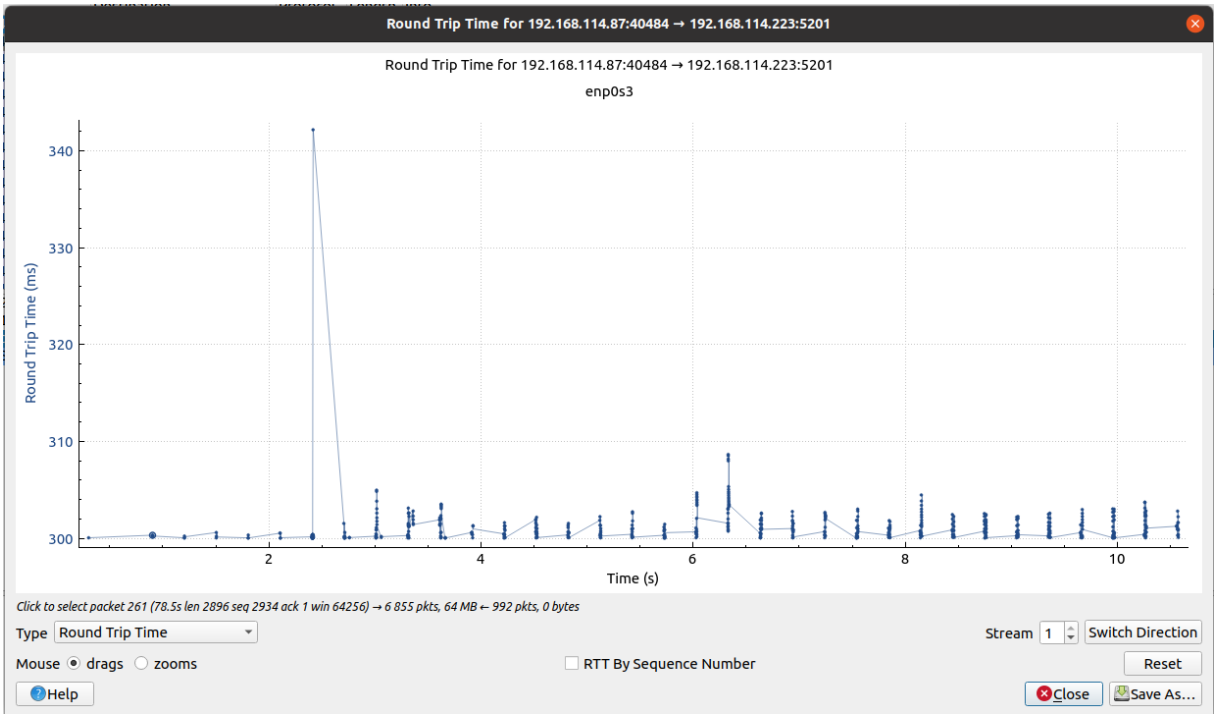
**Wyniki iperf TCP <Zrzut ekranu>. Czym się różnią? Dlaczego?**



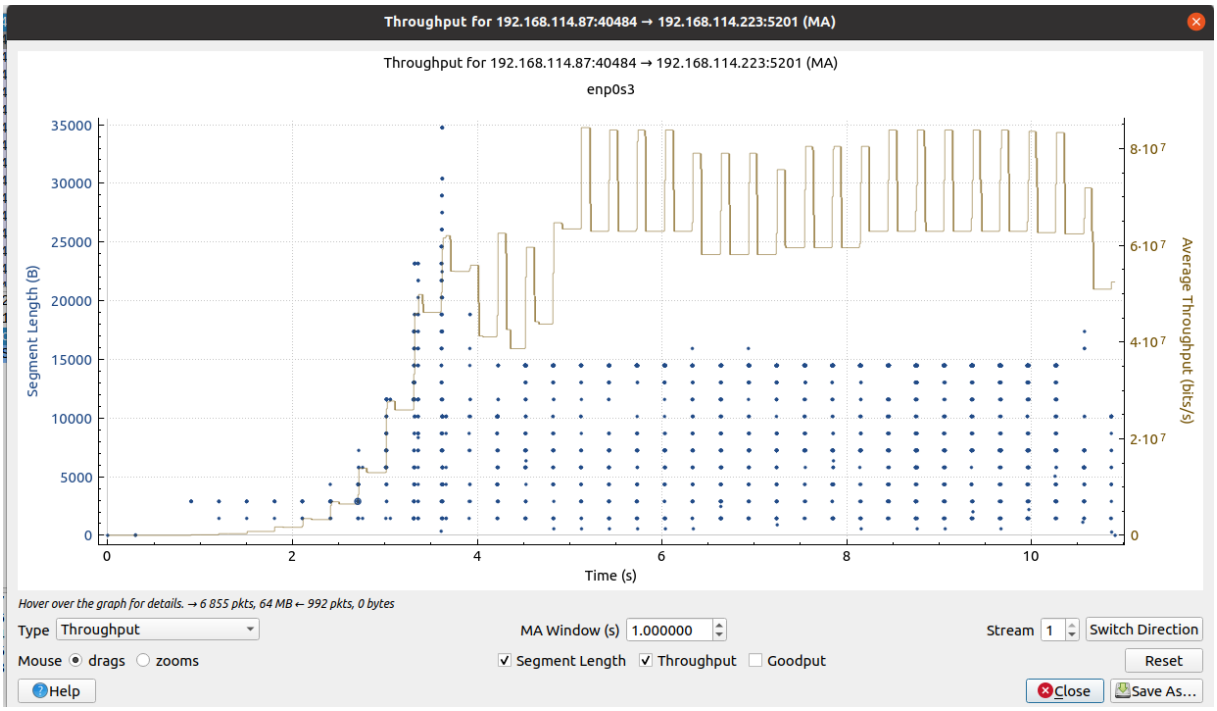




Dla komendy iperf3 -c pojawiły się dwie nowe kolumny: Retr oraz Cwnd. Retr oznacza *Retransmissions* i informuje nas o ilości segmentów TCP, które musiały zostać wysłane ponownie (nie został otrzymany [ACK]). Cwnd natomiast oznacza *Congestion Window* i informuje nas o liczbie bajtów które można wysłać w danym momencie. Cwnd jest utrzymywane przez nadawcę (nie jak poprzednio przez odbiornik) i zapobiega przeciążeniu połączenia z odbiornikiem.

