PSI laboratorium Z 1.2
Zespół 34

Lider - Miłosz Andryszczuk
Aleksander Paliwoda
Maksymilian Zieliński

Z 1.2
Klient ma za zadanie odczytać plik z dysku (proszę wygenerować plik z losowymi 10000B) i wysłać do serwera jego zawartość w paczkach po 100B. Serwer ma zrekonstruować cały plik i obliczyć jego hash. Jako dowód działania proszę m.in. porównać hash obliczony przez serwer z hashem obliczonym przez klienta (może to być wydrukowane w konsoli klienta/serwera, hashe muszą być identyczne). Należy zaimplementować prosty protokół niezawodnej transmisji, uwzględniający możliwość gubienia datagramów. Gubione pakiety muszą być wykrywane i retransmitowane aby serwer mógł odtworzyć cały plik. Należy uruchomić program w środowisku symulującym błędy gubienia pakietów. (Informacja o tym, jak to zrobić znajduje się w skrypcie opisującym środowisko Dockera).

Specyfikacja pakietu: Aby zrealizować transmisję, zdefiniowano prostą strukturę ramki aplikacyjnej przesyłanej w datagramie UDP. Każdy pakiet ma rozmiar 104 bajtów:

- **Nagłówek (4 bajty):** Numer sekwencyjny pakietu (Sequence Number) zapisany jako 32-bitowa liczba całkowita w formacie sieciowym (Big-Endian). Pozwala to serwerowi ułożyć pakiety w odpowiedniej kolejności.
- **Dane (100 bajtów):** Właściwy fragment przesyłanego pliku (Payload).

Działanie programu po stronie klienta w C:

1. Klient generuje losowy plik o rozmiarze 10000 bajtów.
2. Oblicza dla niego hash SHA-256
3. Dzieli plik na 100 pakietów po 100 bajtów
4. Wszystkie pakiety wysyła bez potwierdzeń (burst)
5. Wysyła pakiet FIN i oczekuje na odpowiedź
    - ACK - transmisja zakończona
    - NAK - klient retransmituje brakujące fragmenty i ponawia FIN.
6. Po zakończeniu wypisuje hash wysłanego pliku

Fragment kodu odpowiadający za transmisję oraz retransmisję:

```
void run_transfer(int sockfd, char *packets_buffer) {
  printf("Sending all %d packets...\n", NUM_CHUNKS);
  for (int i = 0; i < NUM_CHUNKS; i++) {
    char *packet_ptr = packets_buffer + (i * PACKET_SIZE);
    send(sockfd, packet_ptr, PACKET_SIZE, 0);
  }
  printf("Packets sent.\n");

  int32_t fin_packet = htonl(FIN_PACKET);
  char recv_buffer[4096];

  while (1) {
    printf("Sending FIN packet (-1)...\n");
    send(sockfd, &fin_packet, sizeof(fin_packet), 0);

    int n = recv(sockfd, recv_buffer, 4096, 0);

    if (n < 0) {
      if (errno == EAGAIN || errno == EWOULDBLOCK) {
        printf("Timeout (%ds) waiting for server ACK/NAK.\n",
CLIENT_TIMEOUT);
        continue;
      } else {
        perror("recvfrom error");
        break;
      }
```

```c
    }

    if (n < 4)
      continue;

    int32_t resp_type = ntohl(*(int32_t *)recv_buffer);

    if (resp_type == 0) {
      printf("\nServer received all packets. File transfer complete.\n");
      break;
    } else if (resp_type > 0) {
      int num_missing = resp_type;
      printf("Server sent NAK. Missing %d packets.\n", num_missing);

      if (n < 4 + (num_missing * 4)) {
        printf("Received incomplete NAK packet!\n");
        continue;
      }

      for (int i = 0; i < num_missing; i++) {
        int32_t missing_seq = ntohl(*(int32_t *)(recv_buffer + 4 + (i *
4)));

        if (missing_seq < 0 || missing_seq >= NUM_CHUNKS) {
          printf("Server requested non-existent packet %d\n",
missing_seq);
          continue;
        }

        char *packet_ptr = packets_buffer + (missing_seq * PACKET_SIZE);
        send(sockfd, packet_ptr, PACKET_SIZE, 0);
      }
    }
  }
}
```

Działanie programu po stronie serwera w Pythonie:

Serwer umieszcza pakiety bezpośrednio w tablicy file_buffer na podstawie numeru sekwencyjnego. Odrzuca pakiety o nieprawidłowej długości. Po otrzymaniu pakietu FIN serwer wysyła:

- jeśli otrzymał wszystkie pakiety:
  ACK -> `struct.pack('!i', 0)`
- jeśli brakuje pakietów:
  NAK -> `struct.pack(f'!i{missing_count}i', *args_to_pack)`

Po zebraniu wszystkich fragmentów wylicza hash SHA-256 z odtworzonego bufora i wypisuje wynik.

