**Dokumentacja**

**Treść zadania:**

SK.TIN.2

Zaimplementować generator ruchu wykorzystujący standardowy stos TCP/IP w systemie Linux. Ruch powinien być generowany z wykorzystaniem protokołu UDP. Należy zaimplementować aplikację klienta i serwera. Należy przyjąć że serwer obsługuje w danym momencie tylko jednego klienta. Generator powinien umożliwiać jak najprecyzyjniejsze generowanie ruchu (pakietów UDP o określonej długości) z zadaną szybkością w obu kierunkach (szybkość dla obu kierunków powinna być ustalana niezależnie). Zarówno klient jak serwer powinien być zaimplementowany jako aplikacja wielowątkowa z dedykowanymi wątkami do generacji i odbierania pakietów (wątki te powinny być niezależnie od wątku głównego). Generator powinien mierzyć uzyskiwaną przepływność w obu kierunkach transmisji. Zarówno klient jak i serwer powinien umożliwiać wyświetlenie chwilowej przepływności uzyskiwanej w obu kierunkach transmisji np. wpisanie komendy print na konsoli klient powinno skutkować pobraniem statystyki z serwera i jej wyświetlenie na ekranie. Porównać dokładność działania zaimplementowanego generatora z aplikacją ***TRex*** (<https://trex-tgn.cisco.com/trex/doc/index.html>)

**Analiza problemu:**

Celem projektu było stworzenie aplikacji klienta i serwera, które przy pomocy protokołu UDP będą przesyłały do siebie pakiety losowych danych o określonej długości.

Zarówno klient, jak i serwer posiadają dedykowane wątki do generowania oraz odbierania pakietów. Ruch generowany jest z zadaną szybkością. Wątki generatora badają przepływność w obu kierunkach transmisji, a aktualną wartość można sprawdzić poprzez wpisanie w terminalu komendy ***print***. Zaimplementowany generator został porównany z aplikacją TRex pod kątem funkcjonalności oraz przepływności transmisji (sekcja znajduje się poniżej).

Aplikacje klienta i serwera współdziałają ze sobą, bez znaczenia czy uruchamiane są na tym samym, czy różnych urządzeniach.



**Stos technologiczny:**

System: Linux,

Język: C++

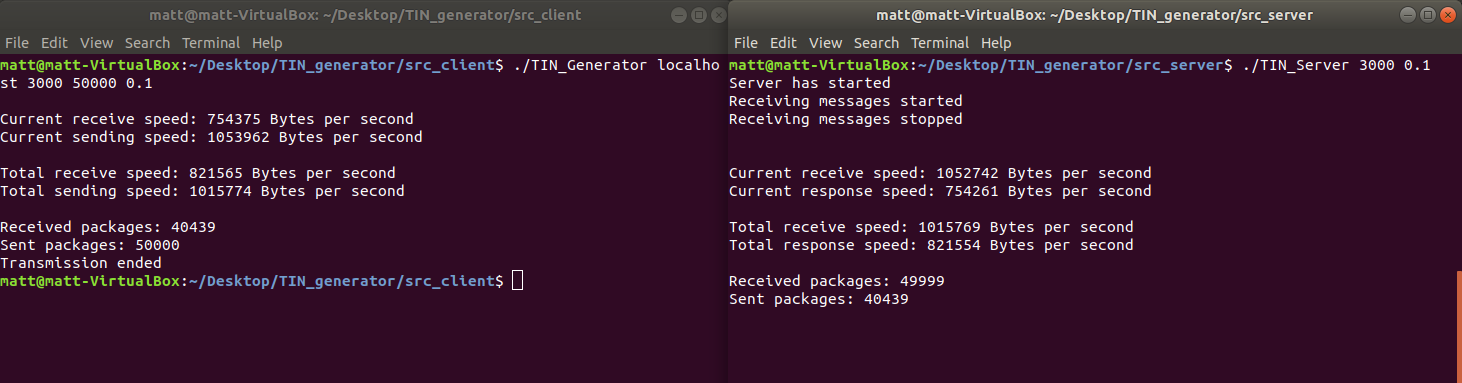
Biblioteki: Boost, Boost.Asio

**Uruchomienie projektu:**

W celu uruchomienia projektu należy:

1. W głównym folderze projektu wywołać komendę ***cmake .***
2. W podfolderach ***src\_server*** oraz ***src\_client*** wywołać komendę ***make***
3. Uruchomić wątek serwera poleceniem ***./TIN\_SERVER <numer portu> <czas odpowiedzi>***
4. Uruchomić wątek klienta poleceniem ***./TIN\_GENERATOR <nazwa hosta> <numer portu> <liczba pakietów do wysłania> <czas odpowiedzi>***, gdzie nazwa hosta jest nazwą lub adresem IP urządzenia, na którym został uruchomiony wątek serwera.

Punkty 1. oraz 2. odpowiadają za budowę elementów składowych projektu, natomiast punkty 3. oraz 4. powodują uruchomienie wątków serwera oraz klienta. Przykładowy efekt uruchomienia:



**Porównanie implementacji z aplikacją *TRex*:**

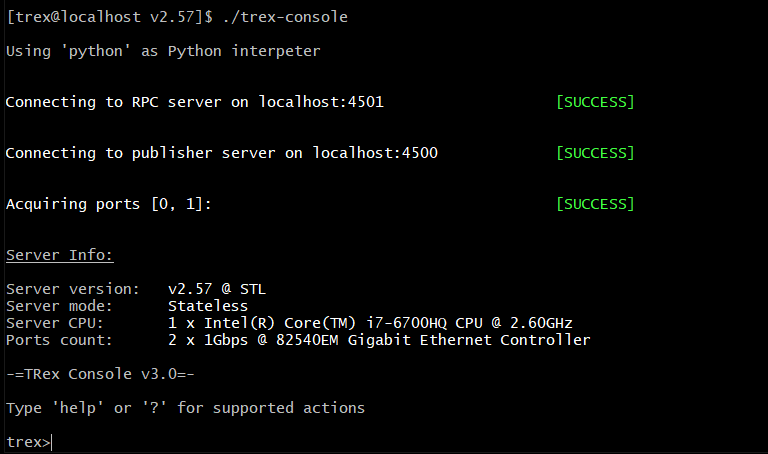
***TRex*** jest programem typu open source do generowania ruchu stworzonym przez wykorzystanie bibliotek DPDK. Może być generatorem zarówno stanowym (wykorzystywany do sprzętu typu hardware), jak i bezstanowym (wykorzystywane tam, gdzie istotne jest generowanie i przesyłanie strumieni pakietów podlegających testowaniu - ten tryb był istotny dla naszego projektu). Jest wykorzystywany m.in. do generowania ruchu na wysoką skalę przypominającego atak typu DDoS, do elastycznego testowania wszelakiej maści przełączników sieciowych czy do testów kontrolera obsługującego znaczne ilości klientów bądź serwerów.