## Wykres RTT TCP  <Zrzut ekranu>

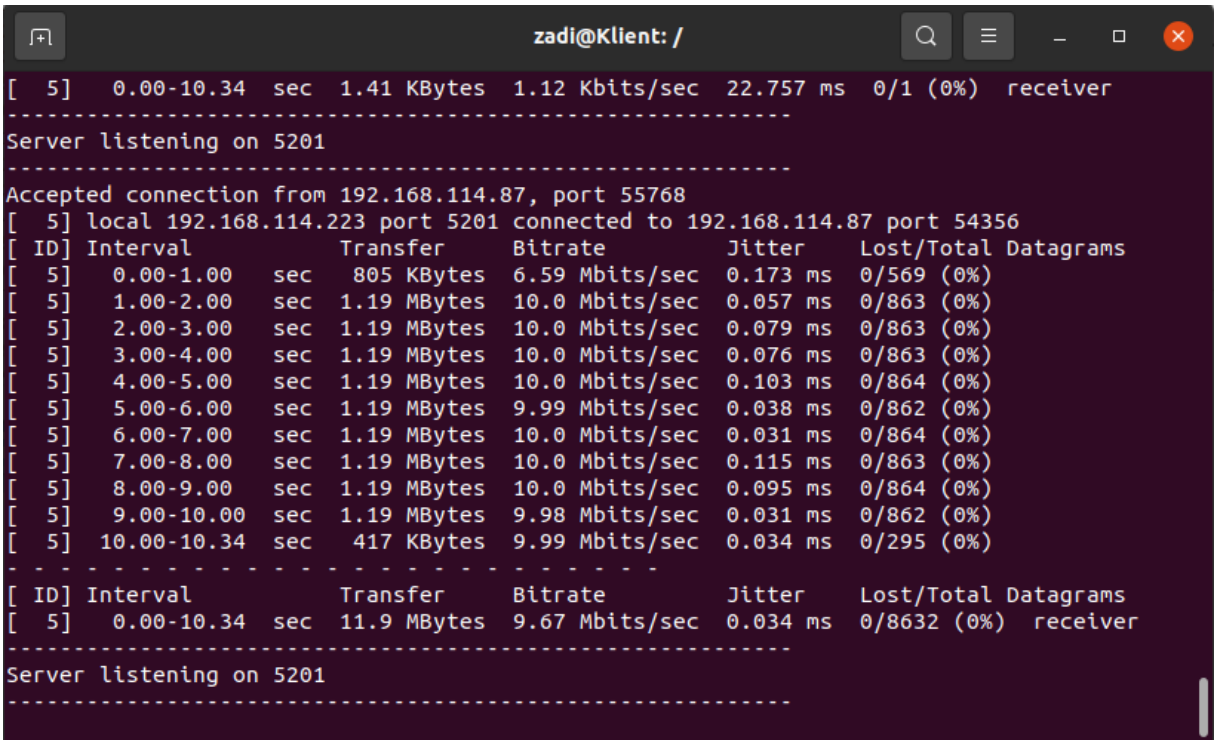



## Wykres Throughput TCP  <Zrzut ekranu>



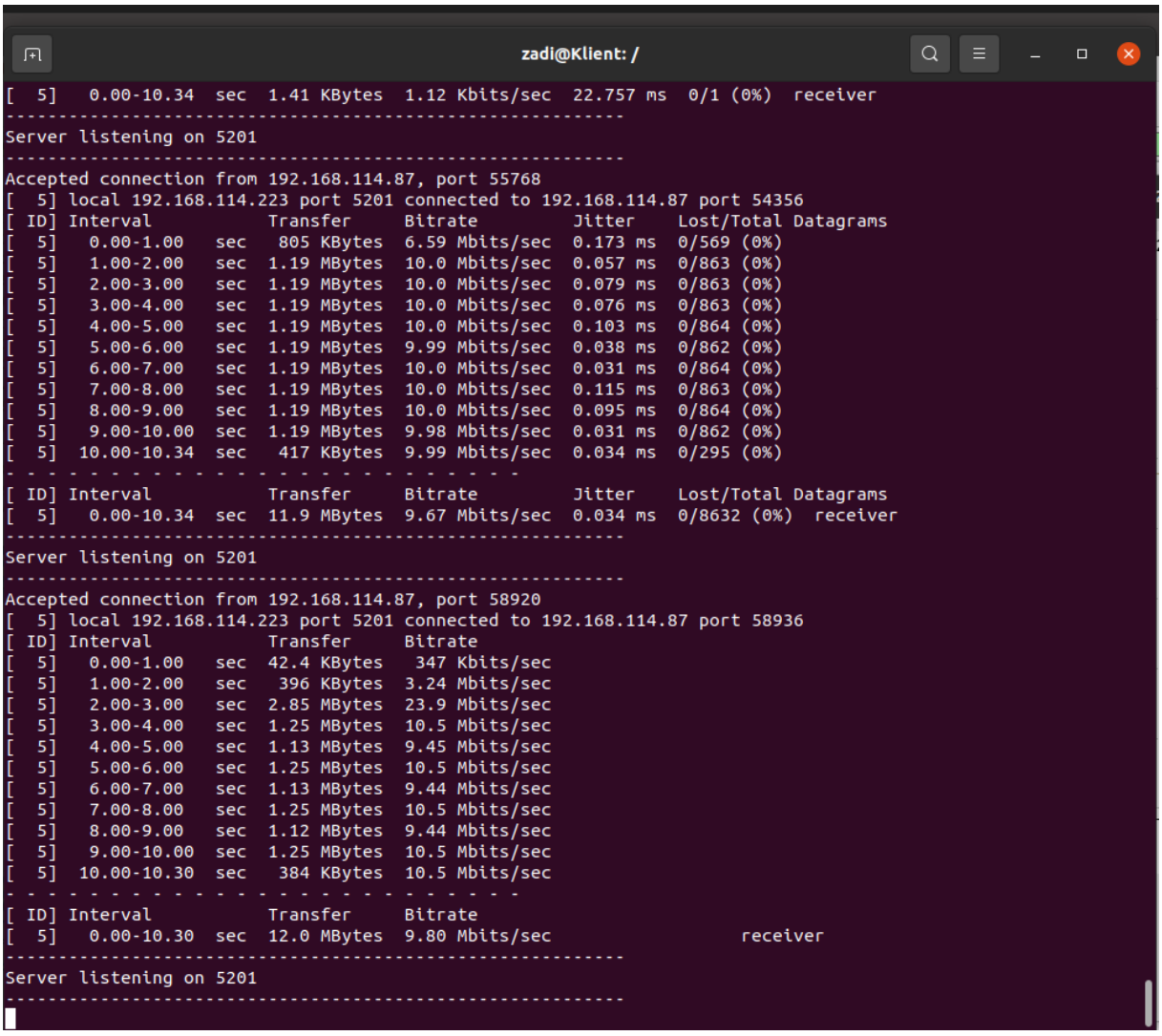

Powtarzamy pomiar dla UDP – bez Wireshark. Pomiar ma miec te same parametry co w stanie nominalnym (czas, przepustowość).

## Wyniki iperf UDP <Zrzut ekranu>. Czym się różnią?




Wynik tak jak wcześniej różni się występowaniem kolumny z danymi Jitter oraz Lost/Total Datagrams. Ich działanie wytłumaczone jest na początku ćwiczenia drugiego. Widać również, że nie pojawiły się kolumny z danymi Retr oraz Cwnd, ponieważ są to dane pojawiające się tylko dla protokołu TCP (nie UDP).

**Który z protokołów, UDP czy TCP jest bardziej podatny na dodatkowe opóźnienia? Dlaczego?