Testowanie wydajności sieci

Programowanie sieciowe - projekt

Projekt wykonany przez **zespół nr 27** w składzie:
- Alicja Kubiszyn
- Sara Fojt
- **Dominika Ferfecka (lider zespołu)** - dominika.ferfecka.stud@pw.edu.pl
- Mykhailo Marfenko

Data przekazania dokumentacji wstępnej: 27.12.2023
Data przekazania dokumentacji finalnej: 24.01.2024

Temat projektu: Testowanie wydajności sieci

1. Treść zadania

Projekt testera wydajności (przepustowości) sieci z użyciem protokołu UDP. Narzędzie składa się z serwera i klienta. Parametrem wywołania klienta określona będzie wielkość pakietu danych oraz adres i port serwera.
Pod koniec testu wyświetlane będą statystyki dotyczące prędkości upload i download oraz utraty pakietów i opóźnienie. Testy powinny być także przeprowadzone w rzeczywistych warunkach sieci rozległych w postaci połączenia z serwerem umieszczonym, np. na maszynie wirtualnej w usłudze chmurowej.
Zobacz: narzędzie iperf w systemach Linux.

2. Interpretacja treści zadania

Projekt ma na celu opracowanie narzędzia do testowania wydajności sieci przy użyciu protokołu UDP. Głównym celem jest zbadanie przepustowości sieci poprzez analizę prędkości uploadu i downloadu, utraty pakietów oraz opóźnienia w transmisji danych. Narzędzie będzie składać się z dwóch głównych komponentów: serwera i klienta, działających w oparciu o określone parametry.
Inspiracją projektu jest narzędzie **iperf** w systemach Linux.

Kluczowe elementy projektu:
1. Dwa komponenty: serwer i klient.
2. Konfiguracja klienta: Wielkość pakietu danych, tryb testu, adres i port serwera.
3. Statystyki: Prędkość uploadu i downloadu, utrata pakietów, opóźnienie.
4. Testy w rzeczywistych warunkach: Użycie serwerów w chmurze lub maszyn wirtualnych.

3. Opis funkcjonalny

- Interfejs użytkownika i interakcja z systemem
    - Aby uruchomić klienta, użytkownik wprowadza komendę w terminalu lub interfejsie graficznym, określając opcje konfiguracji, takie jak wielkość pakietu danych, tryb testu, adres IP oraz port serwera.
    - Możliwość wyboru trybu testu, np. ciągły, jednorazowy, lub z określonym czasem trwania
- Reakcje systemu na działania użytkownika
    - Po wprowadzeniu komend, klient nawiązuje połączenie z serwerem i rozpoczyna transmisję danych
    - W czasie rzeczywistym monitorowanie przepustowości, zliczanie utraconych pakietów i mierzenie opóźnień.
    - Wyświetlane statystyki na ekranie użytkownika
- Wyniki działania systemu
    - Po zakończeniu testu, użytkownik otrzymuje szczegółowy raport, zawierający prędkość uploadu i downloadu, procentową utratę pakietów, średnie i maksymalne opóźnienie, oraz inne istotne metryki.
    - [Opcjonalne] Możliwość wizualizacji wyników z użyciem biblioteki *matplotlib* w Pythonie
- Działanie w różnych wielkościach sieci
    - Narzędzie będzie przystosowane do testowania w różnych środowiskach sieciowych, od lokalnych sieci LAN do sieci rozległe WAN.
- Obsługa błędów i wyjątki
    - W przypadku problemów z w połączeniu lub innych problemów sieciowych, system będzie odpowiednio reagował i rejestrował te zdarzenia. System zapewni informacje zwrotne w przypadku niepowodzenia nawiązania połączenia lub innych błędów konfiguracyjnych.

4. Opis i analiza poprawności stosowanych protokołów komunikacyjnych

UDP - lekki protokół transportu danych. Jest to jeden z podstawowych elementów stosu protokołów internetowych, na których działa Internet.

Format pakietu:

Struktura nagłówka UDP

	Bity 0-15	Bity 16-31
0	Port nadawcy	Port odbiorcy
32	Długość	Suma kontrolna
64	Dane	

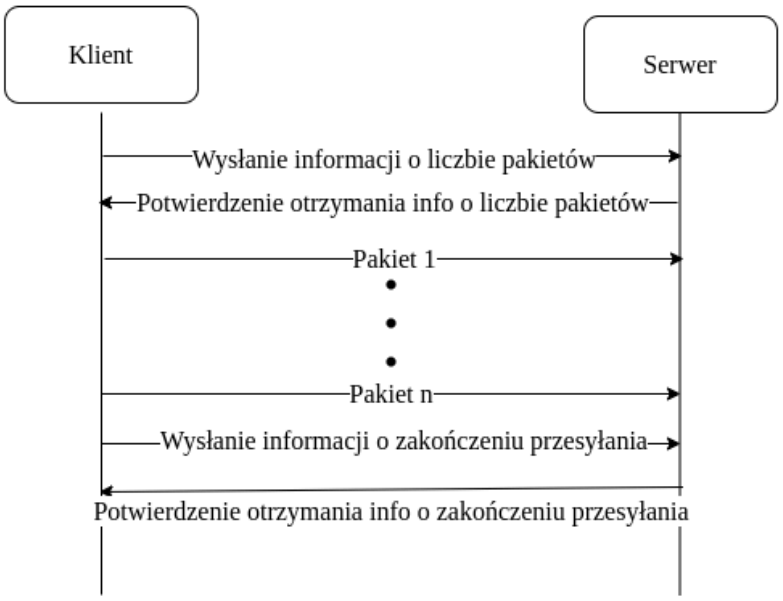
Pierwsze cztery bajty nagłówka UDP przechowują numery portów dla miejsca źródłowego i docelowego.
Następne dwa bajty reprezentują długość segmentu. Maksymalna długość segmentu UDP wynosi 65 535 bajtów.