W celu uruchomienia aplikacji ***TRex*** należy pobrać z oficjalnej strony *Cisco* obraz systemu *TRex Fedora* dla Dockera bądź VirtualBoxa i uruchomić go. Domyślnie jest w tym obrazie wersja programu v1.62, dlatego należy zaktualizować aplikację do najnowszej wersji (w projekcie to wersja v2.57). Istotnym jest aby aplikacja była w folderze o uprawnieniach administratora (może to być na przykład folder *tmp* w systemie Linux. Po takiej konfiguracji możemy uruchomić w tym folderze serwer ***TRexa***:

[trex@localhost v2.57]$ sudo ./t-rex-64 -i

Aby móc śledzić wykonanie generowania ruchu oraz testować aplikację wykorzystaliśmy możliwość połączenia do maszyny z serwerem poprzez wystawiony jej port 3022 dla protokołu SSH. Dla tak skonfigurowanego środowiska przeprowadzaliśmy testy dla przekrojowych rozmiarów pakietów i sprawdzaliśmy prędkości, z jakimi ***TRex*** przesyła pakiety, zarówno tylko z jednego portu o numerze ***0***, jak i obu dostępnych portów ***0*** i ***1***. Przykładowe skrypty napisane w Pythonie, które tworzą konfiguracje i ustalają ustawienia dla generatora są dostępne w folderze ***/stl*** aplikacji. Można je uruchamiać poprzez konsolę ***trex-console*** poleceniem:

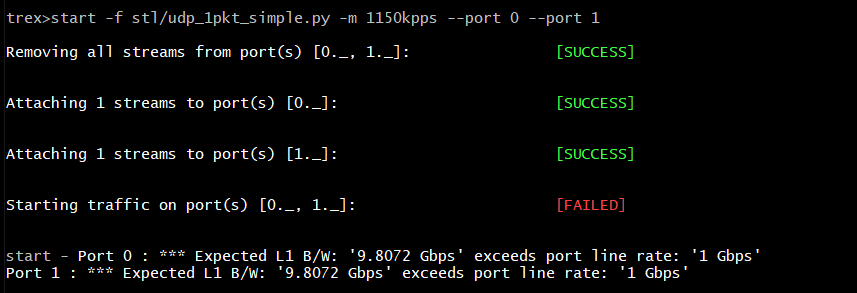
[trex@localhost v2.57]$ ./trex-console



Przykładowe uruchomienie skryptu:

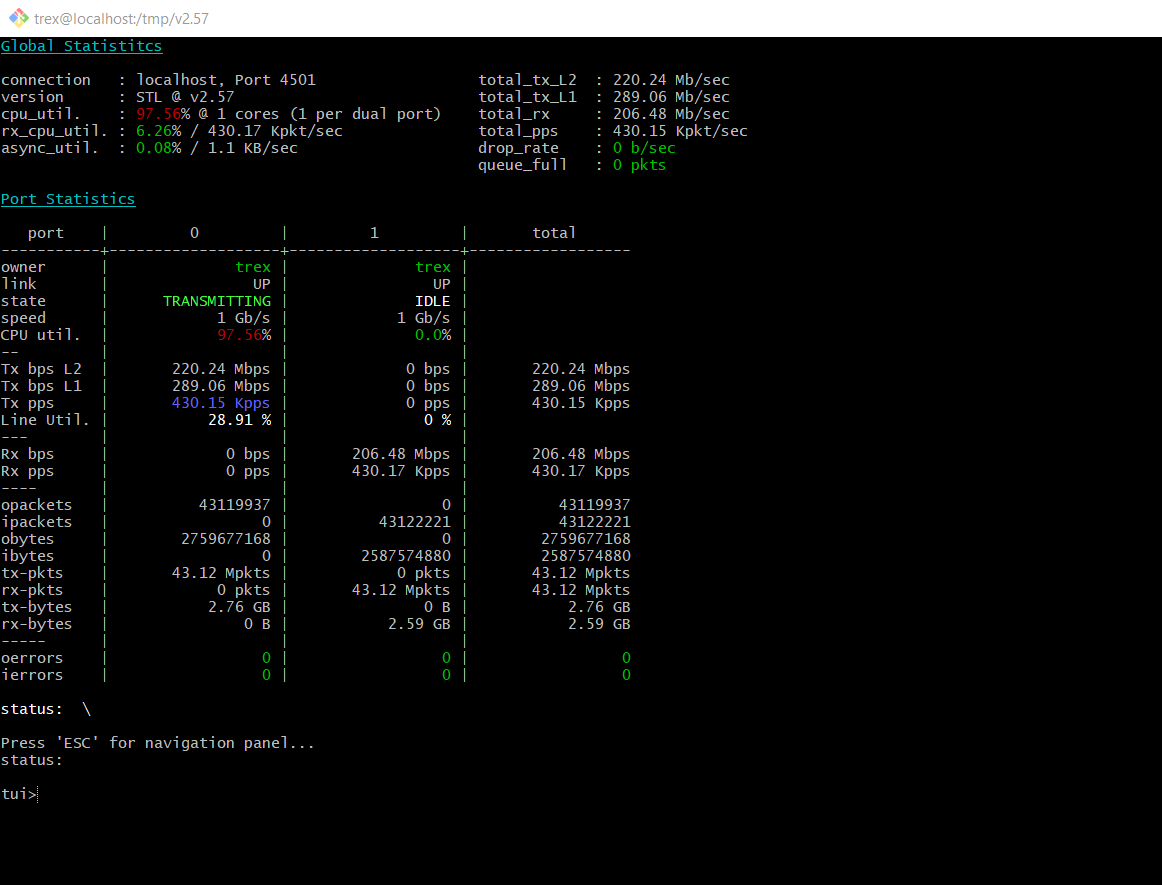
trex> start -f stl/udp\_1pkt\_simple.py -m 900kpps --port 0

gdzie poprzez flagę *-m* podajemy prędkość, z jaką mają być przesyłane pakiety oraz poprzez flagę *--port* na jakim porcie/portach ma zostać uruchomione generowanie ruchu. Należy zaznaczyć, że program wykrywa przed uruchomieniem (co widać na powyższym obrazku) wersję serwera, tryb, ilość rdzeni procesora oraz liczbę portów wraz z ich przepustowościami. Uruchomienie zostanie zablokowane, jeśli ilość danych do przesyłania na sekundę, jaką zadamy programowi (ilość bajtów w pakiecie \* zadana szybkość), przekroczy wartość dla danego portu (w tym wypadku 1Gbps). Przykładowy błąd przy uruchomieniu:

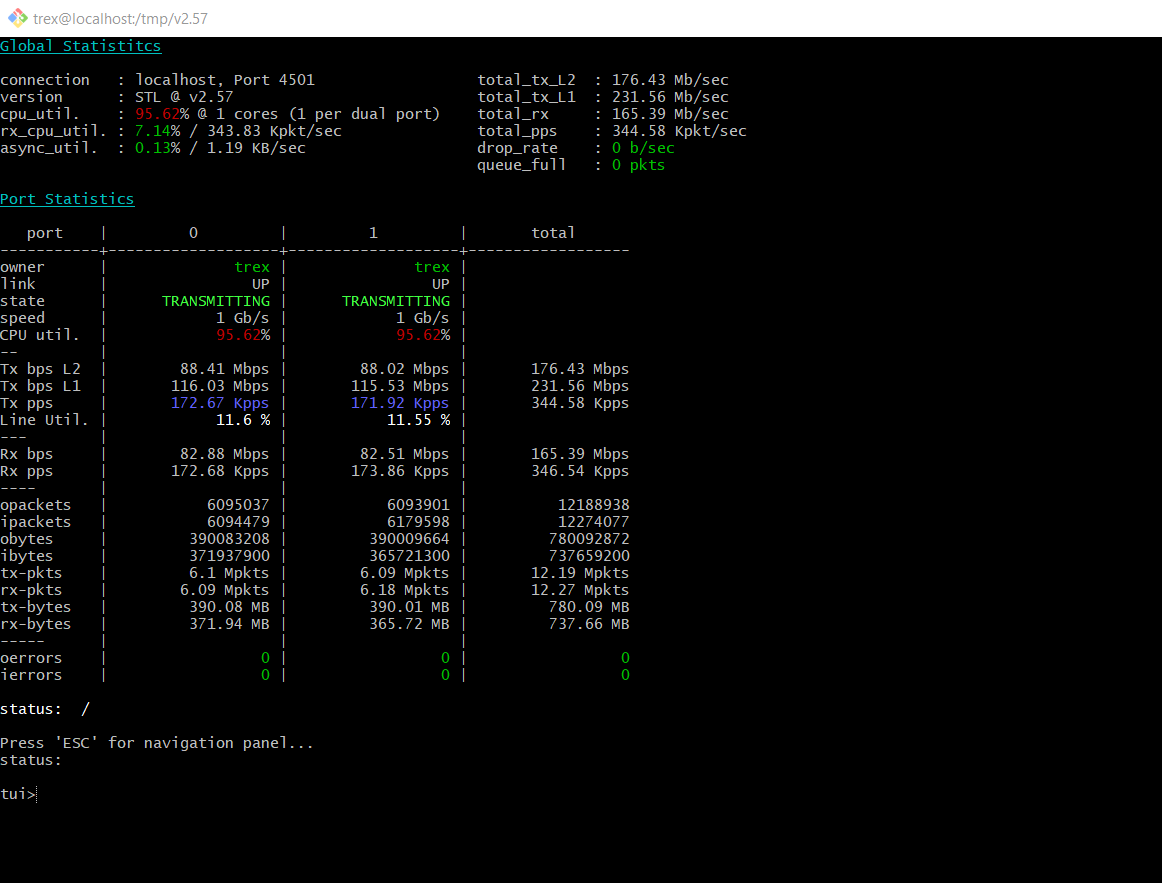


W związku z tym ograniczeniem wykonaliśmy testy dla przekrojowych wielkości pakietów - 1, 10, 100 oraz 1000 bajtów znaku ‘x’ (kod ASCII: 0x78). Wyniki przedstawiają się następująco (jest to okienko otwartego połączenia ssh wraz z widokiem ***tui***, który w konsoli ***trex-console*** pokazuje aktualne statystyki generowanego ruchu na serwerze):

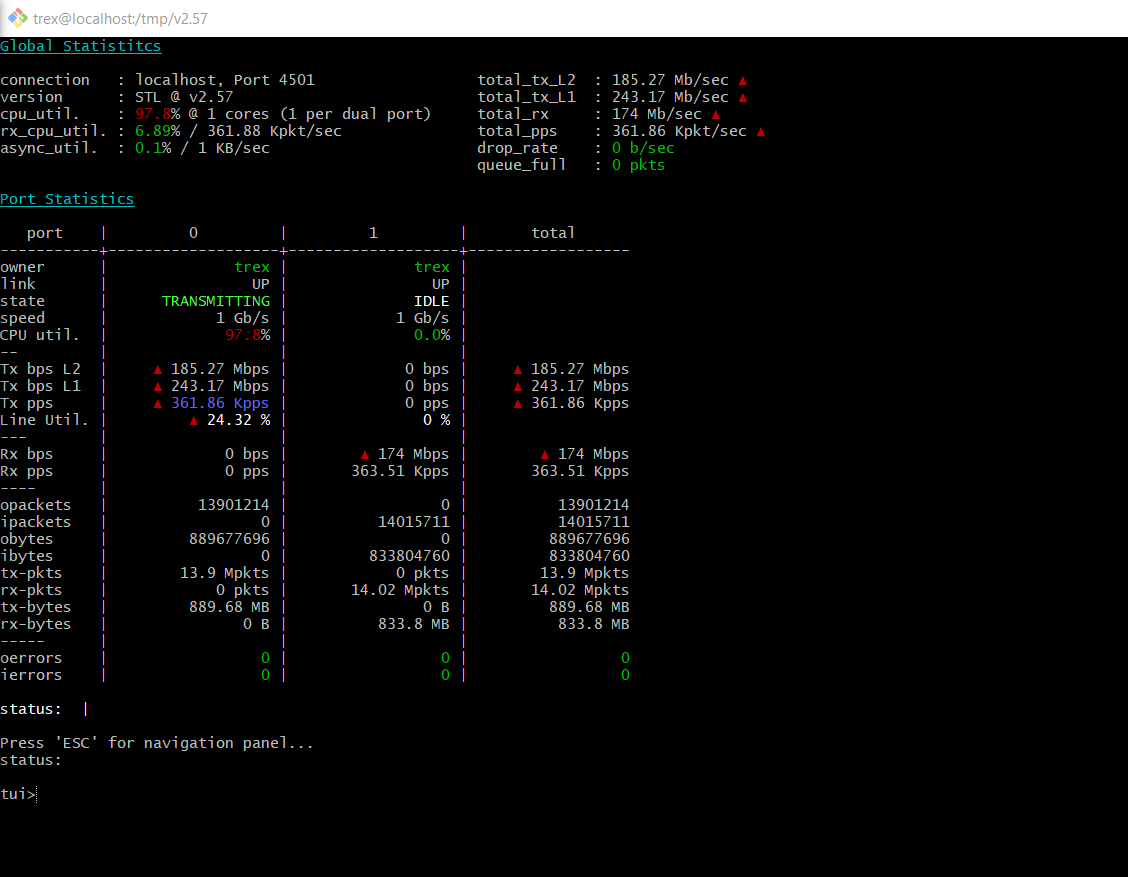
* 1 bajt znaku ‘x’:
  + port ***0***, zadana szybkość przesyłania - 999 Kpps

