# Programowanie Sieciowe - Z 1.2

Jan Lewandowski
Szymon Ryszka
Krzysztof Sokół

23 Listopada 2024

## 1 Treść zadania

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).

## 2 Rozwiązanie problemu

### 2.1 Skrócony opis rozwiązania

Kod implementuje uproszczoną wersję protokołu przesyłania danych opartego na alternatywnym potwierdzaniu (Bit Alternating Protocol - BAP). Składa się z dwóch plików: `client1_2.py` (klient) i `server1_.py` (serwer), które komunikują się za pomocą protokołu UDP. Komunikacja polega na wymianie pakietów z numerami sekwencyjnymi i potwierdzeniami, by zagwarantować poprawność przesyłu danych.

## 2.2 Zmiany w kodzie względem poprzedniego zadania po stronie klienta

Do wykonania tego zadania użyliśmy kodu w pythonie. Zmiana przedstawiona poniżej dotyczy funkcji `send_data()`.

Zmiany w funkcji `send_data()`

```
try:
    packet_num = int(response[3:6])
    if packet_num != n:
        print(f"received {packet_num} does not equal sent
            {n}")
        return -1
except ValueError:
    print("incorrect response format : packet number")
    end(s)
if(response[:3] == b"MIS"):
    print(f"packet {packet_num} have not arrived!")
    return -3
elif(response[:3] != b"ACK"):
    print("incorrect response format : ACK")
    end(s)
```

Opis kodu:

- Sprawdza poprawność numeru pakietu odebranego od serwera
- Odbiera odpowiedź od serwera i sprawdza jej poprawność
  - Jeśli odpowiedź to `MIS<nr>`: wykryto brak pakietu
  - Jeśli odpowiedź to `ACK<nr>`: pakiet został odebrany poprawnie

Zmiany w funkcji `setup()`

```
# Fixed-length message
msg_size = 512
payload = "A" * (msg_size - 10)
time.sleep(4)

# send 12 packets
n = 1
timeout= 0
while n <= 12:
    if timeout > 10:
        print("server not responding, aborting")
        end(s)
    msg = "packet" + str(n).zfill(3) + ' ' + payload
    res = send_data(s, msg, n)
    if res == -1:
        print("failed verification, retrying...")
        continue
    if res == -2:
        print("did not receive confirmation, retrying...")
        timeout += 1
        continue
    if res == -3:
        print("re-sending missing packet")
        n -= 1
        continue
    timeout = 0
    n+=1
```

Opis kodu:

- Tworzy gniazdo UDP i ustawia timeout.
- Generuje i wysyła 12 pakietów z numerami od 1 do 12.
- Obsługuje ponowne wysyłanie pakietów w przypadku braku odpowiedzi lub problemów.

## 2.3 Komunikacja po stronie klienta

- Tworzy pakiety o stałej wielkości (512 B, które zawierają numer pakietu i dane)

- Wysyła pakiet do serwera
- Czeka na potwierdzenie od serwera (`ACK`) o poprawnym odbiorze pakietu lub informację o brakującym pakiecie (`MIS`)
- Powtarza transmisję w przypadku problemów (błędy formatu, brak odpowiedzi, utrata pakietu).

## 2.4 Zmiany w kodzie względem poprzedniego zadania po stronie serwera

Zmiany w funkcji `recv_data()`

```
# Verify packet size
size_rec = sys.getsizeof(data) - 33
if size_rec != expected_size:
    print(f"unexpected packet size: {size_rec} bytes
        (expected {expected_size})")
    continue

if data[:6] != b"packet":
    print("error: missing or incorrect prefix")
    continue
cur_packet = int(data[6:9])
if cur_packet <= last_packet:
    print("incorrect packet number")
    continue
if cur_packet > last_packet+1:
    print(f"packet {last_packet+1} lost!")
    s.sendto(str.encode(f"MIS{last_packet+1}"), ret_addr)
    continue
print(f"packet {data[0:9]} verified")
last_packet = int(data[6:9])
print(f"received {size_rec} bytes from {ret_addr}")
s.sendto(str.encode(f"ACK{last_packet}"), ret_addr)
```