Suma kontrolna:
Następne dwa bajty nagłówka to suma kontrolna, która służy do upewnienia się, że dane nie zostały uszkodzone.
Przed wysłaniem pakietu nadawca oblicza sumę kontrolną na podstawie danych w segmencie i umieszcza ją w polu suma kontrolna. Po odebraniu danych odbiorca ponownie oblicza sumę kontrolną i porównuje ze sobą dwie wartości. Jeśli nie są one równe oznacza to, że dane zostały uszkodzone.

Dane:
W segmencie Dane znajdują się właściwe dane przesyłane do odbiorcy.

W naszym projekcie użyjemy biblioteki struct do przygotowania pakietów do wysłania przez sieć i odczytania przez odbiorcę. Przekażemy do structa wartość timestamp, na podstawie której będzie można obliczyć opóźnienie wysyłania pakietów. Przed uruchomieniem testowania musi być zapewniona synchronizacja czasu.

Najpierw wyślemy serwerowi informację o ilości pakietów, a serwer potwierdzi odebranie tej informacji.
Następnie klient wyśle odpowiednią liczbę pakietów.
Po przesłaniu wszystkich pakietów klient wyśle informacje do serwera, a serwer potwierdzi otrzymanie ich i na podstawie liczby otrzymanych pakietów i informacji, ile klient planował ich wysłać, serwer będzie mógł obliczyć ile pakietów zostało utraconych.
Zależności czasowe przy wymianie komunikatów:



5. Planowany podział na moduły i strukturę komunikacji między nimi

W podstawowej wersji naszego programu będzie podział na dwa główne moduły:

- Klient - odpowiadający za generację i nadanie danych
- Serwer - otrzymujący dane od klientów

Parametry klienta:

- Adres IP serwera (do którego wysyłamy pakiety)
- Numer portu serwera
- Wielkość pakietu danych (określona w bajtach)

W celu zbadania przepustowości łącza w sieci rozproszonej, nasz serwer umieścimy w maszynie wirtualnej. Testy przeprowadzimy kilkukrotnie, umieszczając serwer w różnych miejscach na świecie, w celu najlepszego porównania wyników.

Stworzymy również interfejs użytkownika, dzięki któremu użytkownik będzie mógł za pomocą komend w terminalu, podawać parametry dla klienta i serwera.

Do komunikacji między klientem a serwerem będziemy używać modułu struct. Klient przygotowane dane w pythonie zawierające tekst do wysłania oraz numer pakietu, za pomocą struct.pack() zamieni je na binarne. Tak przygotowane dane binarne zostaną przesłane do serwera. Po otrzymaniu ich serwer za pomocą struct.unpack() zamieni je na typy struktur w Pythonie.

Diagram klient - serwer

Powyżej jest przedstawiona podstawowa wersja naszego testera przepustowości sieci, zawierająca komunikację między klientem a serwerem.

