

# Programowanie sieciowe

## zadanie 1.2

12.11.2024

Bogumił Stoma

Aleksander Stanoch

Sebastian Abramowski

## Treść zadania

### Z 1.1

Wychodzimy z kodu z zadania 1.1, tym razem pakiety datagramu mają stałą wielkość, można przyjąć np. 512B. Należy zaimplementować prosty protokół niezawodnej transmisji, uwzględniający możliwość gubienia datagramów. Rozszerzyć protokół i program tak, aby gubione pakiety były wykrywane i retransmitowane. Wskazówka – **Bit alternate protocol**. Należy uruchomić program w środowisku symulującym błędy gubienia pakietów. (Informacja o tym, jak to zrobić znajduje się w skrypcie opisującym środowisko Dockera).

To zadanie można wykonać, korzystając z kodu klienta i serwera napisanych w C lub w Pythonie (do wyboru). Nie trzeba tworzyć wersji w obydwu językach.

## Rozwiązanie zadania

Zadanie polegało na poprawieniu kodu z poprzedniego zadania. Natomiast tym razem, nie musieliśmy sprawdzać gdzie leży limit wielkości wysyłanego datagramu, a mieliśmy zapewnić by po zgubieniu wysłanego pakietu, wysłać go ponownie. Do tego celu skorzystaliśmy z poleconego w zadaniu **Bit alternate protocol**. Przy wysyłaniu z klienta pakietu wystarczyło dodać jeden bajt

sekwencyjny, po czym sprawdzić **ACK, czyli numer sekwencyjny otrzymany od klienta w serwerze**, a w serwerze trzeba było otrzymać pakiet, i odesłać odpowiedni ACK. W wypadku gdy otrzymany ACK nie zgadza się z oczekiwanym, retransmitujemy pakiet.

Mechanizm działania:

1. Każdy pakiet zawiera numer sekwencyjny (0 lub 1). Numer ten zmienia się po pomyślnym odbiorze pakietu.
2. Serwer wysyła potwierdzenie zawierające numer sekwencji pakietu.
3. Retransmisja w przypadku błędów:
  - Zakleszczenie:
    - Klient wysyła sendto do serwera, ale pakiet jest gubiony w sieci
    - Serwer blokuje się w recvfrom i nic nie dostaje
    - A klient też czeka w recvfrom, bo server mu powinien odesłać 1 lub 0 a server jest zablokowany
    - Dlatego skorzystaliśmy z socket.settimeout aby wyjść z zakleszczenia
  - Jeśli klient nie otrzyma ACK lub otrzyma nieprawidłowe, retransmituje ten sam pakiet.
  - Jeśli serwer otrzyma pakiet z numerem sekwencyjnym, który już wcześniej został przetworzony, odpowiada tym samym potwierdzeniem (ACK), które wysłał wcześniej, aby poinformować klienta, że pakiet został już odebrany

## Implementacja w Pythonie

Wprowadzone zmiany były stosunkowo proste, jednak wymagały dodania mechanizmów zarządzania numerami sekwencji oraz dodania socket.settimemout, aby program się nie zakleszczał.

### Serwer:

- Nasłuchuje na porcie UDP.
- Oczekuje na pakiety od klienta.

- Sprawdza numer sekwencyjny pakietu i ewentualnie zmienia numer sekwencyjny oczekiwanego pakietu.
- Wysyła **ACK** z numerem sekwencji.
- Obsługuje **timeout** i zgłasza błędy, jeśli pakiety są zgubione lub opóźnione.