**




TCP (na dole) jest bardziej podatny na opóźnienia niż UDP. Widać to np. porównując do siebie Bitrate tych dwóch protokołów. Protokół TCP jest skupiony na niezawodnym przesyłaniu danych. TCP opiera się na potwierdzaniu otrzymania pakietów oraz zapewnieniu, że wszystkie pakiety otrzymuje. Powoduje to, że ten protokół bardziej skupia się na rzetelności przesyłu danych kosztem opóźnień. UDP natomiast nie musi czekać na pakiety potwierdzające otrzymanie danych, stąd też nie jest tak podatny na opóźnienia jak TCP.

**Komenda do usunięcia chaosu dla interfejsu:**

sudo tc qdisc del dev enp0s3 root

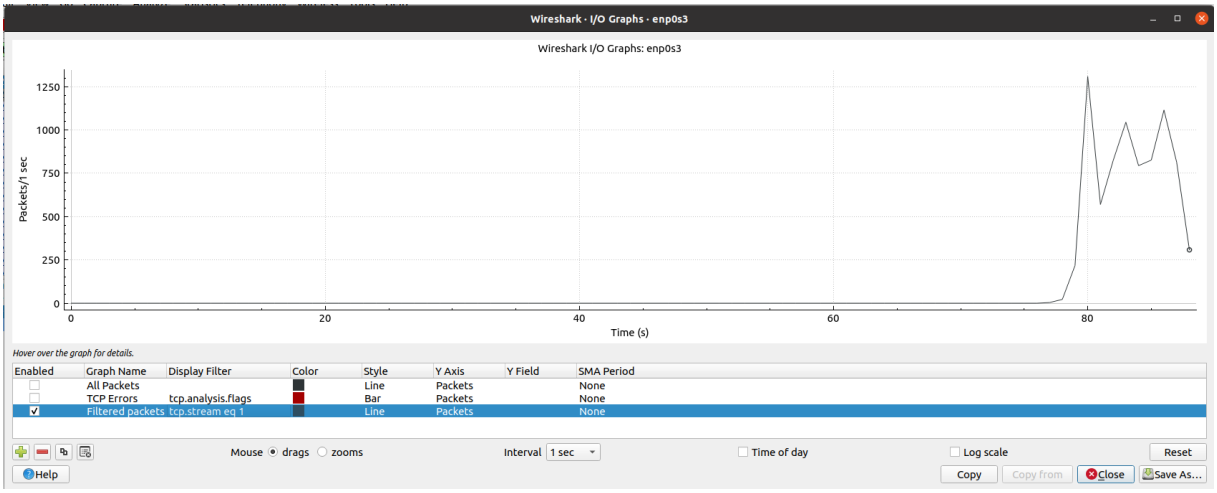
## Strata pakietów – UDP vs TCP

*Powtarzamy pomiary dla UDP z tymi samymi parametrami, jedynie tym razem dodajemy stratę pakietów na poziomie 5%.*