Diagram systemu rozproszonego

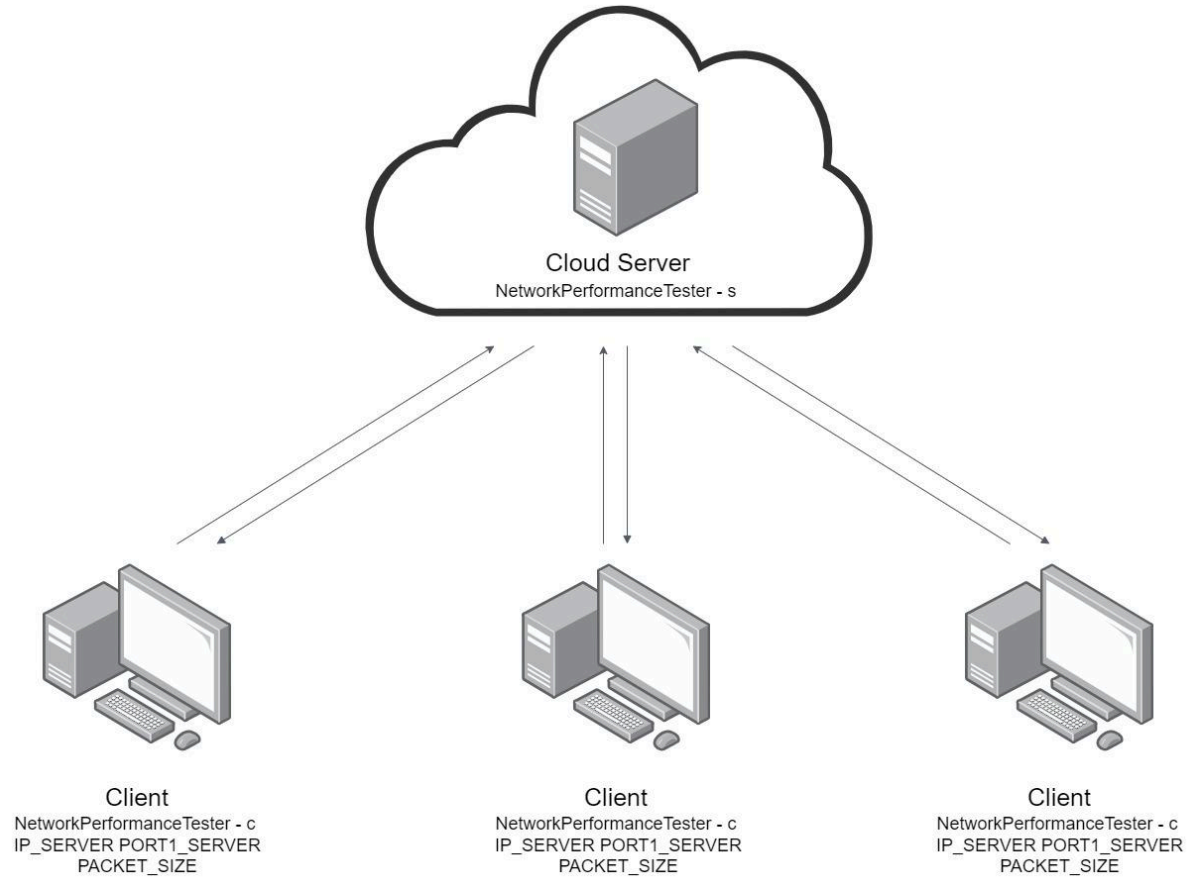


Diagram systemu rozproszonego przedstawia bardziej zaawansowany test naszego protokołu, serwer będzie umieszczony w chmurze, aby przetestować działanie sieci w różnych regionach świata.
Powyższy diagram przedstawia też potencjalne rozwiązanie do zaimplementowania w przyszłości, które umożli przesyłanie danych równocześnie od wielu klientów. Do takiego mechanizmu również musiałaby być zaimplementowana współbieżność. W naszym rozwiązaniu na razie skoncentrujemy się na stworzeniu testu dla jednego klienta.

6. Zarys koncepcji implementacji (język, biblioteki, narzędzia, etc.)

Nasza implementacja składa się z dwóch programów - serwera oraz klienta. Instancje klienta mogą być uruchamiane wielokrotnie na kilku maszynach, znajdujących się w różnych sieciach. Zarówno klient jak i serwer są stworzone w języku Python.

Do przeprowadzenia testów w rzeczywistych warunkach tj. sieci rozległej, użyjemy swoich maszyn lokalnych, kontenerów na maszynie bigubu, lecz w innej sieci lub maszyny wirtualnej uruchomionej w usłudze chmurowej Azure.

Biblioteki z jakich korzystamy to:

- argparse - przetwarzanie argumentów podanych przy uruchamianiu programów
- socket - tworzenie połączeń UDP między klientem i serwerem
- struct - wysyłanie i odbieranie pakietów informacji np. dane, numer pakietu oraz timestamp wysyłki pakietu
- time - obliczanie opóźnień w wysyłaniu pakietu
- matplotlib - tworzenie graficznej reprezentacji statystyk otrzymanych z testów programu

Dodatkowo będziemy wzorować się na narzędziu iperf, którego kod do trzeciej wersji jest dostępny na platformie github. Głównie jednak będziemy używać tego narzędzia do porównywania wyników uzyskanych za pomocą naszej implementacji z wynikami wygenerowanymi przez iperf.

Sposoby implementacji mierzenia wyników:
- prędkość upload i download - wyliczenie na podstawie analizy rozmiarów i ilości wysłanych / odebranych pakietów w określonym czasie (suma wielkości pakietów / czas)
- ilość zgubionych pakietów - porównanie liczbę wysłanych i otrzymanych pakietów. Rozwiązanie to będzie się opierać na dodatkowych pakietach rozpoczynających i kończących komunikację.
- opóźnienie pakietów - przed wysyłką pakietu zostanie stworzony timestamp i następnie spakowany do struct wraz z danymi. Po wysyłce do serwera i rozpakowaniu, tworzony jest nowy timestamp i wartości obu z nich są odejmowane. Wynikiem tej operacji będzie nasze opóźnienie w wysyłce pakietów. Przed porównaniem musi nastąpić synchronizacja.

Projekt finalny

7. Opis najważniejszych rozwiązań funkcjonalnych wraz z uzasadnieniem (opis protokołów, struktur danych, kluczowych funkcji)

Protokół połączenia

1. Klient wysyła pakiet z informacją o ilości danych i oczekuje na ACK od serwera
    a. jeśli nie otrzyma ACK w ciągu 2 sekund, wysyła pakiet ponownie
2. Po potwierdzeniu pierwszego pakietu, klient wysyła wszystkie zaplanowane pakiety bez potwierdzeń
3. W przypadku ostatniego pakietu zmienia się flaga is_last wysyłana w strukturze paczki na 1 i serwer po zarejestrowaniu tej flagi wysyła ACK otrzymania ostatniej paczki

4. Klient powtarza wysyłanie ostatniej paczki, aż do otrzymania ACK od serwera o jej otrzymaniu

Struktura danych pakietu

- pierwszy pakiet - struct składający się z:
    - int - liczba paczek - pobrana w argumencie od użytkownika
- reszta pakietów - struct BigEndian składający się z:
    - w przypadku obliczania opóźnienia
        - bit - flaga is_last - flaga oznaczająca ostatni pakiet jako 1
        - double float - timestamp - moment czasowy, w którym pakiet został wysłany
        - sekwencja bajtów - random_bytes - losowo generowane bajty jako wiadomość
    - bez obliczania opóźnienia
        - bit - flaga is_last - flaga oznaczająca ostatni pakiet jako 1
        - sekwencja bajtów - random_bytes - losowo generowane bajty jako wiadomość