Opis kodu:

- Odbiera pakiety i sprawdza ich poprawność:
  - Sprawdza rozmiar, format danych i numer pakietu
  - Jeśli pakiet jest poprawny wysyła `ACK<nr>`

  - Jeśli wykryje brakujący pakiet wysyła `MIS<nr>`

### 2.5 Komunikacja po stronie serwera

- Nasłuchuje na porcie UDP.
- Odbiera pakiety od klienta.
- Sprawdza poprawność pakietu (rozmiar, numer sekwencji).
  - `ACK<nr>` - potwierdzenie odbioru pakietu.
  - `MIS<nr>` - zgłoszenie nieodebrania pakietu o numerze niższym niż obecny. Operacja nadmiarowa dla algorytmu stop and wait.

## 3 Opis konfiguracji testowej

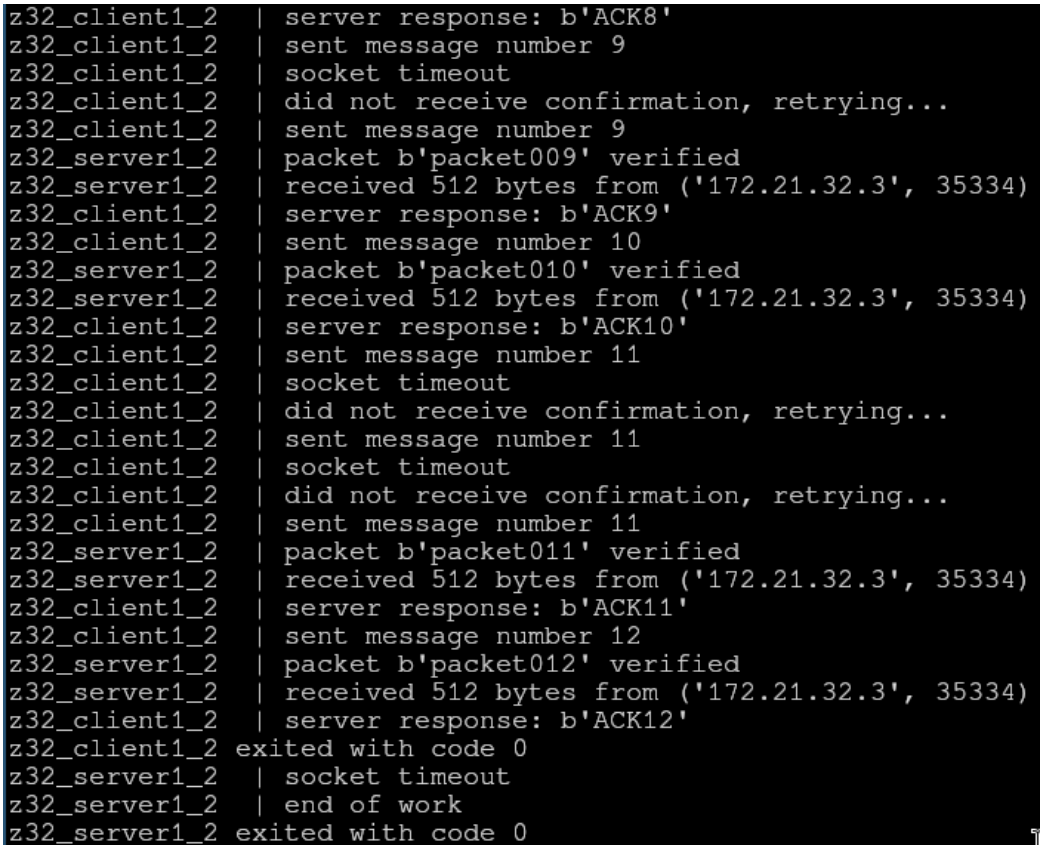
Dla kontenera `z_32` skonfigurowano warunki sieciowe:

- opóźnienie średnio 1000 ms z wariancją 500 ms,
- utrara pakietów 33%,
- wykorzystany adres ip: 127.0.0.1,
- wykorzystany port: 3201,
- do manipulowania przepustowością korzystalismy z komendy: `tc qdisc` służącej do dodawania i zarządzania dyscyplin kolejkowania w interfejsach sieciowych.

