Data: 15.11.2025

PSI – Sprawozdanie zad 1.1

Autorzy:
Weronika Maślana
Alesia Filinkova
Diana Pelin

Z 1 Komunikacja UDP
Napisz zestaw dwóch programów – klienta i serwera wysyłające datagramy UDP. Proszę napisać jedno zadanie w konfiguracji klient/server Python/C, a drugie w konfiguracji klient/server 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 eskperymentalnie 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.

1. Opis rozwiązania problemu

Celem zadania było przygotowanie zestawu programów klient–serwer komunikujących się za pomocą protokołu UDP. Klient wysyłał kolejne datagramy o rosnącym rozmiarze (2, 4, 8, …, aż do maksymalnego rozmiaru możliwego do wysłania) i mierzył czas odpowiedzi od serwera. Serwer odbierał datagramy i odsyłał stałą odpowiedź `"ACK"`.

W ramach realizacji zadania utworzono:

- **Serwer UDP w języku C** – obsługujący dowolny rozmiar datagramu do 65535 bajtów, działający w pętli nieskończonej, odbierający datagramy i odsyłający potwierdzenie do nadawcy.
- **Klient UDP w Pythonie** – wysyłający kolejne datagramy o rosnącym rozmiarze, mierzący czas wysłania i odebrania odpowiedzi oraz zapisujący wyniki w formie wykresu.

Fragment kodu serwera, odpowiadający na przychodzące datagramy:

```c
while (1) {
    client_len = sizeof(client_addr);
    n = recvfrom(fd: sock, buf: buffer, n: BUFFER_SIZE, flags: 0, addr: (struct sockaddr *)&client_addr, addr_len: &client_len);
    if (n < 0) {
        perror(s: "recvfrom failed");
        continue;
    }

    printf(format: "Received %d bytes\n", n);

    // Client answer
    const char *ack = "ACK";
    if (sendto(fd: sock, buf: ack, n: strlen(s: ack), flags: 0, addr: (struct sockaddr*)&client_addr, addr_len: client_len) < 0) {
        bailout(s: "sendto failed");
    }
}
```

Fragment kodu klienta odpowiadający za wysyłanie datagramów i pomiar czasu:

```
with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as sock:
    sock.settimeout(1)
    for size in sizes:
        data = b"a" * size
        print(f"Sending buffer size = {size} bytes")
        start = time.time()
        try:
            try:
                sock.sendto(data, (host, port))
            except OSError as e:
                print(f"Cannot send {size} bytes: {e}")
                break
            resp, _ = sock.recvfrom(1024)
            end = time.time()
            elapsed = (end - start) * 1000
            times.append(elapsed)
            print(f"Received {len(resp)} bytes, elapsed time = {elapsed:.2f} ms")
        except socket.timeout:
            print(f"Timeout for {size} bytes")
            times.append(None)
```

Do generowania wykresu wykorzystano bibliotekę `matplotlib` z backendem `Agg`, umożliwiającym zapis do pliku w środowisku bez interfejsu graficznego, np. w kontenerze Docker.

2. Uwagi dot. problemów napotkanych i ich rozwiązania

Podczas realizacji zadania napotkano kilka istotnych problemów:

1. **Błąd maksymalnego rozmiaru datagramu UDP** – próba wysłania datagramu o rozmiarze **65508** bajtów powodowała wyjątek `OSError: Message too long`. Rozwiązano to przez obsługę wyjątku i przerwanie pętli w przypadku przekroczenia limitu.
2. **Błędy w Dockerze związane z wyświetlaniem wykresu** – w środowisku Docker brak interfejsu GUI uniemożliwia wyświetlenie wykresu przy użyciu `plt.show()`. Rozwiązano to poprzez ustawienie backendu `Agg` oraz zapis wykresu do pliku w katalogu współdzielonym z hostem, `/output/zad1_1.png`. Oraz komendy `docker run -it -v $(pwd):/output --network z31_network z31_pclient1 z31_pserver1 8001.`

   gdzie: `-v $(pwd):/output` pozwala zachować wykres utworzony w środowisku Doker na hoscie.