Fragment kodu odpowiadający za odbiór po stronie serwera:

```
seq_number = struct.unpack('!i', data[:4])[0]

            if 0 <= seq_number < NUM_CHUNKS:
                if len(data) == PACKET_SIZE and not 
received_chunks[seq_number]:
                    start = seq_number * CHUNK_SIZE
                    end = start + CHUNK_SIZE
                    file_buffer[start:end] = data[4:]

                    received_chunks[seq_number] = True
                    total_received += 1
                    print(f"Received packet {seq_number}. Total: 
{total_received}/{NUM_CHUNKS}")

            elif seq_number == FIN_PACKET:
                if handle_fin_packet(sock, client_addr, received_chunks):
                    break
```

Opis konfiguracji testowej:

Kontener serwera

- nazwa kontenera: z34_server
- port używany przez aplikację: UDP 8888
- kontener działa w sieci z34_network

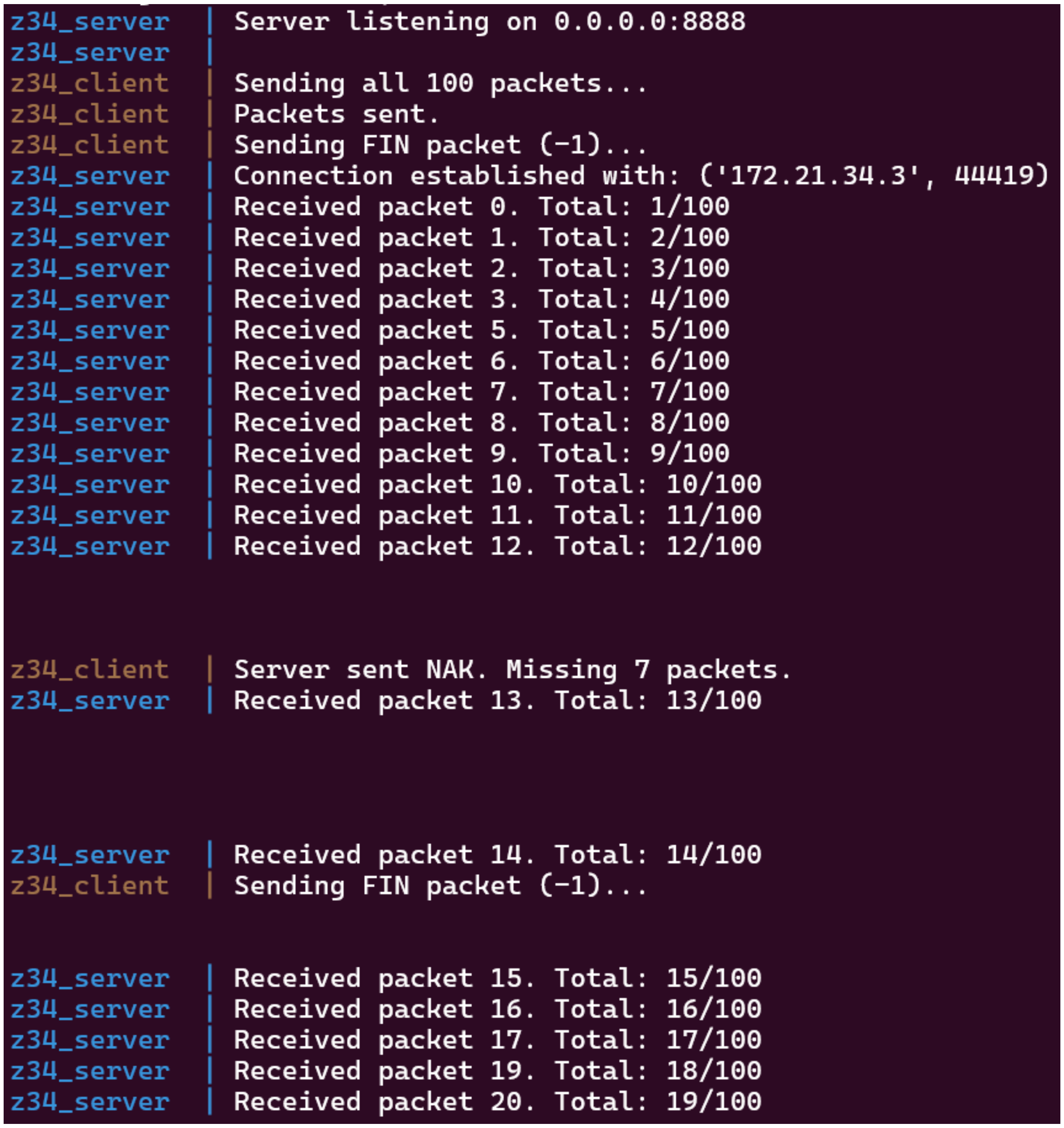
Kontener klienta

- nazwa kontenera: z34_client
- klient łączy się z serwerem poprzez nazwę hosta server na porcie UDP 8888
- kontener posiada uprawnienie NET_ADMIN, konieczne do konfiguracji zakłóceń sieciowych tc
- klient startuje po 1 s opóźnienia, następnie ustawia parametry netem.

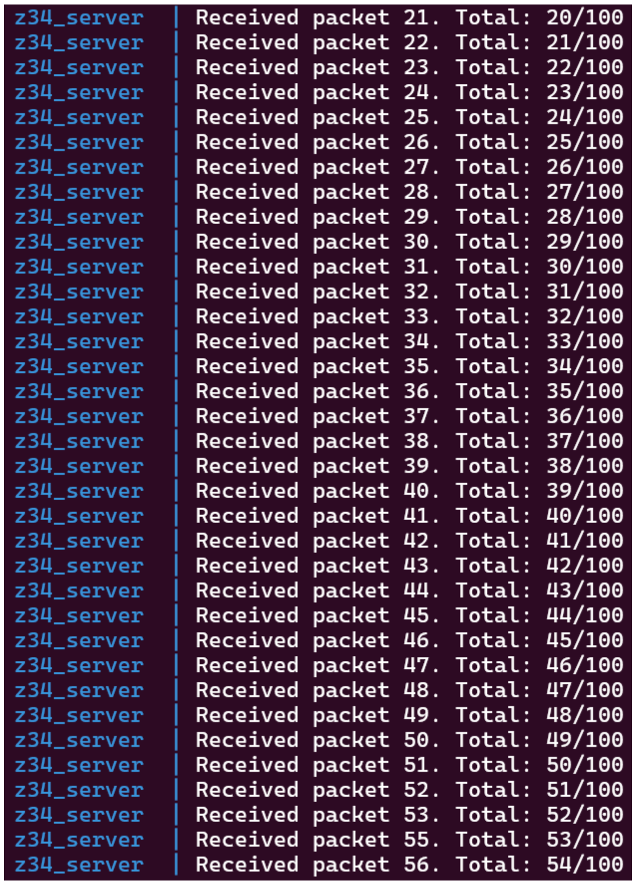
Parametry testowe:

- wymuszone gubienie ok. 10% pakietów,
- dodatkowe opóźnienie 50 ms na każdy pakiet,
- umożliwia to sprawdzenie działania protokołu retransmisji

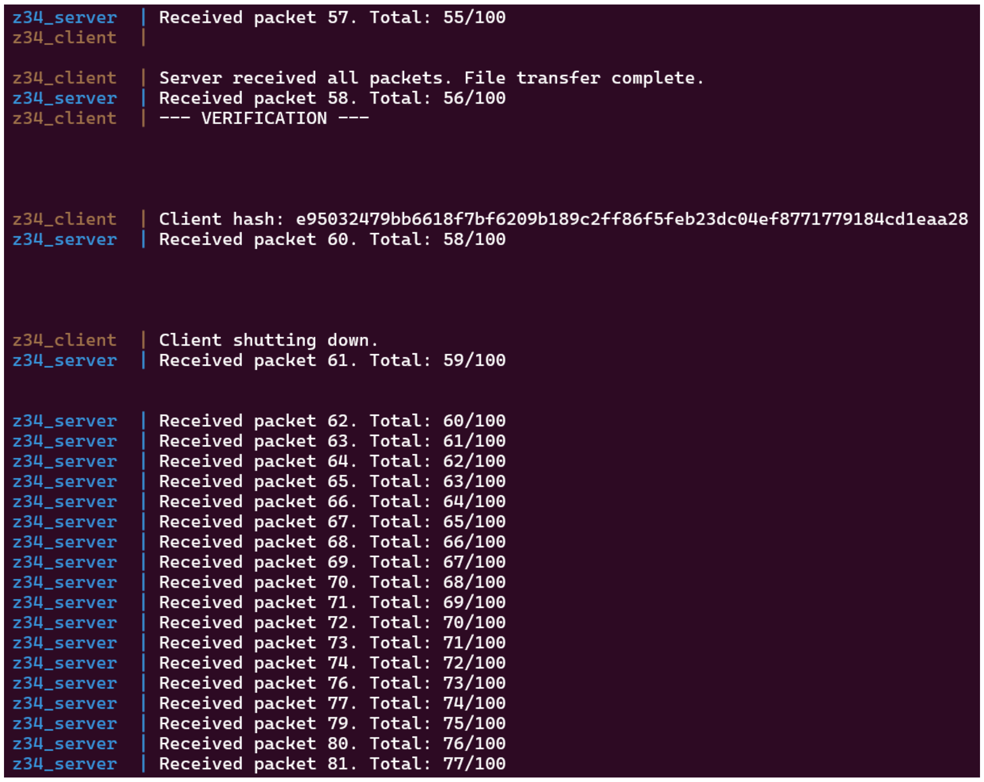
Działanie programu:



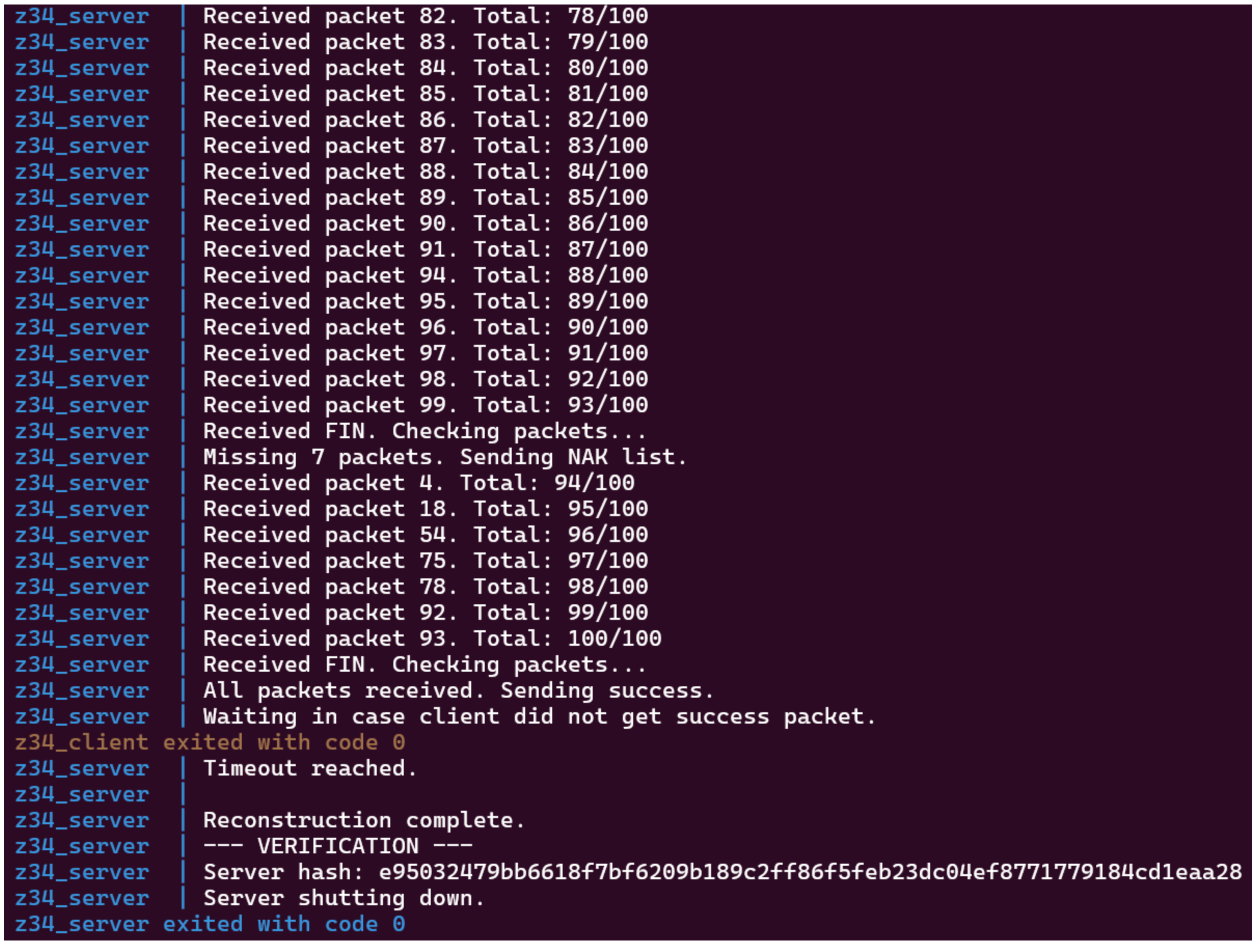
```
z34_server    | Server listening on 0.0.0.0:8888
z34_server    |
z34_client    | Sending all 100 packets...
z34_client    | Packets sent.
z34_client    | Sending FIN packet (-1)...
z34_server    | Connection established with: ('172.21.34.3', 44419)
z34_server    | Received packet 0. Total: 1/100
z34_server    | Received packet 1. Total: 2/100
z34_server    | Received packet 2. Total: 3/100
z34_server    | Received packet 3. Total: 4/100
z34_server    | Received packet 5. Total: 5/100
z34_server    | Received packet 6. Total: 6/100
z34_server    | Received packet 7. Total: 7/100
z34_server    | Received packet 8. Total: 8/100
z34_server    | Received packet 9. Total: 9/100
z34_server    | Received packet 10. Total: 10/100
z34_server    | Received packet 11. Total: 11/100
z34_server    | Received packet 12. Total: 12/100



z34_client    | Server sent NAK. Missing 7 packets.
z34_server    | Received packet 13. Total: 13/100




z34_server    | Received packet 14. Total: 14/100
z34_client    | Sending FIN packet (-1)...


z34_server    | Received packet 15. Total: 15/100
z34_server    | Received packet 16. Total: 16/100
z34_server    | Received packet 17. Total: 17/100
z34_server    | Received packet 19. Total: 18/100
z34_server    | Received packet 20. Total: 19/100
```



```
z34_server  | Received packet 21. Total: 20/100
z34_server  | Received packet 22. Total: 21/100
z34_server  | Received packet 23. Total: 22/100
z34_server  | Received packet 24. Total: 23/100
z34_server  | Received packet 25. Total: 24/100
z34_server  | Received packet 26. Total: 25/100
z34_server  | Received packet 27. Total: 26/100
z34_server  | Received packet 28. Total: 27/100
z34_server  | Received packet 29. Total: 28/100
z34_server  | Received packet 30. Total: 29/100
z34_server  | Received packet 31. Total: 30/100
z34_server  | Received packet 32. Total: 31/100
z34_server  | Received packet 33. Total: 32/100
z34_server  | Received packet 34. Total: 33/100
z34_server  | Received packet 35. Total: 34/100
z34_server  | Received packet 36. Total: 35/100
z34_server  | Received packet 37. Total: 36/100
z34_server  | Received packet 38. Total: 37/100
z34_server  | Received packet 39. Total: 38/100
z34_server  | Received packet 40. Total: 39/100
z34_server  | Received packet 41. Total: 40/100
z34_server  | Received packet 42. Total: 41/100
z34_server  | Received packet 43. Total: 42/100
z34_server  | Received packet 44. Total: 43/100
z34_server  | Received packet 45. Total: 44/100
z34_server  | Received packet 46. Total: 45/100
z34_server  | Received packet 47. Total: 46/100
z34_server  | Received packet 48. Total: 47/100
z34_server  | Received packet 49. Total: 48/100
z34_server  | Received packet 50. Total: 49/100
z34_server  | Received packet 51. Total: 50/100
z34_server  | Received packet 52. Total: 51/100
z34_server  | Received packet 53. Total: 52/100
z34_server  | Received packet 55. Total: 53/100
z34_server  | Received packet 56. Total: 54/100
```