**Komenda do dodania chaosu dla interfejsu:**

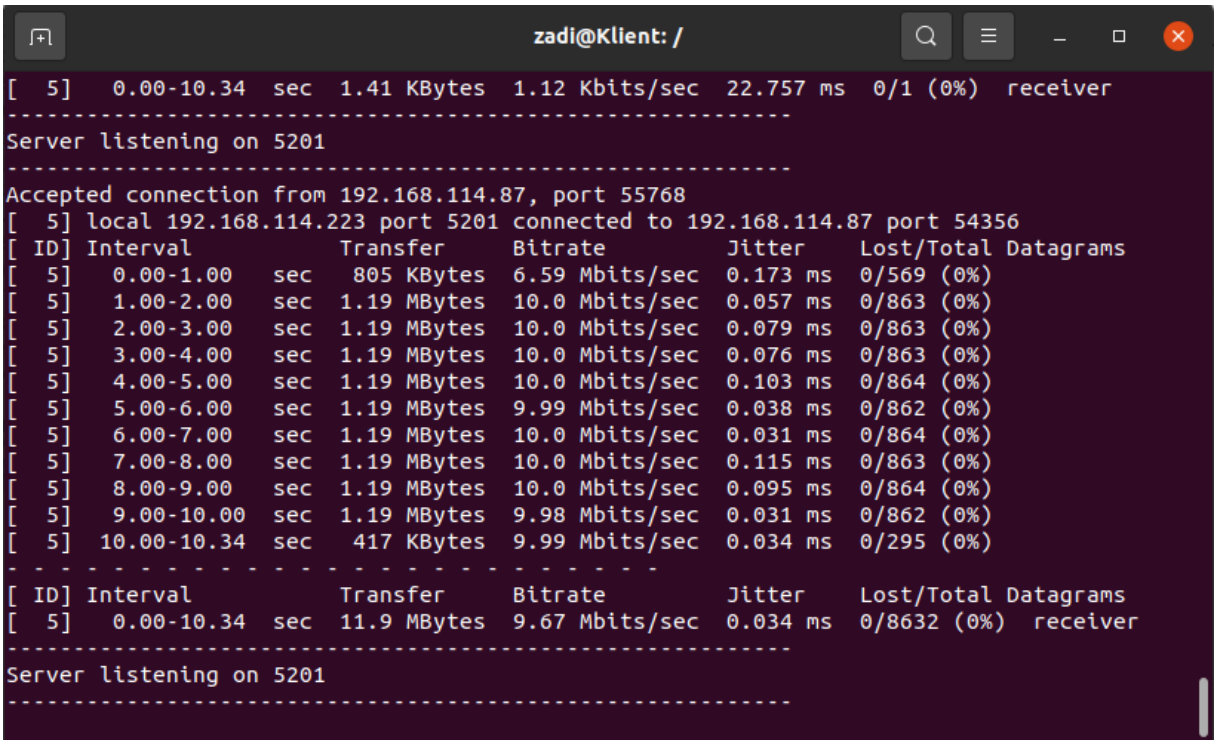
sudo tc qdisc add dev eth0 root netem loss 5%

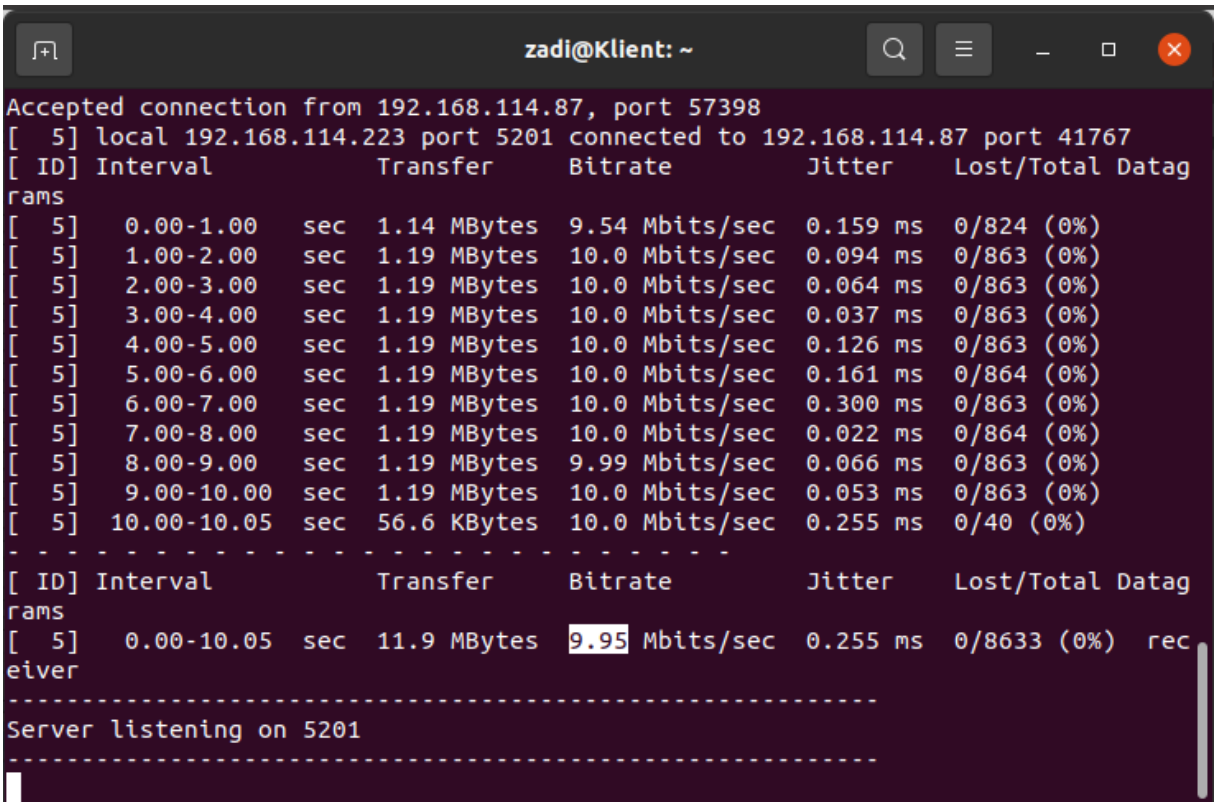
**Wireshark – czy zobaczymy w jakikolwiek sposób straty pakietów dla UDP? Jeśli tak, to <zrzut ekranu>.**




Nie ma strat – gdyby były, pojawiłyby się luki w wykresie I/O.

