Programowanie Sieciowe

Andrii Rybachok Ihor Gavrylenko Anna Tamelo

12.11.2025

Treść zadania

Z 1

Komunikacja UDP Napisz zestaw dwóch programów – klienta i serwera wysyłające datagramy UDP. Proszę napisać jedno zadanie w konfiguracji klient/serwer Python/C, a drugie w konfiguracji klient/serwer C/Python – do wyboru.

Z 1.1

Klient wysyła a serwer odbiera datagramy oraz odsyła ustaloną odpowiedź. Klient powinien wysyłać kolejne datagramy o przyrastającej wielkości, tj. 2, 4, 8, 16, 32, itd. bajtów. Ustalić eksperymentalnie z dokładnością do jednego bajta jak duży datagram jest obsługiwany. Wyjaśnić. Zmierzyć czas pomiędzy wysłaniem wiadomości a odebraniem odpowiedzi po stronie klienta i zestawić wyniki na wykresie.

Opis rozwiązania problemu

Programy klienta i serwera zostały napisane z wykorzystaniem Network API w językach C oraz Python.
Zgodnie z protokołem IPv4, pole Total Length w nagłówku IP ma 16 bitów, co pozwala na przechowywanie wartości od 0 do 65 535.
Z tej wartości należy odjąć długość nagłówków:

- IP – 20 bajtów,
- UDP – 8 bajtów,

co daje maksymalny rozmiar danych w datagramie:

65 535 − 28 = 65 507 bajtów.

Aby to potwierdzić, klient wysyłał datagramy o rosnącym rozmiarze (kolejne potęgi dwójki) aż do osiągnięcia wartości maksymalnej.

Przykład kodu klienta w języku Python:

```
with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
    s.settimeout(10)
    for i in range(1, 17):
        data_size = math.pow(2, i) - 1
        if data_size > MAX_SIZE:
            data_size = MAX_SIZE
        send_data = b"x" * int(data_size)
        start_time = time.time()
```

```
        try:
            s.sendto(send_data, (HOST, SERVER_PORT))
            data = s.recvfrom(1024)
            end_time = time.time()
            if data[0] == ACK_MSG:
                rtt = (end_time - start_time) * 1000  # w ms
                print(f"{int(data_size)} bajtów OK, RTT = {rtt:.3f} ms")
                sizes.append(int(data_size))
                times.append(rtt)
            else:
                print("Niepoprawna odpowiedź")
                break
        except socket.timeout:
            print(f"Timeout przy rozmiarze {int(data_size)} bajtów -
przekroczony maksymalny rozmiar datagramu")
            break
```

Uwagi dotyczące problemów

Podczas testowania pojawił się problem z ustaleniem adresu IP serwera uruchomionego w kontenerze Docker. Zostało to rozwiązane poprzez użycie aliasu sieciowego kontenera (np. serverc, serverp), który następnie był przekazywany klientowi jako parametr.
Klient uzyskiwał adres IP aliasu przy pomocy zapytania DNS w ramach sieci Docker

Opis konfiguracji testowej

Serwer jest uruchomiony na

- Port - 12345
- Alias sieciowy - serverc/serverp
- IP - jest przydzielony przez docker, korzystamy z sieci z35_network

Podczas uruchomienia kontenera klienta podajemy w argumentach adres portu oraz alias sieciowy serwera.

Opis testowania i wydruki z testów

Konfiguracja 1 - server C, klient Python

Tworzymy kontener **klienta** za pomocą komend:

docker build -t z35_zad1_clientp
docker run -it --name z35_zad1_clientp_kont --network z35_network z35_zad1_clientp 12345 serverc

Komendy dla **serwera:**

docker build -t z35_zad1_serverc .
docker run -it --network-alias serverc --name z35_zad1_serverc_kont --network z35_network z35_zad1_serverc

Wyniki testowania

Log serwera:

1 bajtów OK, RTT = 1.096 ms
3 bajtów OK, RTT = 0.314 ms
7 bajtów OK, RTT = 0.347 ms
…
32767 bajtów OK, RTT = 0.265 ms
65507 bajtów OK, RTT = 0.376 ms

Log klienta:

Odebrano datagram o długości 1 bajtów
Odebrano datagram o długości 3 bajtów
Odebrano datagram o długości 7 bajtów
…
Odebrano datagram o długości 32767 bajtów
Odebrano datagram o długości 65507 bajtów

Konfiguracja 2 - serwer Python, klient C

Komendy do tworzenia **klienta**:
docker build -t z35_zad1_clientc .
docker run -it --name z35_zad1_clientc_kont --network z35_network z35_zad1_clientc serverp 12345