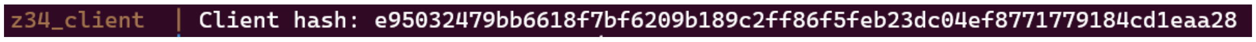


```
z34_server  | Received packet 57. Total: 55/100
z34_client  |

z34_client  | Server received all packets. File transfer complete.
z34_server  | Received packet 58. Total: 56/100
z34_client  | --- VERIFICATION ---




z34_client  | Client hash: e95032479bb6618f7bf6209b189c2ff86f5feb23dc04ef8771779184cd1eaa28
z34_server  | Received packet 60. Total: 58/100




z34_client  | Client shutting down.
z34_server  | Received packet 61. Total: 59/100


z34_server  | Received packet 62. Total: 60/100
z34_server  | Received packet 63. Total: 61/100
z34_server  | Received packet 64. Total: 62/100
z34_server  | Received packet 65. Total: 63/100
z34_server  | Received packet 66. Total: 64/100
z34_server  | Received packet 67. Total: 65/100
z34_server  | Received packet 68. Total: 66/100
z34_server  | Received packet 69. Total: 67/100
z34_server  | Received packet 70. Total: 68/100
z34_server  | Received packet 71. Total: 69/100
z34_server  | Received packet 72. Total: 70/100
z34_server  | Received packet 73. Total: 71/100
z34_server  | Received packet 74. Total: 72/100
z34_server  | Received packet 76. Total: 73/100
z34_server  | Received packet 77. Total: 74/100
z34_server  | Received packet 79. Total: 75/100
z34_server  | Received packet 80. Total: 76/100
z34_server  | Received packet 81. Total: 77/100
```