**Jak zmieniły się wyniki iperf UDP w stosunku do stanu nominalnego? O ile% zmalał throughput?**



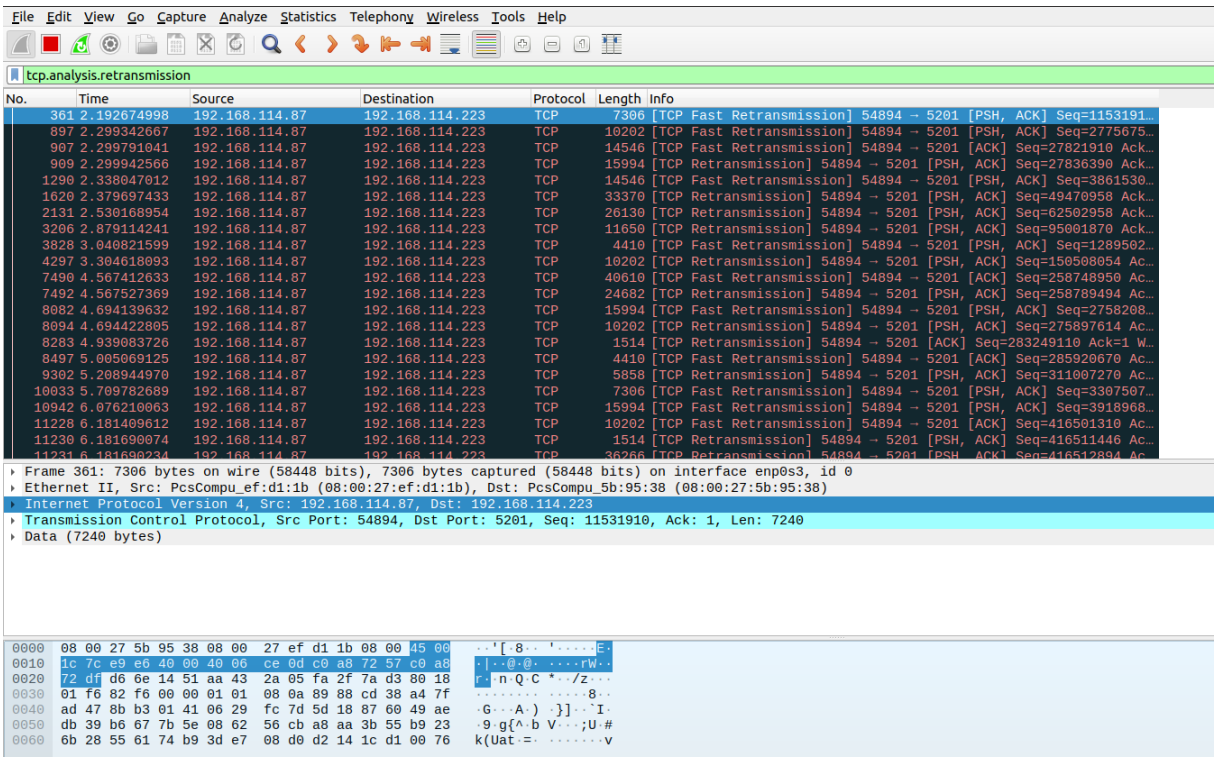

Na pierwszym zdjęciu ukazany jest stan nominalny wyników iperf UDP. Na drugim zdjęciu ukazany jest stan badany. Oba wyniki różnią się wartościami Bitrate (spadek o około 0,3 Mbit/s) oraz wartością Jitter (7,5 razy większa od stanu nominalnego).

Poniższe równanie opisuje o ile procent zmalał throughput dla protokołu UDP w stosunku do stanu nominalnego:

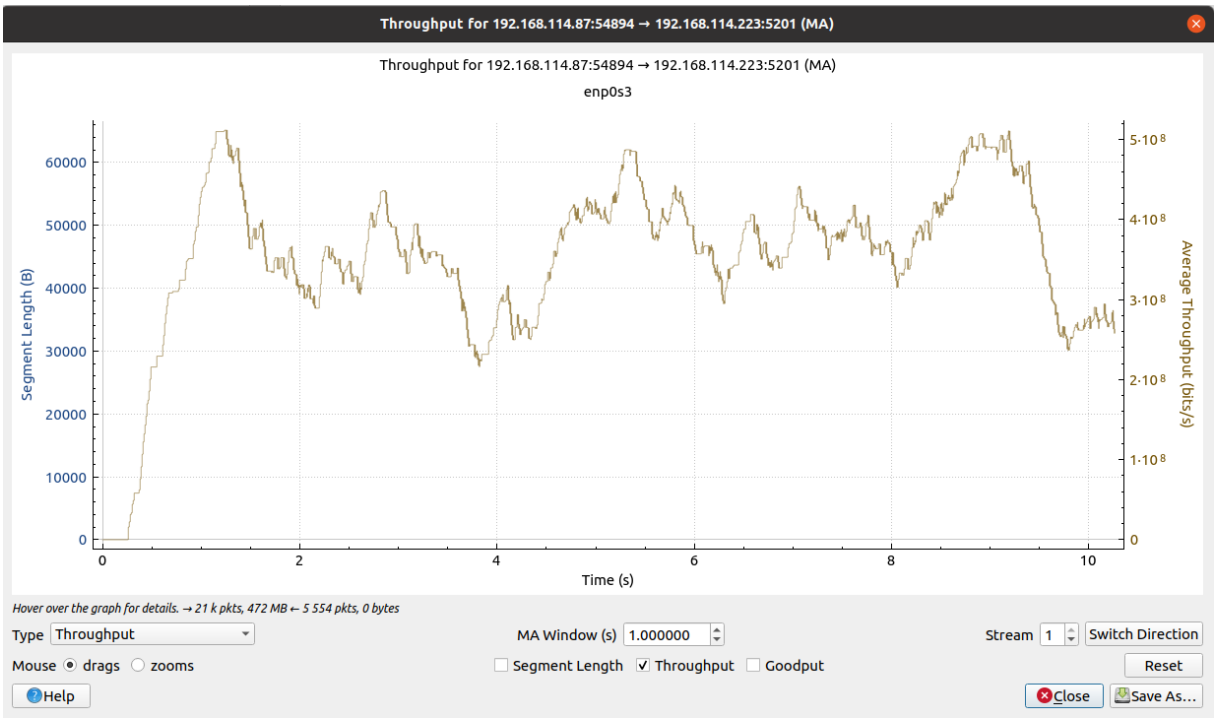
$$\frac{9{,}67 \;-\; 9{,}95}{9{,}67} \cdot 100\,\% \;\approx -3\,\%$$