Kluczowe funkcje

client.py

- measure_one_packet_speed
    - params:
        - one_packet_start_time - timestamp rozpoczęcia uploadowania pakietu
        - one_packet_end_time - timestamp zakończenia uploadowania pakiet
        - message - dane, które są wysyłane w pakiecie
        - i - numer pakietu
    - funkcja oblicza prędkość uploadowania jednego pakietu
- measure_total_speed
    - params:
        - total_start_time - timestamp rozpoczęcia wysyłania pierwszej paczki
        - total_bytes_sent - ilość wszystkich przesłanych bajtów
- process_ack
    - params:
        - s - socket służący do otwartej komunikacji
        - message - dane do wysłania w danej wiadomości
        - server_address - adres serwera, z którym się komunikujemy
        - is_last - flaga do rozpoznania pierwszego od ostatniego pakietu
        - send_time - timestamp rozpoczęcia wysyłania (w przypadku liczenia opóźnienia)
    - funkcja ponawiająca wysyłanie wiadomości w przypadku braku otrzymania ACK od serwera i obsługująca otrzymanie go

server.py

- packet_loss
    - params:
        - received_packets - liczba otrzymanych pakietów od danego klienta

        - packets_quantity - oczekiwana liczba pakietów do otrzymania od klienta
    - funkcja obliczająca ilość zgubionych paczek
- measure_one_packet_speed
    - analogicznie do funkcji o tej samej nazwie w client.py, lecz mierzy download
- measure_total_speed
    - analogicznie do funkcji o tej samej nazwie w client.py, lecz mierzy download
- start_communication
    - params:
        - s - otwarty socket do komunikacji z danym klientem
    - funkcja obsługuje otrzymanie pierwszego pakietu i wysłanie ACK do klienta
- handle_communication
    - params:
        - s - otwarty socket do komunikacji z danym klientem
        - packets_quantity - oczekiwana liczba paczek do otrzymania
    - funkcja obsługuje komunikację z klientem - otrzymywanie paczek i wysłanie ACK do ostatniego pakietu oraz wywołuje funkcje do obliczania statystyk połączenia

8. Szczegółowy opis interfejsu użytkownika

Uruchomienie serwera

python server.py <port>

Przykładowe wywołanie:
python server.py 8080

Uruchomienie klienta lokalnego

python ./client_local.py <ip_serwera> <port> <ilość_paczek>

Przykładowe wywołanie:
python ./client_local.py 172.162.233.92 8080 10

Uruchomienie klienta zewnętrznego

python ./client_external.py <ip_serwera> <port> <ilość_paczek>

Przykładowe wywołanie:
python ./client_external.py 172.162.233.92 8080 10

Uruchomienie statystyk

Włączenie statystyki upload download można wykonać poprzez dodanie flagi **--speed_statistic** zarówno do klienta jak i serwera

Przykładowe wywołania:
python server.py 8080 --speed_statistic
python ./client_local.py 10.0.0.4 8080 10 --speed_statistic

Włączenie statystyki opóźnienia pakietów można wykonać poprzez dodanie flagi **--delay** zarówno do klienta jak i serwera

Włączenie statystyki gubienia pakietów można wykonać poprzez dodanie flagi **--packet_loss_statistic**

Przykładowe wywołania:
python server.py 8080 --delay
python ./client_local.py 10.0.0.4 8080 10 --delay

9. Postać wszystkich plików konfiguracyjnych, logów

Statystyka prędkości upload i download

Prędkość wysyłania jak i odbierania danych jest mierzona dla klienta i serwera.
Mierzymy zarówno prędkość przesyłu pojedynczej paczki jak i całość procesu nawiązywania połączenia i wysłania wszystkich paczek.

Statystyka opóźnienia pakietów

Opóźnienie pakietów jest mierzone zarówno na stronie klienta, jak i serwera. Analiza ta obejmuje ocenę czasu, który upływa od momentu wysłania pakietu przez klienta do momentu jego odbioru przez serwer, a także czas powrotu potwierdzenia do klienta. Dla każdej wysłanej paczki mierzymy czas, jaki upłynął od momentu jej wysłania do momentu otrzymania odpowiedzi (ACK) od serwera. Pozwala to na obliczenie czasu podróży pojedynczego pakietu (RTT - Round Trip Time).

Statystyka gubienia pakietów

Gubienie pakietów jest obliczane po stronie serwera. Pierwszy pakiet jaki dostaje od klienta zawiera informację ile klient łącznie zamierza przesłać pakietów. Następnie zliczana jest faktyczna ilość pakietów, które otrzymał serwer. Na tej podstawie obliczana jest ilość pakietów zgubionych.

10. Opis wykorzystanych narzędzi, itp.

Do wykonania projektu użyliśmy Pythona oraz jego dodatkowym bibliotek, m.in. argparse, time, struct. Dodatkowo skorzystaliśmy z platformy Azure, aby zrealizować rozproszenie sieci. Maszyna z serwerem oraz maszyna z klientem lokalnym znajdowały się na serwerach Switzerland North, zaś klient zewnętrzny był postawiony na serwerach East US. Pozwoliło nam to zaobserwować różnicę w prędkości przesyłu danych na różne odległości.