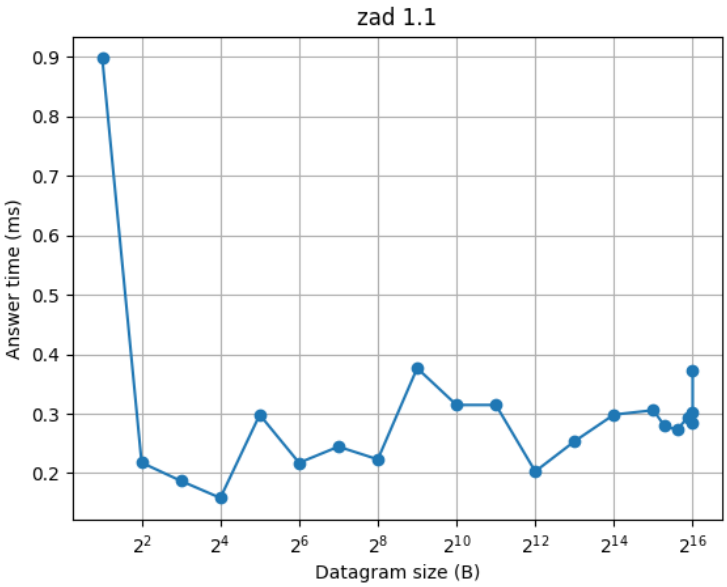
3. Opis konfiguracji testowej

- **Serwer UDP:** kontener Docker `z31_pserver1`, port 8001, sieć Docker `z31_network`, host i alias kontenera ustawione na `z31_pserver1`.
- **Klient UDP:** kontener Docker `z31_pclient1`, połączony do tej samej sieci `z31_network`, komunikacja z serwerem pod adresem `z31_pserver1:8001`.
- **Środowisko uruchomieniowe:** Docker z obrazami Python 3 dla klienta i GCC 4.9 dla serwera. Backend matplotlib `Agg` dla klienta do zapisu wykresu w środowisku headless.

4. Opis testowania i wyniki

Przeprowadzono serię testów, w których klient wysyłał datagramy o rosnącej wielkości i mierzył czas odpowiedzi serwera.

Wynikowy wykres:

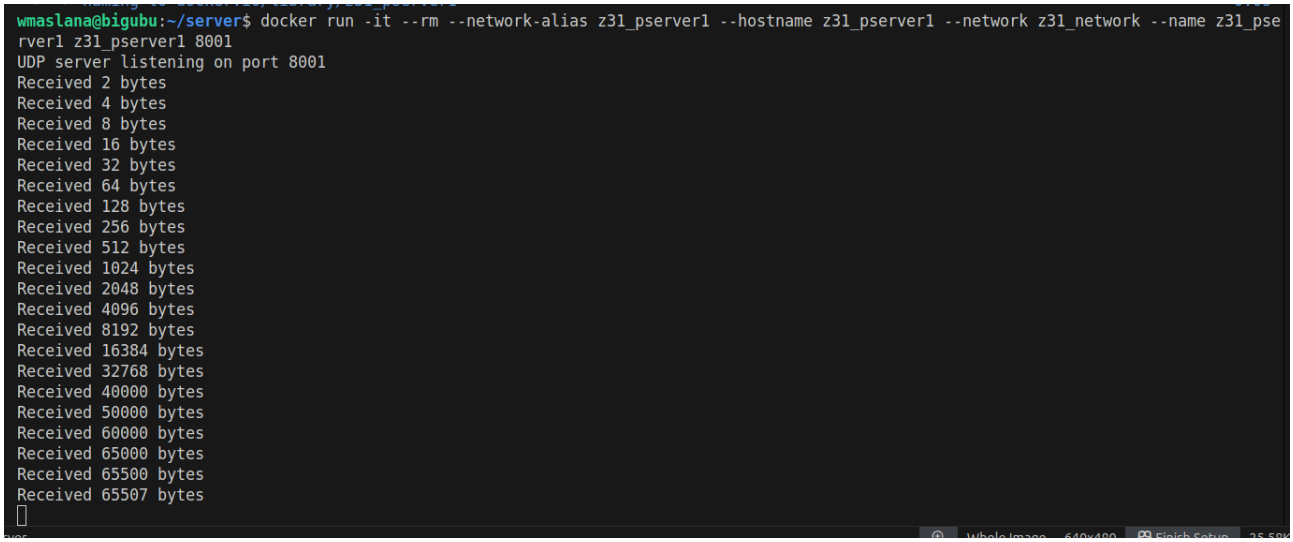


Datagramy powyżej **65507** bajtów były odrzucane przez system i generowały wyjątek, dlatego test zakończono na tym rozmiarze. Wszystkie poprawnie wysłane i odebrane datagramy zostały zwizualizowane na wykresie zapisanym do pliku `zad1_1.png` i przedstawionym powyżej.

Błąd z klienta i przesłane datagramy:

```
wmaslana@bigubu:~/client$ docker run -it --rm -v $(pwd):/output --network z31_network z31_pclient1 z31_pserver1 8001
Will send to z31_pserver1:8001
Sending buffer size = 2 bytes
Received 3 bytes, elapsed time = 1.23 ms
Sending buffer size = 4 bytes
Received 3 bytes, elapsed time = 0.45 ms
Sending buffer size = 8 bytes
Received 3 bytes, elapsed time = 0.43 ms
Sending buffer size = 16 bytes
Received 3 bytes, elapsed time = 0.32 ms
Sending buffer size = 32 bytes
Received 3 bytes, elapsed time = 0.26 ms
Sending buffer size = 64 bytes
Received 3 bytes, elapsed time = 0.32 ms
Sending buffer size = 128 bytes
Received 3 bytes, elapsed time = 0.23 ms
Sending buffer size = 256 bytes
Received 3 bytes, elapsed time = 0.37 ms
Sending buffer size = 512 bytes
Received 3 bytes, elapsed time = 0.23 ms
Sending buffer size = 1024 bytes
Received 3 bytes, elapsed time = 0.26 ms
Sending buffer size = 2048 bytes
Received 3 bytes, elapsed time = 0.30 ms
Sending buffer size = 4096 bytes
Received 3 bytes, elapsed time = 0.22 ms
Sending buffer size = 8192 bytes
Received 3 bytes, elapsed time = 0.22 ms
Sending buffer size = 16384 bytes
Received 3 bytes, elapsed time = 0.36 ms
Sending buffer size = 32768 bytes
Received 3 bytes, elapsed time = 0.34 ms
Sending buffer size = 40000 bytes
Received 3 bytes, elapsed time = 0.37 ms
Sending buffer size = 50000 bytes
Received 3 bytes, elapsed time = 0.40 ms
Sending buffer size = 60000 bytes
Received 3 bytes, elapsed time = 0.41 ms
Sending buffer size = 65000 bytes
Received 3 bytes, elapsed time = 0.44 ms
Sending buffer size = 65500 bytes
Received 3 bytes, elapsed time = 0.36 ms
Sending buffer size = 65507 bytes
Received 3 bytes, elapsed time = 0.36 ms
Sending buffer size = 65508 bytes
Cannot send 65508 bytes: [Errno 90] Message too long
Client finished.
wmaslana@bigubu:~/client$
```

Odpowiedzi serwera:



5. Wnioski końcowe

1. `65535` to maksymalny rozmiar pakietu IP, więc

   dane = 65535 − 20(maksymalny rozmiar pakietu IP) − 8(bajtów nagłówka UDP) = **65507 B**.

   W eksperymencie w Dockerze największy możliwy do wysłania datagram miał rozmiar **65507 B**.

2. Podczas testów zauważono, że czas przesyłania pierwszego datagramu jest relatywnie wysoki (rzędu ok. 0.9 ms), co wynika z kosztu wywołań systemowych i pomiaru w Pythonie, a nie z samej transmisji sieciowej.

   Dla średnich rozmiarów czas jest stabilny i wynosi około 0.3 ms.

3. W pracy z Dockerem konieczne jest uwzględnienie środowiska headless – wykresy należy zapisywać do pliku zamiast używać `plt.show()`.

4. Zadanie pozwoliło zweryfikować ograniczenia praktyczne protokołu UDP i efekty pracy w środowisku wirtualnym