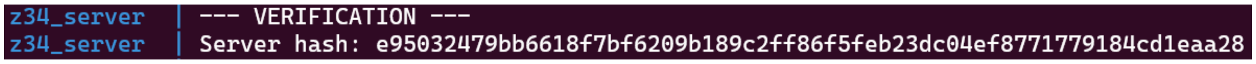


```
z34_server   | Received packet 82. Total: 78/100
z34_server   | Received packet 83. Total: 79/100
z34_server   | Received packet 84. Total: 80/100
z34_server   | Received packet 85. Total: 81/100
z34_server   | Received packet 86. Total: 82/100
z34_server   | Received packet 87. Total: 83/100
z34_server   | Received packet 88. Total: 84/100
z34_server   | Received packet 89. Total: 85/100
z34_server   | Received packet 90. Total: 86/100
z34_server   | Received packet 91. Total: 87/100
z34_server   | Received packet 94. Total: 88/100
z34_server   | Received packet 95. Total: 89/100
z34_server   | Received packet 96. Total: 90/100
z34_server   | Received packet 97. Total: 91/100
z34_server   | Received packet 98. Total: 92/100
z34_server   | Received packet 99. Total: 93/100
z34_server   | Received FIN. Checking packets...
z34_server   | Missing 7 packets. Sending NAK list.
z34_server   | Received packet 4. Total: 94/100
z34_server   | Received packet 18. Total: 95/100
z34_server   | Received packet 54. Total: 96/100
z34_server   | Received packet 75. Total: 97/100
z34_server   | Received packet 78. Total: 98/100
z34_server   | Received packet 92. Total: 99/100
z34_server   | Received packet 93. Total: 100/100
z34_server   | Received FIN. Checking packets...
z34_server   | All packets received. Sending success.
z34_server   | Waiting in case client did not get success packet.
z34_client exited with code 0
z34_server   | Timeout reached.
z34_server   |
z34_server   | Reconstruction complete.
z34_server   | --- VERIFICATION ---
z34_server   | Server hash: e95032479bb6618f7bf6209b189c2ff86f5feb23dc04ef8771779184cd1eaa28
z34_server   | Server shutting down.
z34_server exited with code 0
```

Porównanie hashy:



```
z34_client   | Client hash: e95032479bb6618f7bf6209b189c2ff86f5feb23dc04ef8771779184cd1eaa28
```



```
z34_server   | --- VERIFICATION ---
z34_server   | Server hash: e95032479bb6618f7bf6209b189c2ff86f5feb23dc04ef8771779184cd1eaa28
```

Hashe się zgadzają.

Wnioski końcowe:

- Zaimplementowany protokół niezawodnego przesyłu (bazujący na mechanizmie Selective Repeat ARQ) działa poprawnie w warunkach zawodnej sieci UDP.

- Mechanizm FIN + NAK okazał się wydajny: zamiast retransmitować cały plik, klient wysyłał ponownie jedynie te fragmenty, które rzeczywiście zaginęły (w teście było to około 8-10% pakietów).
- Zastosowanie sumy kontrolnej SHA-256 po obu stronach (przed wysyłką i po rekonstrukcji) pozwoliło jednoznacznie potwierdzić integralność danych.
- Testy w środowisku z opóźnieniem (delay 50ms) wykazały, że mechanizm timeoutu po stronie klienta jest niezbędny, aby zapobiec zawieszeniu się aplikacji w przypadku utraty pakietów kontrolnych (np. zagubienie potwierdzenia ACK od serwera).