## Klient:

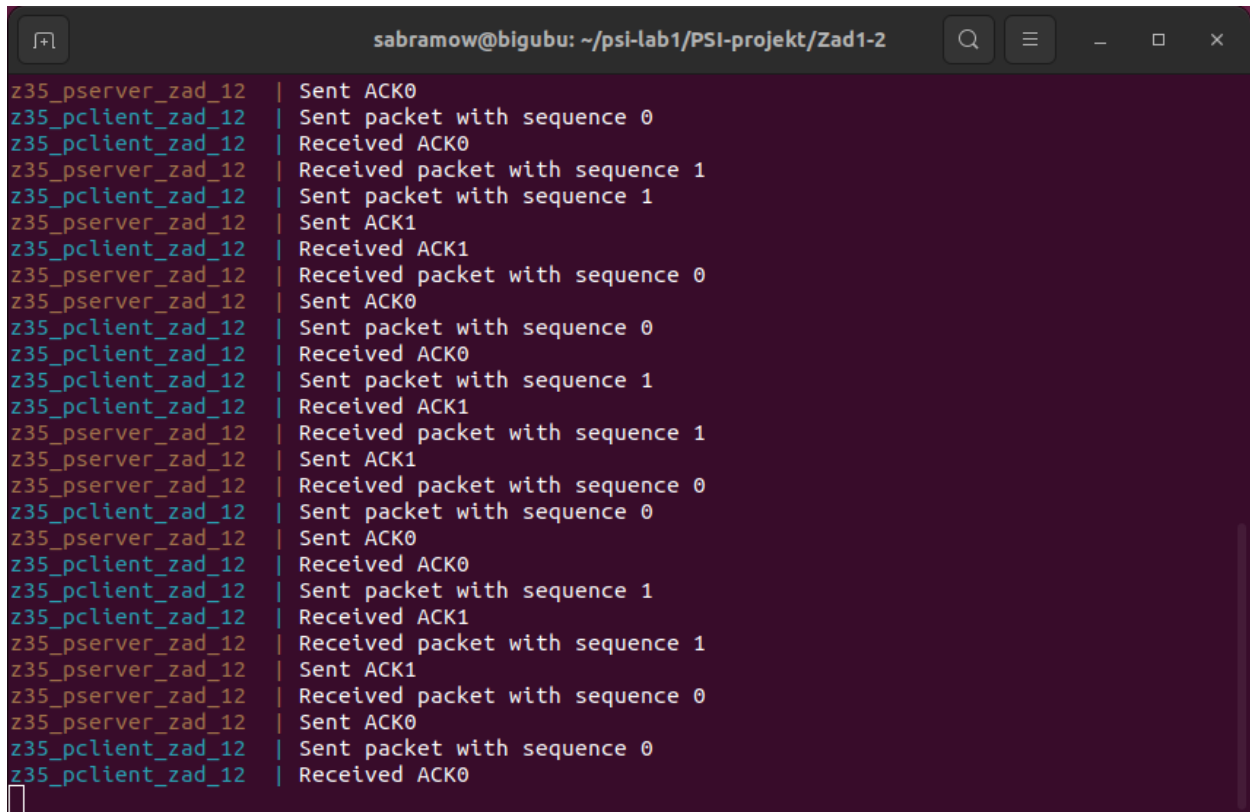
- Wysyła pakiety UDP do serwera.
- Każdy pakiet ma numer sekwencyjny (zmienia się naprzemiennie).
- Oczekuje na **ACK** od serwera.
- Jeśli **ACK** jest poprawne, zmienia numer sekwencyjny i wysyła kolejny pakiet.
- Jeśli **ACK** jest niepoprawne lub wystąpi **timeout**, retransmituje pakiet.
- Ogranicza liczbę prób (**retry limit**) i kończy działanie po osiągnięciu limitu.

## Opis konfiguracji testowej

- **Adres IP serwera:** 172.21.35.2 (może być przekazany jako argument w linii poleceń).
- **Port:** 12345 (domyślnie, ale można zmienić, podając go jako argument w linii poleceń).
- **Docker:** Środowisko testowe zostało uruchomione przy użyciu Docker Compose, co umożliwiło uruchomienie dwóch kontenerów w jednej sieci. Zasymulowano gubienie pakietów wysyłanych przez klienta, używając komendy tc w kontenerze klienta do wprowadzenia odpowiedniej straty pakietów w sieci.

