Programowanie sieciowe

Miłosz Cieśla, Filip Ryniewicz, Aleksander Szymczyk

Sprawozdanie z zadania 1.1

16.11.2024

1 Treść zadania

Z 1 - Komunikacja UDP

Napisz zestaw dwóch programów – klienta i serwera wysyłających datagramy UDP. Wykonaj ćwiczenie w kolejnych inkrementalnych wariantach (rozszerzając kod z poprzedniej wersji). Zarówno klient, jak i serwer powinien być napisany w językach C oraz Python (4 programy). Sprawdź i przetestuj działanie międzyplatformowe, tj. klient w C z serwerem w Pythonie i odwrotnie.

Z 1.1

Klient wysyła, a serwer odbiera datagramy o stałym rozmiarze (rzędu kilkuset bajtów). Datagramy powinny posiadać ustaloną formę danych. Przykładowo:

- Pierwsze dwa bajty datagramu mogą zawierać informację o jego długości.
- Kolejne bajty mogą zawierać kolejne litery A-Z powtarzające się wymaganą liczbę razy (inne rozwiązania również są dopuszczalne).

Odbiorca powinien weryfikować odebrany datagram i odsyłać odpowiedź o ustalonym formacie. Klient powinien wysyłać kolejne datagramy o przyrastającej wielkości, np.:

$$1, 100, 200, 1000, 2000, \ldots \text{ bajtów.}$$

Sprawdź, jaki był maksymalny rozmiar wysłanego (przyjętego) datagramu. Ustal z dokładnością do jednego bajta, jak duży datagram jest obsługiwany. Wyjaśnij.

2 Opis rozwiązania

Ustalony format datagramu:

- pierwszy i drugi bajt: długość przesyłanego datagramu
- trzeci i czwarty bajt: suma kontrolna obliczana algorytmem fletcher16
- kolejne bajty: zawierają kolejne litery A-Z powtarzające się wymaganą liczbę razy

Ustalony format odpowiedzi:

- pierwszy i drugi bajt: długość przesyłanego datagramu
- trzeci i czwarty bajt: suma kontrolna którą serwer odczytał dla wcześniej przesłanej wiadomości

W celu określenia maksymalnego rozmiaru datagramu, przeprowadziliśmy eksperyment polegający na przesyłaniu datagramów o rosnących wielkościach.

2.1 Schemat działania

- Serwer i klient tworzą własne sockety UDP.
- Serwer podpina swój socket pod wybrany przez argument wywołania port.
- Serwer oczekuje na nadejście datagramu.
- Klient generuje wiadomość.
- Na podstawie wiadomości klient generuje sumę kontrolną.
- Tworzymy datagram w założonym formacie 2.
- Klient przesyła datagram.
- Serwer odbiera datagram.
- Serwer oblicza dla wiadomości z datagramu sumę kontrolną i porównuje ją z otrzymaną od serwera. Korzystając z tego może on potwierdzić poprawność danych.

- Serwer tworzy datagram w założonym formacie odpowiedzi 2
- Serwer wysyła datagram do adresu otrzymanego wcześniej dzięki funkcji recvfrom.
- Klient odbiera datagram z odpowiedzią.
- Klient sprawdza czy suma kontrolna z odpowiedzi jest zgodna z tą którą wysyłał, dzięki temu potwierdza on udaną komunikację.

Struktura projektu została opisana w README.md.
Kod zawiera komentarze opisujące jego działanie.

3 Problemy

- Problem z zapisem danych w poprawnym formacie wewnątrz struktury danych BytesIO. Rozwiązaliśmy go używając wbudowanej funkcji pythonowej to_bytes, było to dla nas mylące ze względu na wbudowany mechanizm pythona wyświetlający wartości bajtów jako ASCII.
- Przy przesyłaniu z klienta na serwer nie uwzględniliśmy odpowiednio rozmiaru metadanych, przez co wiadomość była odczytywana dopiero przy rozmiarze datagramu większego niż 4 bajty. Naprawiliśmy to uwzględniając długość headera w długości datagramu.
- Mieliśmy problem z logowaniem wiadomości z print() w kontenerze, naprawiliśmy to podając poprawne flagi -dit.

4 Konfiguracja testowa

Rozwiązanie było testowane w dwóch wariantach:

4.1 Lokalnie

Utworzono sieć docker typu bridge o nazwie `z40_network`. Serwer przyjmował adres IP: `172.19.0.2`, a klient: `172.19.0.3`. Serwer UDP uruchamiany był na porcie `8000`, który to port był następnie podawany do klienta wraz z nazwą sieci.

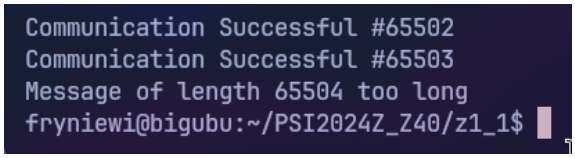
4.2 Na serwerze `bigubu.ii.pw.edu.pl`

Do testów wykorzystywano utworzoną wcześniej sieć o nazwie `z40_network`. Serwer przyjmował adres IP: `172.21.40.2`, a klient: `172.21.40.3`. Serwer UDP uruchamiany był na porcie `8000`, który to port był następnie podawany do klienta wraz z nazwą sieci.