******

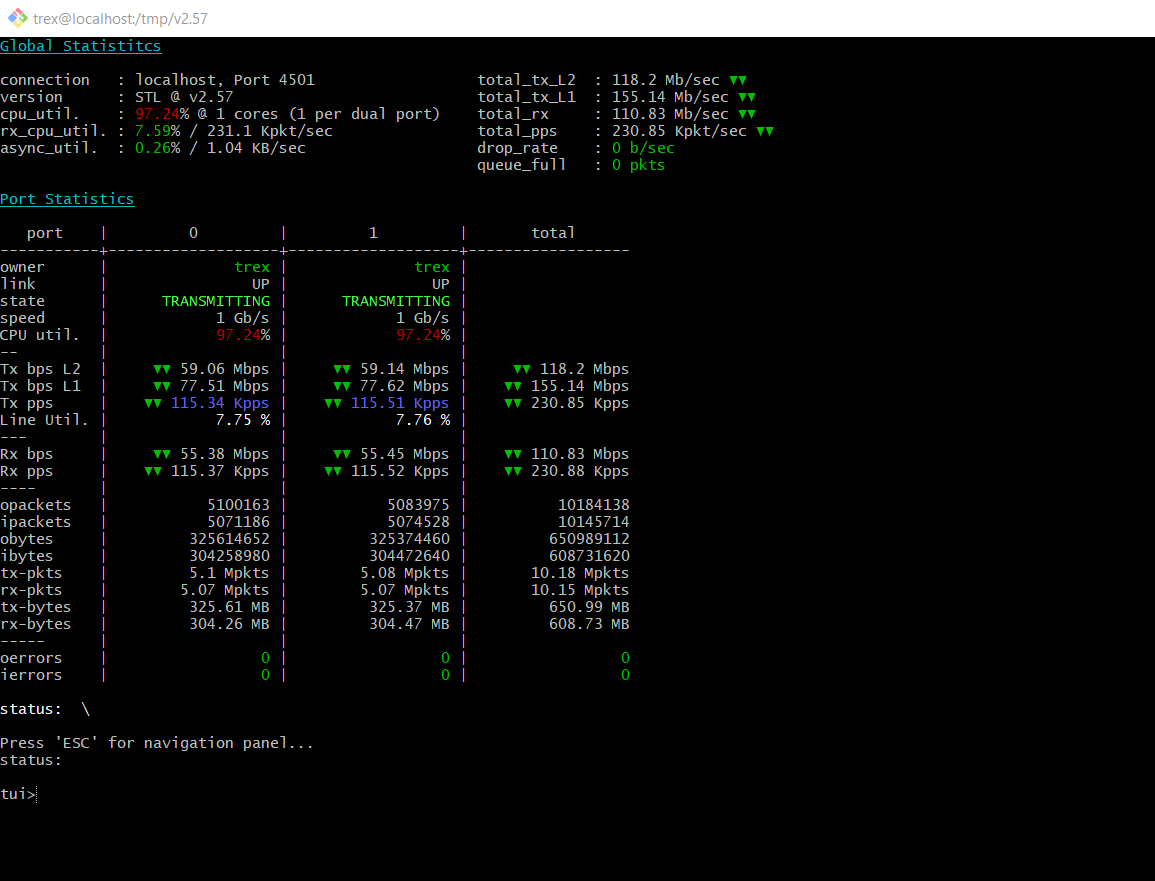
* + port ***0*** oraz port ***1***, zadana szybkość przesyłania - 999 Kpps