*Powtarzamy pomiary dla TCP z tymi samymi parametrami, jedynie tym razem dodajemy stratę pakietów na poziomie 5% (zostawiamy interfejs z chaosem packet loss).*

**Wireshark – efekty straty pakietów, TCP Retransmission (np.) <Zrzut ekranu>:**




**<Zrzut ekranu> Graph Throughput TCP – wyjaśnij co się na nim dzieje**




Wykres przedstawiony na zdjęciu powyżej przedstawia jak średnia przepustowość zmieniała się w czasie trwania połączenia między kakosz@serwer a zadi@klient.

Widoczna na wykresie jest faza powolnego startu charakterystyczna dla protokołu TCP (pierwsze 1,5 sekundy). Jest to moment, gdzie przepustowość gwałtownie rośnie od zera.

Po fazie powolnego startu wartość przepustowości utrzymuje się, lecz waha, w przedziale od $3 \cdot 10^8$ do $5 \cdot 10^8$.

Widać, że dla protokołu TCP wartość przepustowości waha się, a nie jest ustabilizowana (jak dla UDP). Jest to spowodowane oczekiwaniem na pakiety powrotne o fladze [ACK] oraz specyfikacją protokołu TCP, jaką jest upewnianie się, że każdy pakiet danych zostanie dostarczony do odbiornika.

**Jak zmieniły się wyniki TCP iperf w stosunku do stanu nominalnego? O ile% zmalał throughput?**

Bitrate ze stratą pakietów na poziomie 5 % wyniósł 1,05 Gbit/s, z kolei nominalny wynosił 2,25 Gbit/s. Zatem Throughput zmienił się o:

$$\frac{1,05 - 2,25}{2,25} \cdot 100\ \% = -53,(3)\ \%$$

**Który z protokołów, UDP czy TCP jest bardziej podatny na stratę pakietów? Dlaczego?**

Żaden z protokołów nie jest bardziej podatny na stratę pakietu. Straty pakietów nie wynikają z rodzaju protokołu, którym je transmitujemy. Kluczowym zagadnieniem jest natomiast to, jak te dwa protokoły – TCP oraz UDP – radzą sobie z utratą pakietów.

Protokół TCP w przypadku utraconego pakietu (nie otrzymał [ACK]) przesyła pakiet po raz kolejny i upewnia się, że każdy pakiet trafia do odbiornika.

Protokół UDP wysyła pakiety do odbiornika, natomiast nie interesuje go, czy te pakiety dotarły do odbiornika czy nie. Oczywiste jest, że protokół robi wszystko tak, aby te pakiety dotarły (nie wysyła ich byle gdzie), natomiast nie idzie o krok dalej, tak jak to robi TCP, aby potwierdzić, że pakiet został przez odbiornik otrzymany.

Tym samym można powiedzieć, **że UDP jest bardziej podatny na stratę pakietów**, lecz wynika to z faktu, że ten protokół, nie upewnia się, że pakiet został prawidłowo dostarczony.

**Komenda do usunięcia chaosu dla interfejsu:**

sudo tc qdisc del dev enp0s3 root

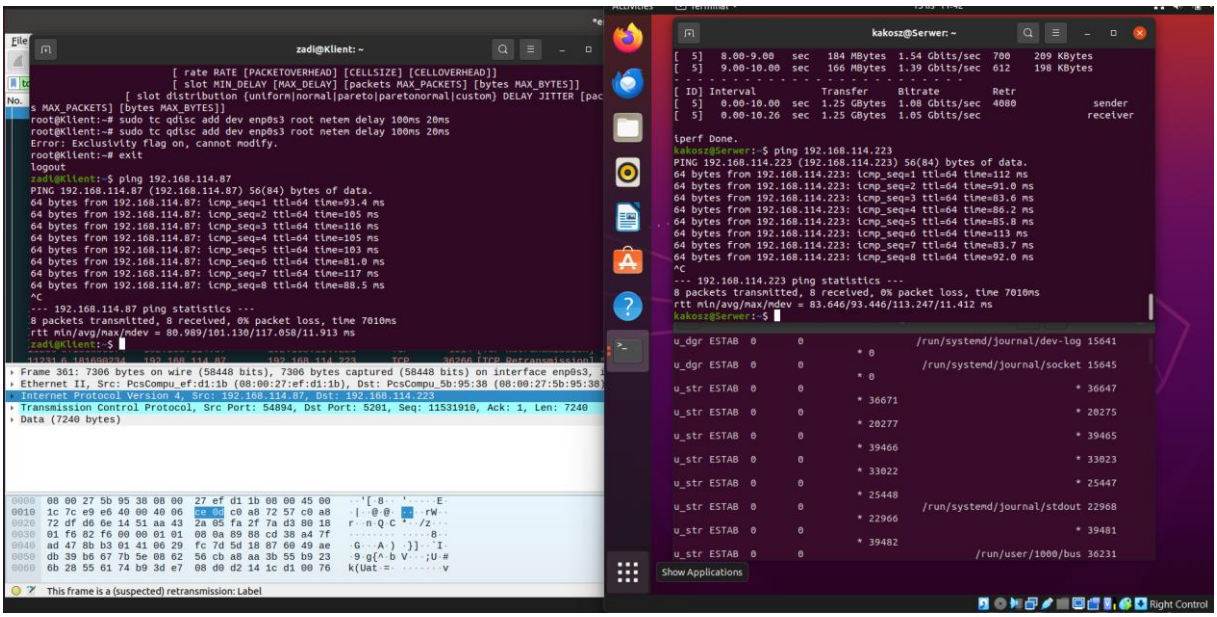
## Dodatkowy Jitter (zmienność opóźnienia)– UDP