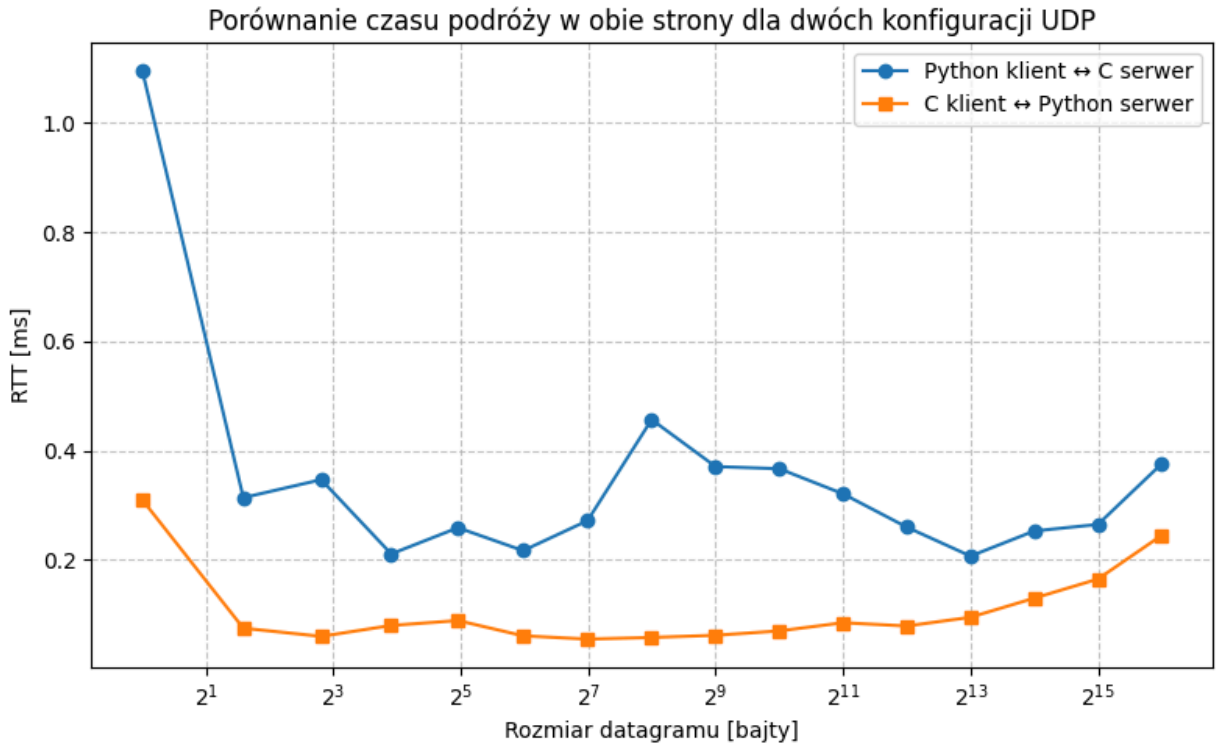
Komendy dla **serwera:**
docker run -it --network-alias serverp --name z35_zad1_serverp_kont --network z35_network z35_zad1_serverp
docker build -t z35_zad1_serverp .

Wyniki testowania:

Logowanie z klienta:
Testowanie rozmiarów datagramów...
1 bajtów OK, RTT = 0.357 ms
3 bajtów OK, RTT = 0.078 ms
…
65507 bajtów OK, RTT = 0.226 ms

Logowanie z serwera:
Serwer UDP nasłuchuje na 0.0.0.0:12345...
Odebrano 1 bajtów od ('172.21.35.3', 43020)
Odebrano 3 bajtów od ('172.21.35.3', 43020)
…
Odebrano 65507 bajtów od ('172.21.35.3', 43020)

Wykres porównujący dwie konfiguracji



Czas, za który wszystkie dane zostały wysłane
Konfiguracja 1 - 5.593s
Konfiguracja 2 - 1.719s

Wnioski końcowe
W obu konfiguracjach udało się potwierdzić maksymalny rozmiar datagramu UDP równy 65 507 bajtów, a konfiguracja klient C / serwer Python okazała się około czterokrotnie szybsza od odwrotnej; zmierzony czas RTT mieścił się w zakresie od około 0,2 do 1 ms, co potwierdza poprawne działanie komunikacji lokalnej UDP, przy czym w praktyce zweryfikowano również

eksperymentalnie limit rozmiaru datagramu wynikający z budowy nagłówków IP i UDP, a napotkane problemy z adresacją w środowisku Docker zostały skutecznie rozwiązane dzięki zastosowaniu aliasów sieciowych i mechanizmu DNS w sieci kontenerowej.