******

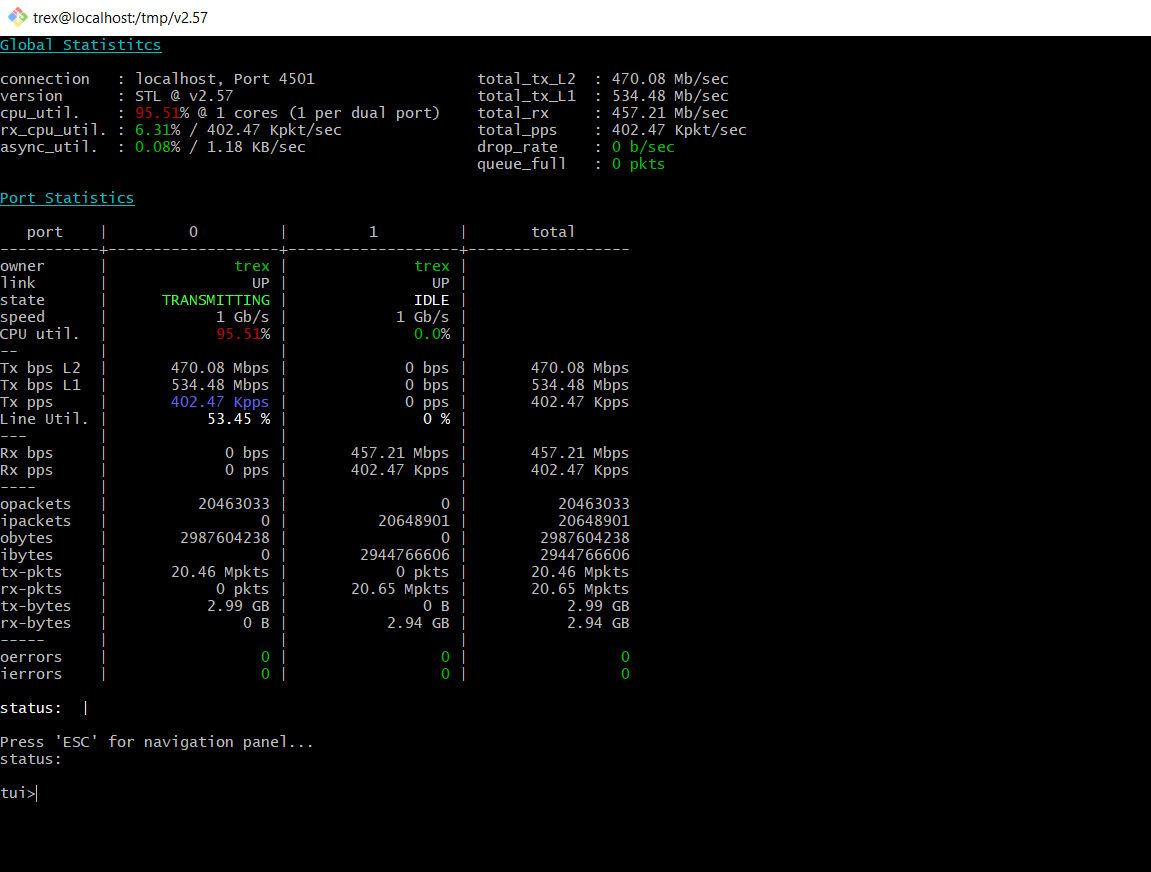
* 10 bajt znaku ‘x’:
  + port ***0***, zadana szybkość przesyłania - 999 Kpps

******

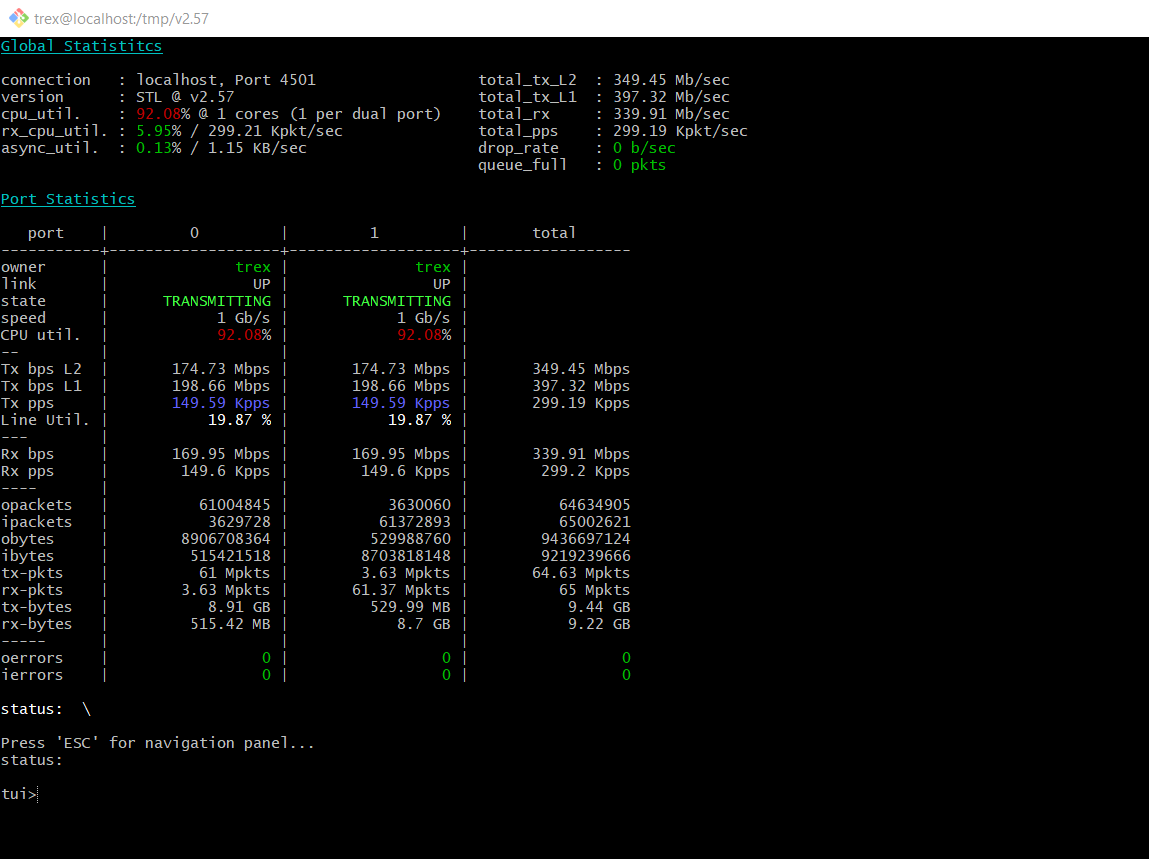
* + port ***0*** oraz port ***1***, zadana szybkość przesyłania - 999 Kpps