11. Opis testów i wyników testowania

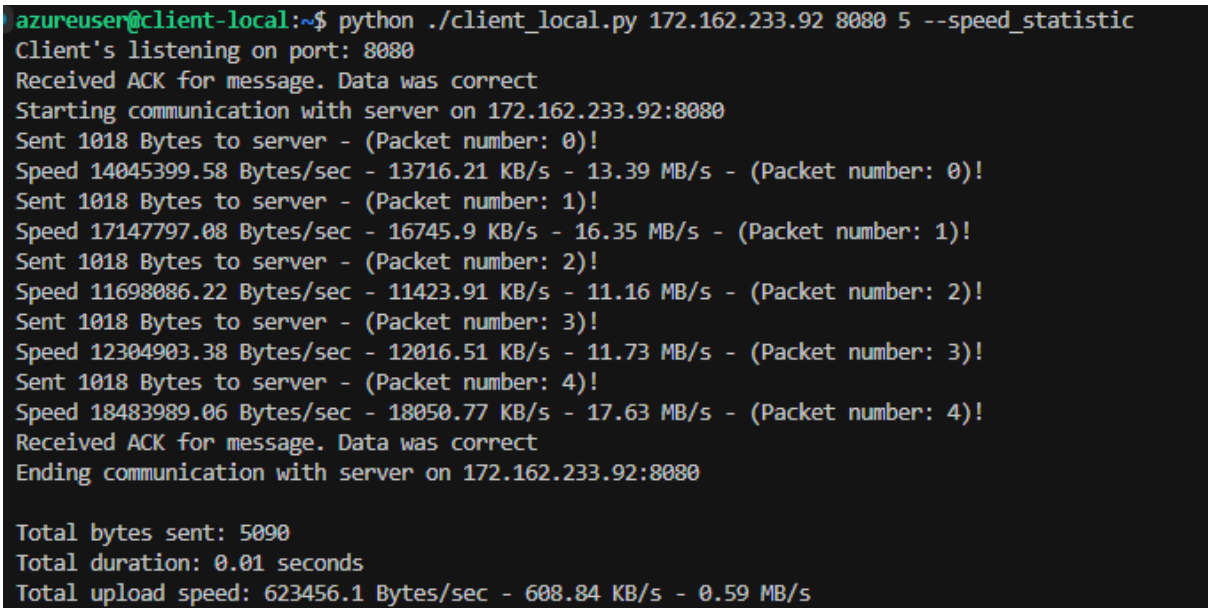
Statystyka prędkości upload i download

Statystyka upload i download została przetestowana zarówno w sieci lokalnej jak i sieci rozproszonej.

Połączenie w sieci lokalnej

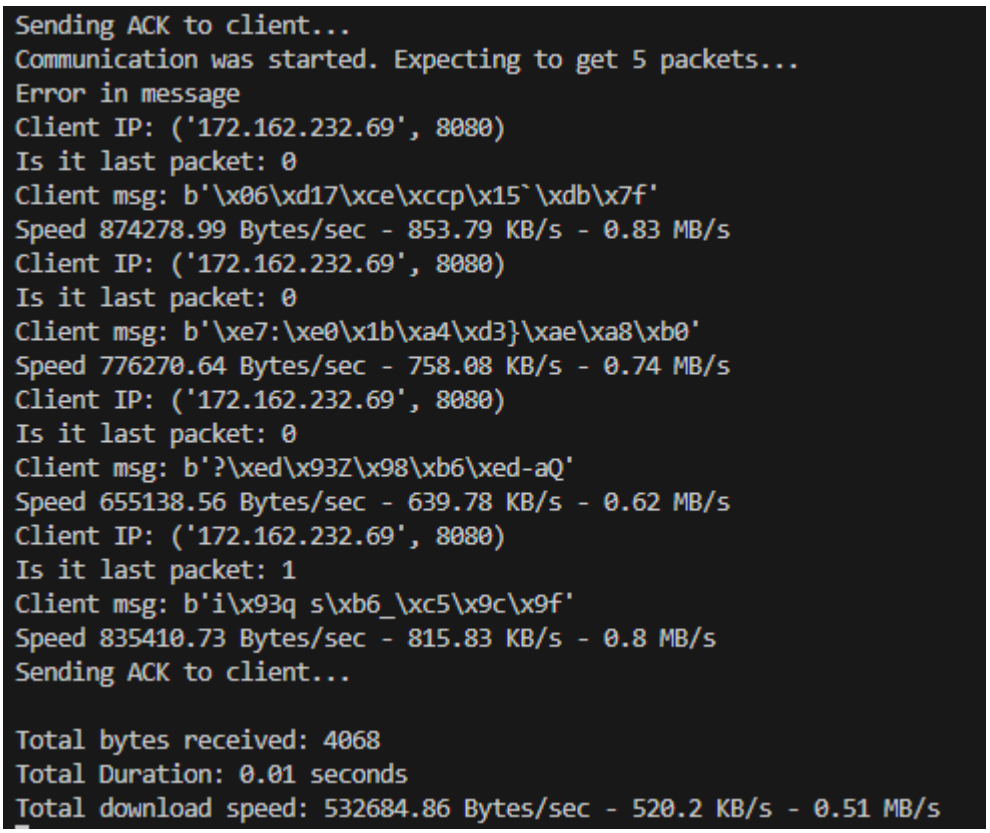
Wysyłanie danych z klienta lokalnego

Po wielokrotnym wykonaniu testów wysyłania dla klienta lokalnego zauważyliśmy, że prędkość wysyłania całości danych najczęściej wynosi około 600 Kb/s. Pojedyncze paczki z 1024 bajtami są wysyłane z prędkością około 13 000 - 18 000 Kb/s.



Odebranie danych z klienta lokalnego na serwerze

Odbieranie całości danych na serwerze wysłanych z klienta lokalnego najczęściej następuje z prędkością około 500 Kb/s. Pojedyncze paczki z 1024 bajtami są odbierane z prędkością około 600 - 800 Kb/s.