*W stronę interfejsu serwera wprowadzamy dodatkowe opóźnienie i jitter – np. 100 ms delay, 20ms jitter.*

**Komenda do dodania chaosu dla interfejsu:**

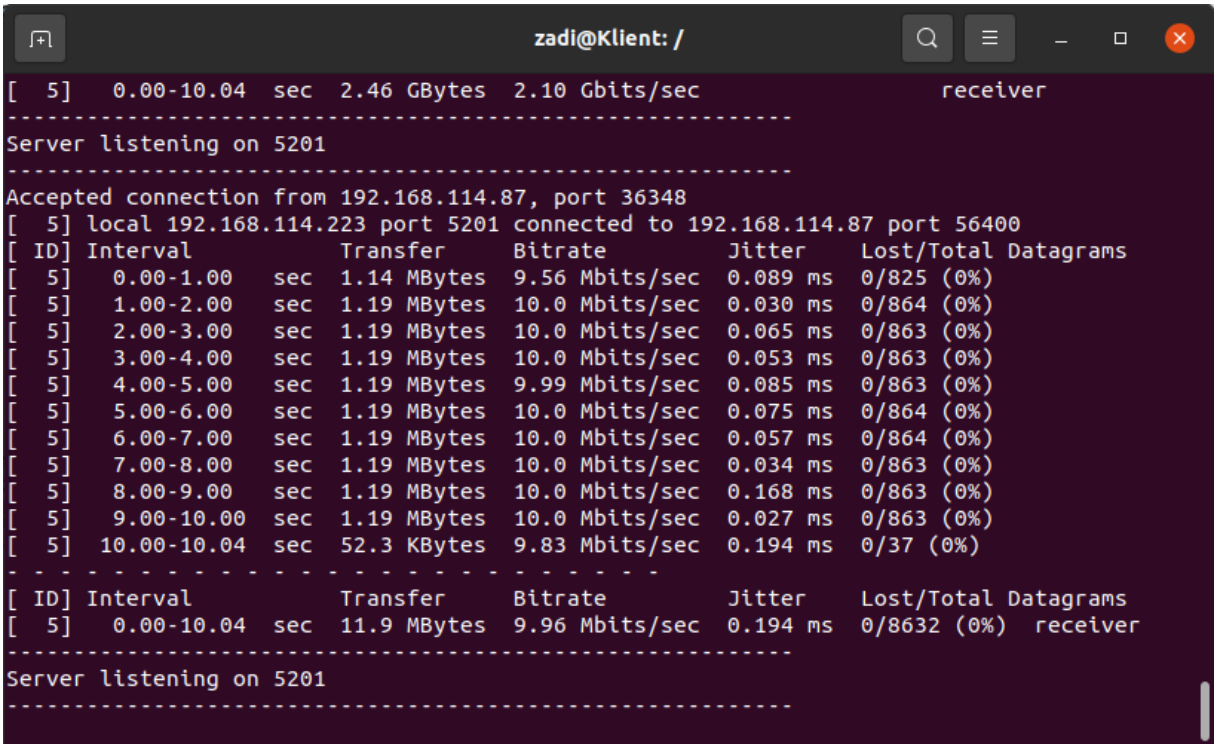
sudo tc qdisc add dev eth0 root netem delay 100ms 20ms

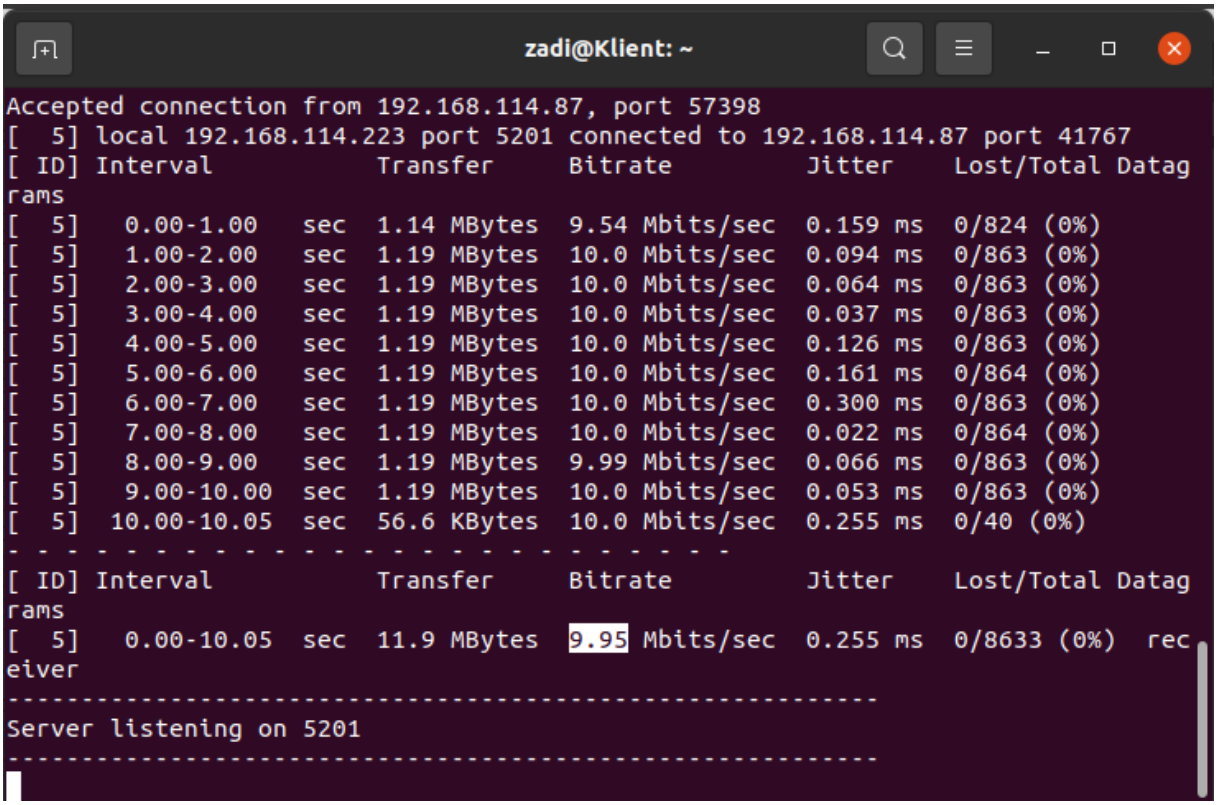
**Sprawdzamy Ping między VM czy opóźnienie wzrosło, i czy jest zmienne <zrzut ekranu>:**