******

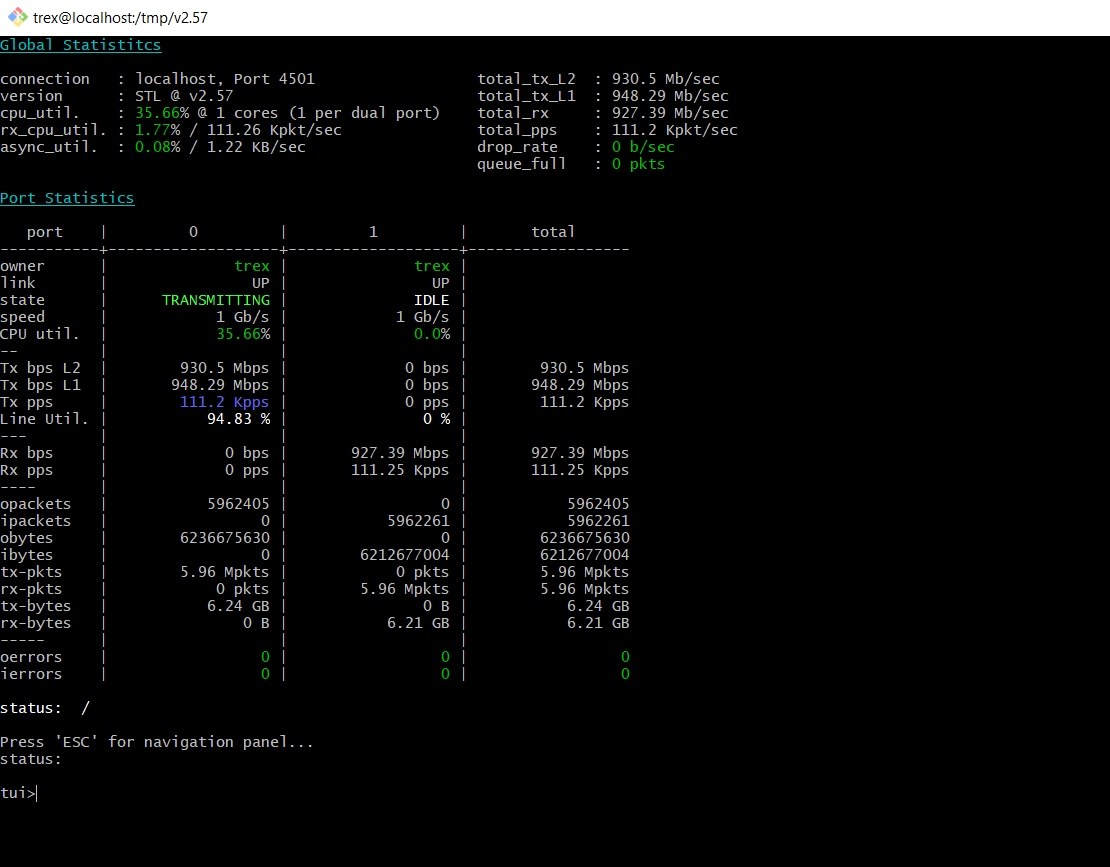
* 100 bajt znaku ‘x’:
  + port ***0***, zadana szybkość przesyłania - 750 Kpps

******

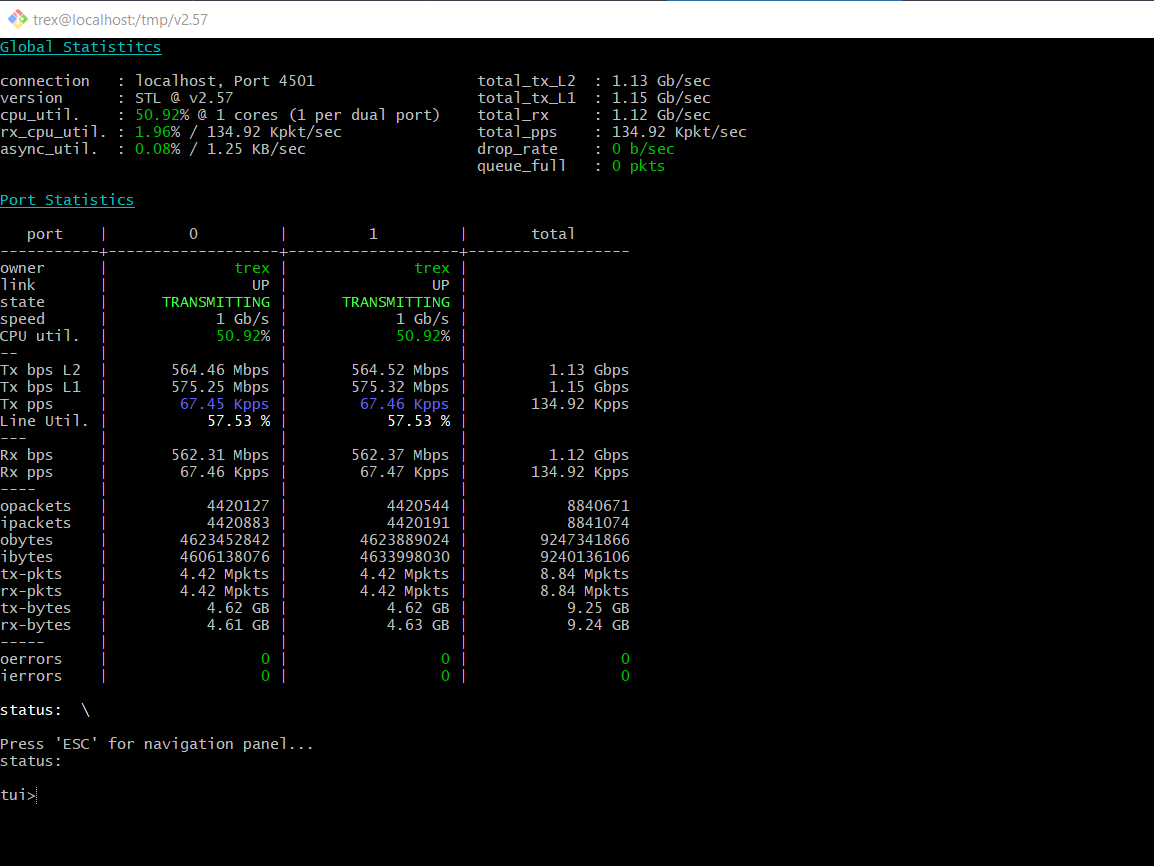
* + port ***0*** oraz port ***1***, zadana szybkość przesyłania - 750 Kpps

******

* 1000 bajt znaku ‘x’:
  + port ***0***, zadana szybkość przesyłania - 115 Kpps

******

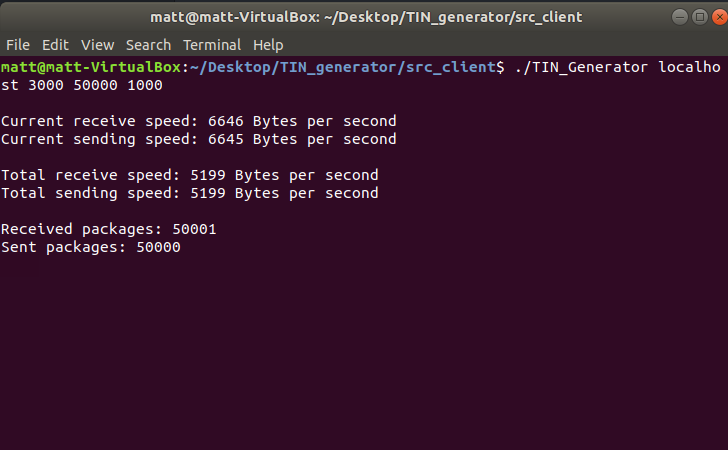
* + port ***0*** oraz port ***1***, zadana szybkość przesyłania - 115 Kpps