Połączenie w sieci rozproszonej

Wysyłanie danych z klienta zewnętrznego

Po wielokrotnym wykonaniu testów wysyłania dla klienta zewnętrznego zauważyliśmy, że prędkość wysyłania całości danych najczęściej wynosi około 25 - 50 Kb/s. Pojedyncze paczki z 1024 bajtami są wysyłane z prędkością około 7000 - 8000 Kb/s. Zatem widać że przesyłanie danych z klienta zewnętrznego zajmuje dużo dłużej niż z klienta lokalnego.

```
azureuser@client-external:~$ python ./client_external.py 172.162.233.92 8080 5 --speed_statistic
Client's listening on port: 8080
Received ACK for message. Data was correct
Starting communication with server on 172.162.233.92:8080
Sent 1018 Bytes to server - (Packet number: 0)!
Speed 7387199.78 Bytes/sec - 7214.06 KB/s - 7.04 MB/s - (Packet number: 0)!
Sent 1018 Bytes to server - (Packet number: 1)!
Speed 8242859.98 Bytes/sec - 8049.67 KB/s - 7.86 MB/s - (Packet number: 1)!
Sent 1018 Bytes to server - (Packet number: 2)!
Speed 10142046.25 Bytes/sec - 9904.34 KB/s - 9.67 MB/s - (Packet number: 2)!
Sent 1018 Bytes to server - (Packet number: 3)!
Speed 5996912.18 Bytes/sec - 5856.36 KB/s - 5.72 MB/s - (Packet number: 3)!
Sent 1018 Bytes to server - (Packet number: 4)!
Speed 7805852.78 Bytes/sec - 7622.9 KB/s - 7.44 MB/s - (Packet number: 4)!
Received ACK for message. Data was correct
Ending communication with server on 172.162.233.92:8080

Total bytes sent: 5090
Total duration: 0.2 seconds
Total upload speed: 25984.83 Bytes/sec - 25.38 KB/s - 0.02 MB/s
```

Odebranie danych z klienta zewnętrznego na serwerze

Odbieranie całości danych na serwerze wysłanych z klienta lokalnego najczęściej następuje z prędkością około 30 - 40 Kb/s. Pojedyncze paczki z 1024 bajtami są odbierane z prędkością około 400 - 700 Kb/s.

```
Sending ACK to client...
Communication was started. Expecting to get 5 packets...
Client IP: ('172.174.177.77', 8080)
Is it last packet: 0
Client msg: b'\xf5L\xe5\xcd2\xf3\x9b\xb6\xf4\xd7'
Speed 10702.71 Bytes/sec - 10.45 KB/s - 0.01 MB/s
Client IP: ('172.174.177.77', 8080)
Is it last packet: 0
Client msg: b'\xde\x1c1k\xec&nV\x94\x0b'
Speed 730412.19 Bytes/sec - 713.29 KB/s - 0.7 MB/s
Client IP: ('172.174.177.77', 8080)
Is it last packet: 0
Client msg: b'\xf0K\t\x97\xc7\xa7d\xad\x1e\xf4'
Speed 694272.0 Bytes/sec - 678.0 KB/s - 0.66 MB/s
Error in message
Client IP: ('172.174.177.77', 8080)
Is it last packet: 1
Client msg: b'\x9d\xfc\x17\x18i\x1b0\xa2\x15\xe4'
Speed 436468.55 Bytes/sec - 426.24 KB/s - 0.42 MB/s
Sending ACK to client...

Total bytes received: 4068
Total Duration: 0.1 seconds
Total download speed: 39729.68 Bytes/sec - 38.8 KB/s - 0.04 MB/s
```

Statystyka opóźnienia pakietów

```
python client_local.py 127.0.0.1 8081 10 --delay
Client's listening on port: 8081
Received ACK for message. Data was correct
RTT for packet: 0.0 seconds
Starting communication with server on 127.0.0.1:8081
Sent 1016 Bytes to server - (Packet number: 0)!
Received ACK for message. Data was correct
RTT for packet: 0.0 seconds
Starting communication with server on 127.0.0.1:8081
Sent 1016 Bytes to server - (Packet number: 1)!
Received ACK for message. Data was correct
RTT for packet: 0.0 seconds
Starting communication with server on 127.0.0.1:8081
Sent 1016 Bytes to server - (Packet number: 2)!
Received ACK for message. Data was correct
RTT for packet: 0.0 seconds
Starting communication with server on 127.0.0.1:8081
Sent 1016 Bytes to server - (Packet number: 3)!
Received ACK for message. Data was correct
RTT for packet: 0.0 seconds
```

Statystyka gubienia pakietów

Przetesujemy gubienie pakietów w sieci lokalneji sieci zewnętrznej. Zarówno dla klienta lokalnego jak i zewnętrznego zostaną przeprowadzone testy dla wysłanej ilości pakietów 10000 i 50000.

Odbieranie pakietów klienta lokalnego

- Sprawdzenie ilości zgubionych pakietów klienta lokalnego, który przesłał 10000 pakietów:

Próba nr 1:

```
Is it last packet: 0
Client msg: b'\x0b1\x8by\xc4\xa0\xf8\x12,\x8a'
Client IP: ('10.0.0.5', 8080)
Is it last packet: 0
Client msg: b'\xce\xb5h\x8d\xcb\xae\xd5\xa1\xf9u'
Client IP: ('10.0.0.5', 8080)
Is it last packet: 1
Client msg: b'\xcb\x80\x06\xa8\xe6\x9b\x16\x0bz\x85'
Sending ACK to client...
Received packets: 8438, expected: 10000, lost packets: 1562
```

Zgubionych zostało prawie 16% pakietów.