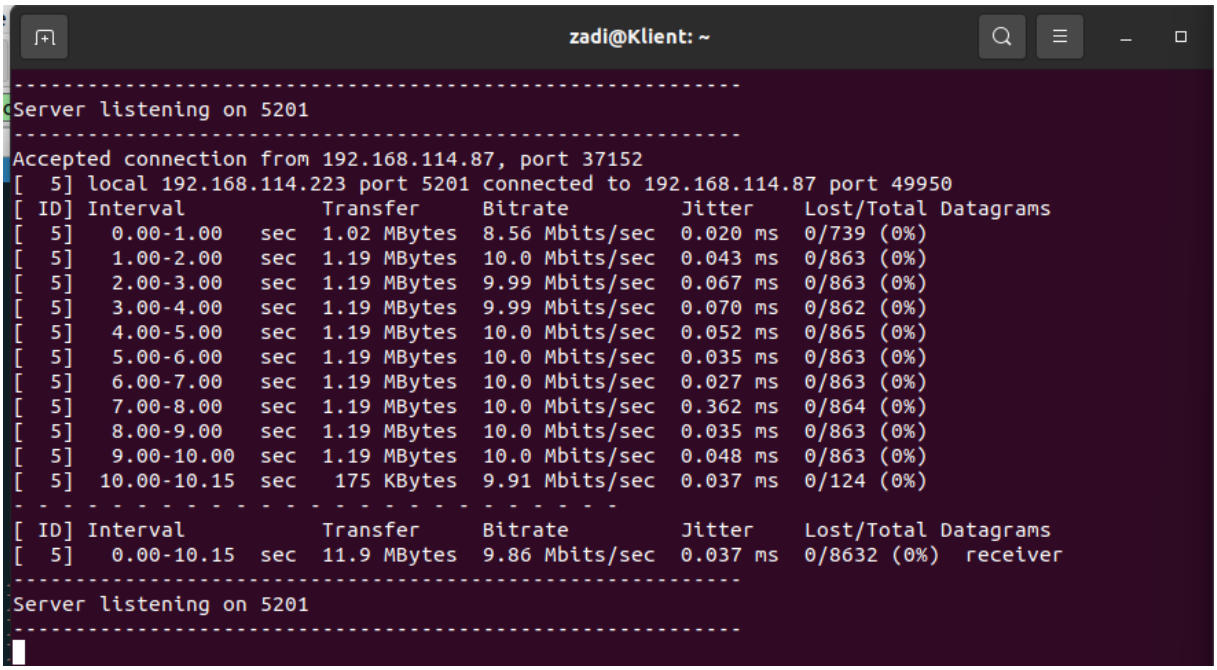
Opóźnienie wzrosło. Jest ono zmienne i waha się w przedziale między około 90 ms do 110 ms.

**Przeprowadzamy ponownie test iperf dla UDP (tylko UDP tym razem). Jaki jest efekt w stosunku do nominalnego wyniku? Jaki w stosunku do wyniku z opóźnieniem?**



si




Na pierwszym zdjęciu widać test iperf dla UDP dla stanu nominalnego (bez żadnych ustawionych opóźnień).

Na drugim zdjęciu widać test iperf dla UDP dla stanu z wprowadzonymi opóźnieniami (niezmiennymi).

Na trzecim zdjęciu widać test iperf dla UDP dla badanego stanu zdefiniowanego w poleceniu zadania (zmienne opóźnienia).

W stosunku do nominalnego wyniku, wartości badanego stanu zmieniły się. Wartość bitrate spadła o 0,1 Mbit/s. Zmalała natomiast wartość w kolumnie Jitter ze 0,194 ms na 0,037 ms. Jest to aż ponad pięciokrotnie mniejsza wartość.

W stosunku do wyniku dla niezmiennego opóźnienia (zdj. drugie), wartości również się zmieniły. Wartosć bitrate spadła o około 0,1 Mbit/s. Większa zmiana pojawiła się w wartości Jitter. Tutaj wartość Jitter zmalała prawie 7 krotnie.

W badanym przez nas stanie można zaobserwować najmniejszą wartość przepustowości jak również najmniejszą wartość Jittera.

**Komenda do usunięcia chaosu dla interfejsu:**

sudo tc qdisc del dev enp0s3 root