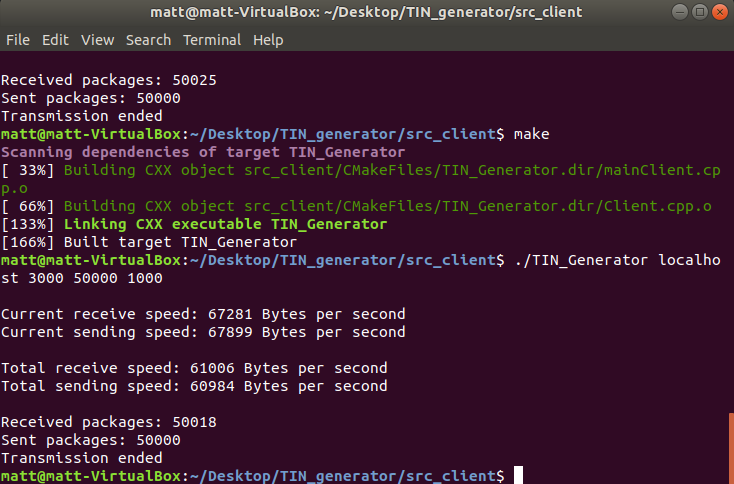
Możemy zauważyć, że różnica pomiędzy szybkością transmisji dla jednego portu w porównaniu do dwóch portów jest rzędu 100 Kpps (wyjątkiem są duże pakiety rzędu kilobajtów, dla których nie mogliśmy uruchomić szybszej transmisji w związku z ograniczeniem na porcie). Jednakże są to w każdym z przypadków setki, a dla dużych pakietów nawet tysiące Mbps przesyłane z portów.

Wyniki naszej implementacji generatora są znacznie niższe:

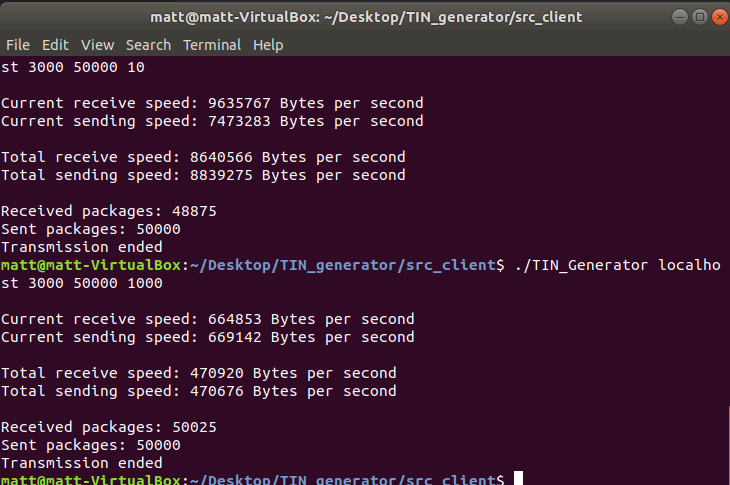
* rozmiar bufora - 10, ilość pakietów - 50000, interwał 1000μs



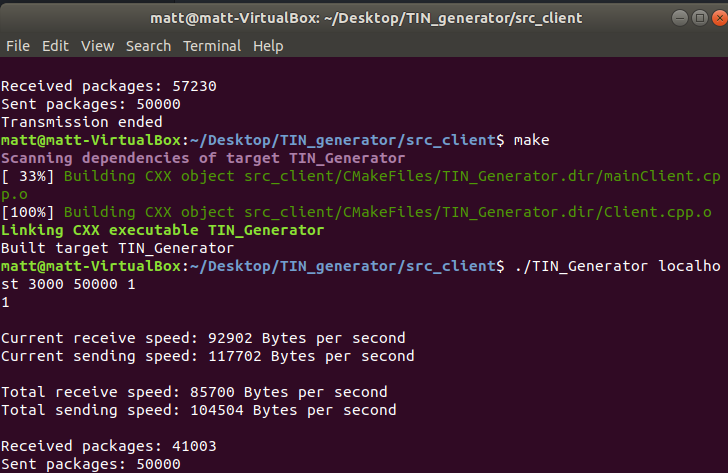
* rozmiar bufora - 100, ilość pakietów - 50000, interwał 1000μs



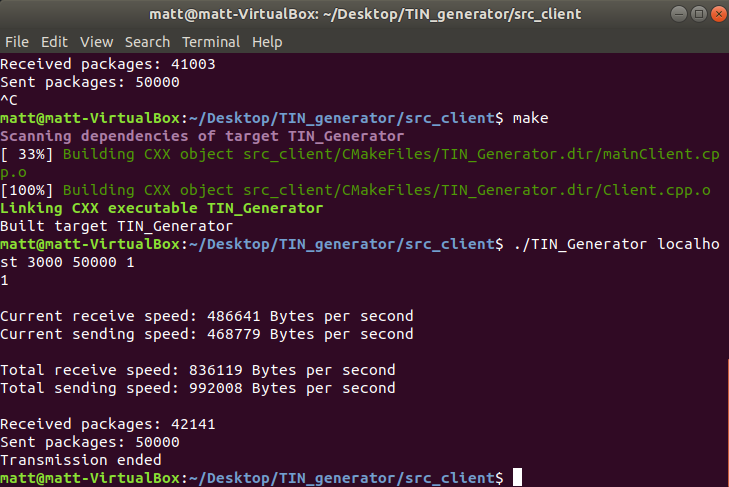
* rozmiar bufora - 1000, ilość pakietów - 50000, interwał 1000μs