## Screeny

Wysyłanie i odbieranie pakietów bez gubienia



```
sabramow@bigubu: ~/psi-lab1/PSI-projekt/Zad1-2
z35_pserver_zad_12 | Sent ACK0
z35_pclient_zad_12 | Sent packet with sequence 0
z35_pclient_zad_12 | Received ACK0
z35_pserver_zad_12 | Received packet with sequence 1
z35_pclient_zad_12 | Sent packet with sequence 1
z35_pserver_zad_12 | Sent ACK1
z35_pclient_zad_12 | Received ACK1
z35_pserver_zad_12 | Received packet with sequence 0
z35_pserver_zad_12 | Sent ACK0
z35_pclient_zad_12 | Sent packet with sequence 0
z35_pclient_zad_12 | Received ACK0
z35_pclient_zad_12 | Sent packet with sequence 1
z35_pclient_zad_12 | Received ACK1
z35_pserver_zad_12 | Received packet with sequence 1
z35_pserver_zad_12 | Sent ACK1
z35_pserver_zad_12 | Received packet with sequence 0
z35_pclient_zad_12 | Sent packet with sequence 0
z35_pserver_zad_12 | Sent ACK0
z35_pclient_zad_12 | Received ACK0
z35_pclient_zad_12 | Sent packet with sequence 1
z35_pclient_zad_12 | Received ACK1
z35_pserver_zad_12 | Received packet with sequence 1
z35_pserver_zad_12 | Sent ACK1
z35_pserver_zad_12 | Received packet with sequence 0
z35_pserver_zad_12 | Sent ACK0
z35_pclient_zad_12 | Sent packet with sequence 0
z35_pclient_zad_12 | Received ACK0
```

Wysyłanie i odbieranie pakietów z gubieniem (ustawionym na np 50%), uzyskane taką komendą:

```
docker exec z35_pclient_zad_12 tc qdisc add dev
eth0 root netem loss 50%
```

```
sabramow@bigubu:~/psi-lab1/PSI-projekt/Zad1-2$ ./simulate_packet_loss.sh 50
sabramow@bigubu:~/psi-lab1/PSI-projekt/Zad1-2$
```

```
z35_pclient_zad_12 | Received ACK1
z35_pclient_zad_12 | Sent packet with sequence 0
z35_pserver_zad_12 | Timeout waiting for ACK
z35_pclient_zad_12 | Timeout waiting for ACK0
z35_pclient_zad_12 | Retry number: 1
z35_pclient_zad_12 | Sent packet with sequence 0
z35_pclient_zad_12 | Received ACK0
z35_pserver_zad_12 | Received packet with sequence 0
z35_pserver_zad_12 | Sent ACK0
z35_pserver_zad_12 | Received packet with sequence 1
z35_pserver_zad_12 | Sent ACK1
z35_pclient_zad_12 | Sent packet with sequence 1
z35_pclient_zad_12 | Received ACK1
z35_pclient_zad_12 | Sent packet with sequence 0
z35_pclient_zad_12 | Received ACK0
z35_pserver_zad_12 | Received packet with sequence 0
z35_pserver_zad_12 | Sent ACK0
z35_pclient_zad_12 | Sent packet with sequence 1
z35_pserver_zad_12 | Timeout waiting for ACK
z35_pclient_zad_12 | Timeout waiting for ACK1
z35_pclient_zad_12 | Retry number: 1
z35_pclient_zad_12 | Sent packet with sequence 1
z35_pserver_zad_12 | Timeout waiting for ACK
z35_pclient_zad_12 | Timeout waiting for ACK1
z35_pclient_zad_12 | Retry number: 2
z35_pclient_zad_12 | Sent packet with sequence 1
```

Na powyższym zrzucie ekranu widać, jak pakiet nie dociera do serwera, co skutkuje przekroczeniem limitu czasu po stronie serwera. W efekcie serwer nie wysyła potwierdzenia ACK, co prowadzi do kolejnego przekroczenia limitu czasu po stronie klienta. Dla klienta ustawiony jest limit liczby ponownych prób wysyłki pakietu, co można zobaczyć w komunikacie „Retry number: n”. Po kilku próbach klient odzyskuje kontakt z serwerem i transmisja jest kontynuowana.

## Wnioski

Zastosowanie **Bit alternate protocol** pozwoliło na obsługę retransmisji zgubionych pakietów, a mechanizm numerów sekwencyjnych umożliwił synchronizację między klientem a serwerem. Timeouty okazały się niezbędne w wypadku zakleszczenia spowodowanego recv/recvfrom.