# 4 Testowanie

```
sryszka@bigubu:~/PSI_2024Z_Z32/Z1.2$ cat result1_2.txt
Attaching to z32_client1_2, z32_server1_2
z32_client1_2  | created socket
z32_server1_2  | server listening on z32_server1_2:3201
z32_client1_2  | sent message number 1
z32_client1_2  | socket timeout
z32_client1_2  | did not receive confirmation, retrying...
z32_client1_2  | sent message number 1
z32_server1_2  | packet b'packet001' verified
z32_server1_2  | received 512 bytes from ('172.21.32.3', 35334)
z32_client1_2  | server response: b'ACK1'
z32_client1_2  | sent message number 2
z32_server1_2  | packet b'packet002' verified
z32_server1_2  | received 512 bytes from ('172.21.32.3', 35334)
z32_client1_2  | server response: b'ACK2'
z32_client1_2  | sent message number 3
z32_server1_2  | packet b'packet003' verified
z32_server1_2  | received 512 bytes from ('172.21.32.3', 35334)
z32_client1_2  | server response: b'ACK3'
z32_client1_2  | sent message number 4
z32_server1_2  | packet b'packet004' verified
z32_server1_2  | received 512 bytes from ('172.21.32.3', 35334)
z32_client1_2  | server response: b'ACK4'
z32_client1_2  | sent message number 5
z32_server1_2  | packet b'packet005' verified
z32_server1_2  | received 512 bytes from ('172.21.32.3', 35334)
z32_client1_2  | server response: b'ACK5'
z32_client1_2  | sent message number 6
z32_client1_2  | socket timeout
z32_client1_2  | did not receive confirmation, retrying...
z32_client1_2  | sent message number 6
z32_server1_2  | packet b'packet006' verified
z32_server1_2  | received 512 bytes from ('172.21.32.3', 35334)
z32_client1_2  | server response: b'ACK6'
z32_client1_2  | sent message number 7
z32_client1_2  | socket timeout
z32_client1_2  | did not receive confirmation, retrying...
z32_client1_2  | sent message number 7
z32_client1_2  | socket timeout
z32_client1_2  | did not receive confirmation, retrying...
z32_client1_2  | sent message number 7
z32_server1_2  | packet b'packet007' verified
z32_server1_2  | received 512 bytes from ('172.21.32.3', 35334)
z32_client1_2  | server response: b'ACK7'
z32_client1_2  | sent message number 8
z32_client1_2  | socket timeout
z32_client1_2  | did not receive confirmation, retrying...
z32_client1_2  | sent message number 8
z32_server1_2  | packet b'packet008' verified
z32_server1_2  | received 512 bytes from ('172.21.32.3', 35334)
z32_client1_2  | server response: b'ACK8'
```

Rysunek 1: Wynik testowania




Rysunek 2: Wynik testowania

Powyższe zrzuty ekranu przedstawiają wynik jednego z uruchomień programów. Można na nich zauważyć następujące informacje:

- wszystkie wiadomości od 1 do 12 zostały odebrane,
- w przypadku gdy klient nie otrzymał odpowiedzi potwierdzającej otrzymanie wiadomości wysyłał ją po raz kolejny (np. jak dla pierwszej wiadomości)
- wszystkie pakiety docierają w odpowiedniej kolejności

# 5 Wnioski końcowe

Implementacja protokołu przesyłania danych z mechanizmem wykrywania utraconych pakietów pokazuje, jak niezawodność można osiągnąć w środowisku o wysokiej zawodności sieci. Testowanie w symulowanych warunkach

(opóźnienia, utraty pakietów) pozwoliło na weryfikację poprawności i stabilności protokołu.