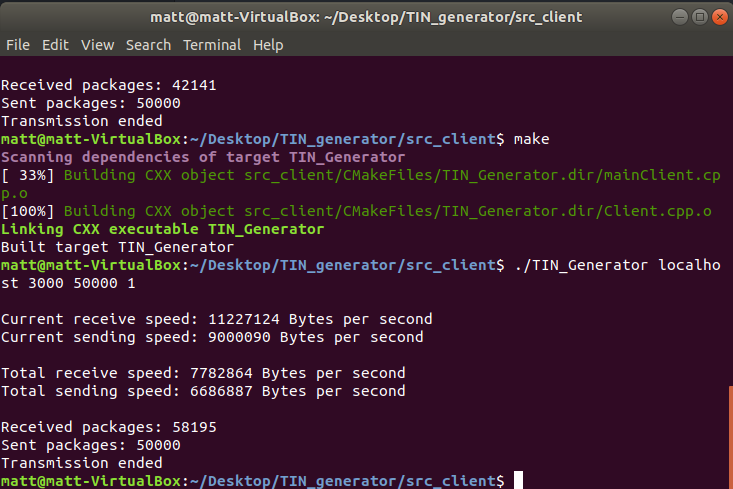
* rozmiar bufora - 10, ilość pakietów - 50000, interwał 1μs



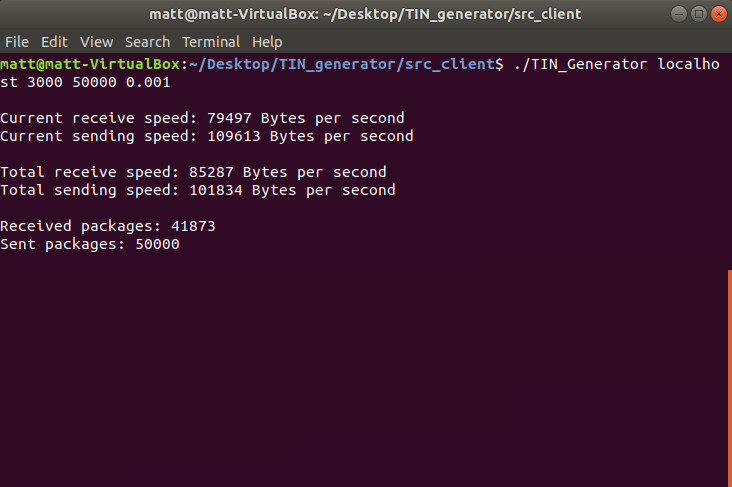
* rozmiar bufora - 100, ilość pakietów - 50000, interwał 1μs



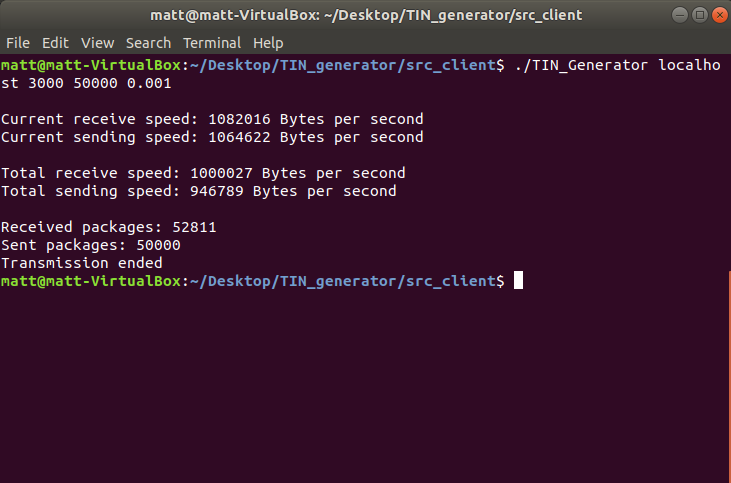
* rozmiar bufora - 1000, ilość pakietów - 50000, interwał 1μs



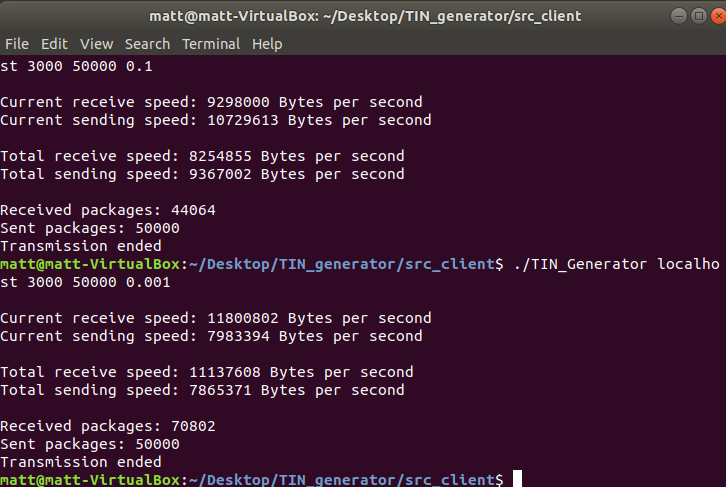
* rozmiar bufora - 10, ilość pakietów - 50000, interwał 0,001μs



* rozmiar bufora - 100, ilość pakietów - 50000, interwał 0,001μs



* rozmiar bufora - 1000, ilość pakietów - 50000, interwał 0,001μs



Jak widać z powyższych przebiegów, dla przesyłania 50000 pakietów w interwale mniejszym od 1 mikrosekundy (co spowoduje wysyłanie i odbieranie wiadomości niemalże non-stop, ponieważ taki sam był czas odpowiedzi serwera), gdzie każdy z pakietów to bufor o rozmiarze 1000, uzyskujemy zaokrąglając wartość rzędu 10 milionów bajtów na sekundę => 10^7 Bps = 9,5 MBps. Zmniejszanie interwału nie zmienia tej wartości, wciąż jest to około 10 MB przesyłanych w ciągu sekundy. Dla porównania ***TRex*** dla takiego samego bufora (jednocześnie będąc ograniczonym przez port do przesyłania 1Gb na sekundę) osiąga prędkość przesyłania około 120MB na sekundę, czyli ok. 12 razy więcej niż w naszym projekcie, co jest znaczącą różnicą.

**Wnioski:**

Wykonana implementacja generatora ruchu spełnia swoje zadanie, przesyłając strumień pakietów o rozmiarze kilku megabajtów na sekundę. W zestawieniu z profesjonalnie stworzonym narzędziem ***TRex*** wyniki naszego projektu są znacznie niższe, ale jednocześnie zadowalające. Jest to dobry punkt wyjścia do dalszych prac i optymalizacji nad generowaniem ruchu poprzez protokół UDP.