Próba nr 2:

```
Is it last packet: 0
Client msg: b'[\x92.\xbbh\x17\xfe\x9fy\x11'
Client IP: ('10.0.0.5', 8080)
Is it last packet: 0
Client msg: b'\x15tlX\x1e\x9b\xc2\x07\xcc\x8e'
Client IP: ('10.0.0.5', 8080)
Is it last packet: 1
Client msg: b'\x9dm\xd4\xb4\xd1\x17\xe5\xc8\x82\xbd'
Sending ACK to client...
Received packets: 7205, expected: 10000, lost packets: 2795
```

Zgubionych zostało blisko 28% pakietów.

Próba nr 3:

```
Is it last packet: 0
Client msg: b'~]j\xfa\xfc\n\x90E|\x17'
Client IP: ('10.0.0.5', 8080)
Is it last packet: 1
Client msg: b'\x98h\xd4\x8eR!-LOe'
Sending ACK to client...
Received packets: 9021, expected: 10000, lost packets: 979
```

Zgubionych zostało prawie 10% pakietów.

Łącznie zostało zgubionych prawie 18% wszystkich pakietów.

- Sprawdzenie ilości zgubionych pakietów klienta lokalnego który przesłał 50000 pakietów:

Próba nr 1:

```
Is it last packet: 0
Client msg: b'\xb2!<\xb2Vq\xde\x13c|'
Client IP: ('10.0.0.5', 8080)
Is it last packet: 0
Client msg: b'\tSR+3\x0c\xb6\xac{\xe5'
Client IP: ('10.0.0.5', 8080)
Is it last packet: 1
Client msg: b"\x1f\xa0\xcbt\xad'A\xc5\x91\xe5"
Sending ACK to client...
Received packets: 46107, expected: 50000, lost packets: 3893
```

Zgubionych zostało prawie 8% pakietów.

Próba nr 2:

```
Is it last packet: 0
Client msg: b'd\x11\xb5\xfe\x9f\t\x104\xe9\xda'
Client IP: ('10.0.0.5', 8080)
Is it last packet: 1
Client msg: b'\xb0\xa7\xf0\xbdW\x86\xc0\xd1\xb3\xf8'
Sending ACK to client...
Received packets: 38181, expected: 50000, lost packets: 11819
```

Zgubionych zostało prawie 24% pakietów.
Próba nr 3:

```
Is it last packet: 0
Client msg: b'y\xae\xafe\xd7\xc9\xfd\x1e\xd4\x13'
Client IP: ('10.0.0.5', 8080)
Is it last packet: 1
Client msg: b'z@\x08\x10ee\xef&&|'
Sending ACK to client...
Received packets: 49165, expected: 50000, lost packets: 835
```

Zgubionych zostało prawie 2% pakietów.

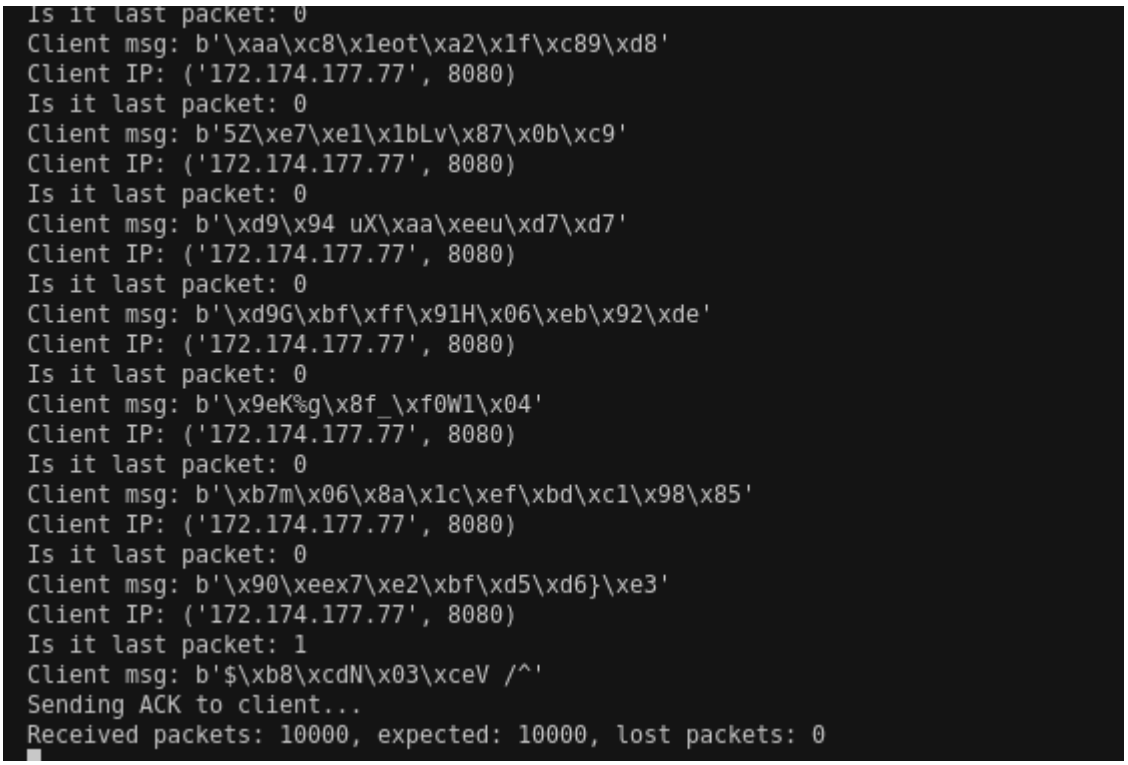
Widzimy, że przedział może być bardzo duży - w niektórych przypadkach zgubione pakiety stanowią jedynie kilka procent wszystkich pakietów - a w innych przypadkach ich liczba może wynosić blisko ¼ wszystkich pakietów.