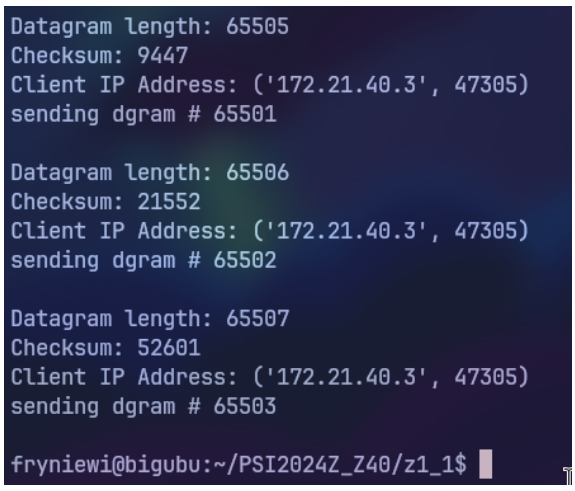
5 Testy

Testy polegały na nawiązaniu komunikacji klienta z serwerem i przesyłaniu datagramów o rosnących rozmiarach. Przetestowane zostało działanie międzyplatformowe, wyniki eksperymentów były identyczne we wszystkich konfiguracjach.

5.1 Python → Python

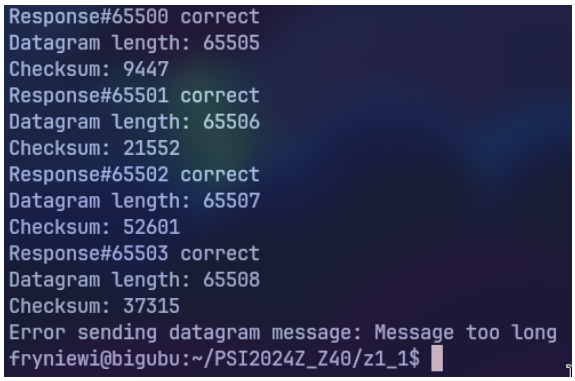


Rysunek 1: Błąd przy wysyłaniu za dużego datagramu (`65504 + 4`) (do długości samej wiadomości `65504` musimy dodać `4` aby otrzymać długość całego datagramu) 2



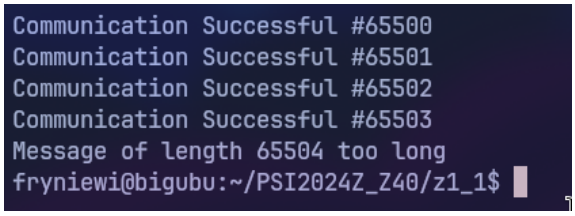
Rysunek 2: Ostatni datagram prawidłowo odebrany przez serwer

5.2 C → C



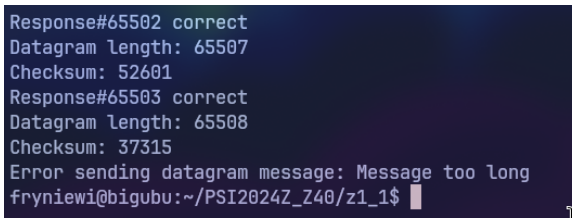
Rysunek 3: Błąd przy wysyłaniu za dużego datagramu (`65504 + 4`)

5.3 Python → C



Rysunek 4: Błąd przy wysyłaniu za dużego datagramu (`65504 + 4`)

5.4 C → Python



Rysunek 5: Błąd przy wysyłaniu za dużego datagramu (`65504 + 4`)

6 Wnioski

Maksymalny rozmiar wysłanego i przyjętego datagramu w protokole UDP to `65507` bajtów. Jest to spowodowane tym, że protokół IPv4 definiuje pole "Total Length"w nagłówku IP jako 16-bitowe, co oznacza, że maksymalny rozmiar całego pakietu IP (łącznie z nagłówkiem) wynosi $2^{16} - 1 = 65\,535$ bajtów. Standardowy nagłówek IP ma rozmiar 20 bajtów, z kolei nagłówek UDP posiada stały rozmiar 8 bajtów.

- **Obliczenie maksymalnego rozmiaru danych w datagramie UDP**
  - Całkowity maksymalny rozmiar pakietu IP: 65 535 bajtów.
  - Odjęcie rozmiaru nagłówka IP: $65\,535 - 20 = 65\,515$ bajtów (maksymalny rozmiar datagramu UDP łącznie z nagłówkiem UDP).
  - Odjęcie rozmiaru nagłówka UDP: $65\,515 - 8 = 65\,507$ bajtów (maksymalny rozmiar danych użytkowych w datagramie UDP).