Łącznie zostało zgubionych prawie 11% wszystkich pakietów.

Odbieranie pakietów klienta zewnętrznego

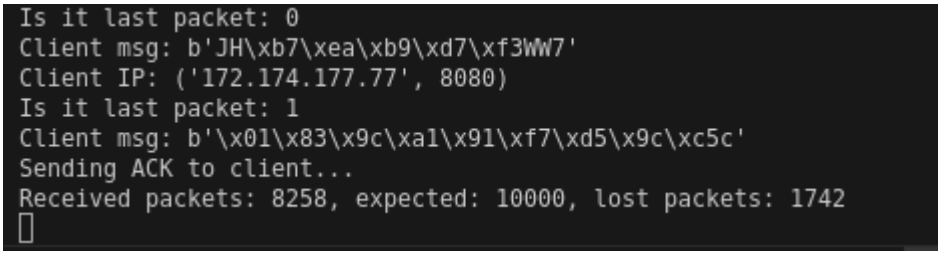
- Sprawdzenie ilości zgubionych pakietów klienta zewnętrznego, który przesłał 10000 pakietów:

Próba nr 1:



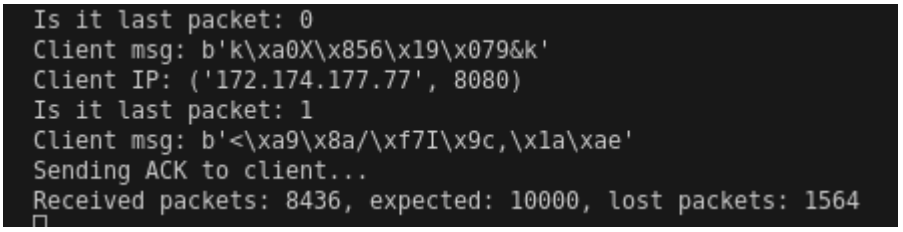
Wszystkie pakiety zostały odebrane przez serwer.

Próba nr 2:



Zgubionych zostało ponad 17% pakietów.

Próba nr 3:

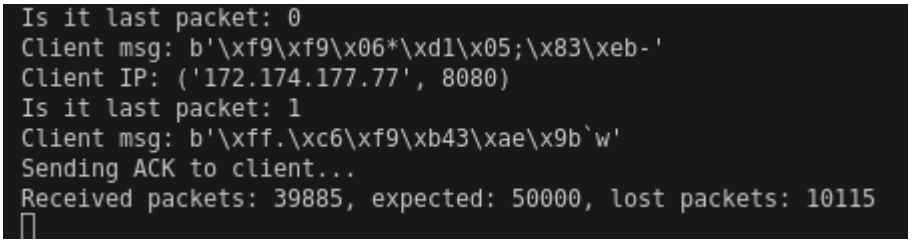


Zgubionych zostało ponad 15% pakietów.

Łącznie zostało zgubionych ponad 11% wszystkich pakietów.

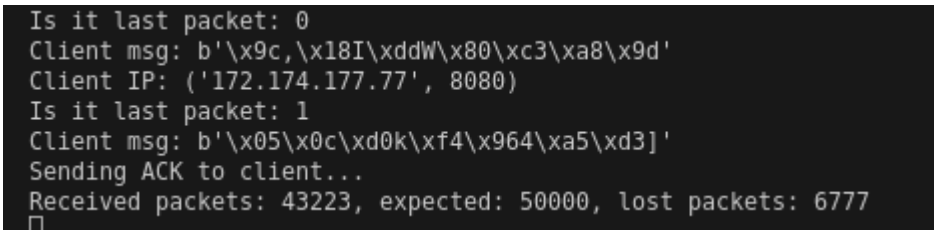
- Sprawdzenie ilości zgubionych pakietów klienta zewnętrznego, który przesłał 50000 pakietów:

Próba nr 1:



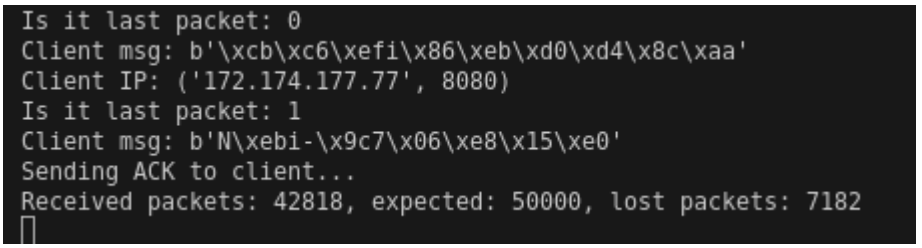
Zgubionych zostało ponad 20% pakietów.

Próba nr 2:



Zgubionych zostało 13,5 % pakietów.

Próba nr 3:



Zgubionych zostało ponad 14 % pakietów.

Łącznie zostało zgubionych 16% wszystkich pakietów.

Podsumowanie:

Przy przesyłaniu 10000 lepiej sobie poradził klient zewnętrzny (11% do 18%), natomiast przy przesyłaniu 50000 pakietów lepsze wyniki osiągnął klient lokalny (11% do 16%). Można więc wysunąć wniosek, że rodzaj klienta nie miał znaczenia w przypadku gubienia pakietów.