

PSI – KOMUNIKACJA TCP (Zad 1.2)

Zespół 32: **Katarzyna Kanicka**, Jan Mizera, Andrii-Stepan Pryimak

Treść zadania

Mieliśmy zaimplementować mechanizm niezawodnej transmisji pliku binarnego o wielkości 10 000 bajtów przy użyciu protokołu UDP. System składa się z klienta (napisanego w języku C) oraz serwera (w języku Python). Kluczowe wymagania obejmowały:

- Podział pliku na fragmenty (paczki) po 100 bajtów.
- Implementację mechanizmu potwierdzeń i retransmisji w celu obsługi gubionych pakietów (symulacja awaryjności łącza).
- Weryfikację integralności przesłanego pliku poprzez obliczenie i porównanie sum kontrolnych (hash MD5) po stronie nadawcy i odbiorcy.

Opis rozwiązania problemu

Zaimplementowano mechanizm **Stop-and-Wait ARQ** z bitem naprzemiennym (Alternating Bit Protocol).

Format pakietu: 1 bajt nagłówka (numer sekwencji 0 lub 1) + 100 bajtów danych.

Klient (C):

- Wysyła pakiet i uruchamia timer (opcja gniazda SO_RCVTIMEO ustawiona na 2s).
- Czeka na potwierdzenie (ACK) o zgodnym numerze sekwencji.
- W przypadku **timeoutu** (brak ACK) – retransmituje ostatni pakiet.

Serwer (Python):

- Odbiera pakiet. Jeśli nr sekwencji jest oczekiwany -> zapisuje dane i zmienia oczekiwany nr.
- Jeśli nr sekwencji jest zły (duplikat) -> ignoruje dane.
- Zawsze odsyła ACK z otrzymanym numerem sekwencji.
- Po odebraniu kompletu danych oblicza hash MD5.

Klient:

```
68     struct timeval timeout;
69     int max_retries = 10;
70
71     timeout.tv_sec = 2;
72     timeout.tv_usec = 0;
73
74     while (i < max_retries) {
75         int nread = recvfrom(sock, &ack, 1, 0, NULL, NULL);
76
77         if (nread > 0 && ack == current_seq) {
78             printf("Received ACK: %d\n", ack);
79             current_seq = !current_seq;
80             i++;
81         }
82         else {
83             printf("[TIMEOUT] Resending packet %d\n", i);
84         }
85     }
```

Server:

```
18     while True:
19         data, addr = s.recvfrom(BUFSIZE)
20
21         seq_rcv = data[0]
22         payload = data[1:]
23
24         if seq_rcv == seq_expected:
25             print(f"Packet {recv_packets} from {addr}: Seq={seq_rcv}, Payload Size={len(payload)} bytes")
26
27             received_data.extend(payload)
28
29             seq_expected = 1 - seq_expected
30             recv_packets += 1
31         else:
32             print("Duplicated data. Resending ACK")
33
```

```
34     ack = bytes([seq_rcv])
35     s.sendto(ack, addr)
```

```
41
42     print("-" * 30)
43     md5_hash = hashlib.md5(received_data).hexdigest()
44     print(f"Server MD5 Hash: {md5_hash}")
45
```

Konfiguracja testowa

```
zad1-2 > client > $ run.sh
1  #!/bin/bash
2
3  docker rm -f z32-client-c
4  docker build -t z32-client-docker .
5  docker run -it --cap-add=NET_ADMIN --network z32_network --name z32-client-c z32-client-docker
```

```
8  #define HOST "z32-server-python"
9  #define PORT 5000
10 #define FILE_NAME "random.bin"
11 #define FILE_SIZE 10000
12 #define PACKET_DATA_SIZE 100
13 #define PACKET_SIZE (PACKET_DATA_SIZE + 1)
14
```

```
zad1-2 > server > $ run.sh
1  #!/bin/bash
2
3  docker rm -f z32-server-python
4  docker build -t z32-server-docker .
5  docker run -it --network-alias z32-server-python --network z32_network --name z32-server-python z32-server-docker
6
```

```
4  HOST = "0.0.0.0"
5  PORT = 5000
6  BUFSIZE = 1024
7  N_PACKETS = 100
8
```

```
zad1-2 > $ disrupt.sh
1  #!/bin/bash
2
3  docker exec z32-client-c tc qdisc add dev eth0 root netem delay 1000ms 500ms loss 50%
```

Serwer: z32-server-python (Docker) **Klient:** z32-client-c (Docker)

Sieć: Docker network z32_network **Port:** 5000

Symulacja błędów: Skrypt disrupt.sh uruchamiany na kontenerze klienta (wprowadza losowe gubienie pakietów).

Testowanie i wyniki

Test wykazał poprawność mechanizmu retransmisji. Klient wykrył brak odpowiedzi i ponowił wysyłanie.

Fragment logów (wykrycie błędu):

Klient:

```
Sent Packet 96: Seq=0
[TIMEOUT] Resending packet 96
Sent Packet 96: Seq=0
Received ACK: 0
Sent Packet 97: Seq=1
[TIMEOUT] Resending packet 97
Sent Packet 97: Seq=1
Received ACK: 1
Sent Packet 98: Seq=0
Received ACK: 0
Sent Packet 99: Seq=1
[TIMEOUT] Resending packet 99
Sent Packet 99: Seq=1
Received ACK: 1
Successfully sent all packets
```

Server:

```
Packet 95 from ('172.18.0.3', 39748): Seq=1, Payload Size=100 bytes
Packet 96 from ('172.18.0.3', 39748): Seq=0, Payload Size=100 bytes
Packet 97 from ('172.18.0.3', 39748): Seq=1, Payload Size=100 bytes
Packet 98 from ('172.18.0.3', 39748): Seq=0, Payload Size=100 bytes
Packet 99 from ('172.18.0.3', 39748): Seq=1, Payload Size=100 bytes
Received all packets
```

Weryfikacja integralności (MD5): Porównano sumy kontrolne pliku wysłanego i odebranego.

```
Successfully sent all packets
```

```
-----
```

```
Client MD5 Hash: a08a77229aad94213dd9ec22c2ff4897
```

```
Received all packets
```

```
-----
```

```
Server MD5 Hash: a08a77229aad94213dd9ec22c2ff4897
```

Problemy napotkane i rozwiązania

W trakcie realizacji zadania nie wystąpiły istotne problemy techniczne ani trudności implementacyjne. Kod serwera w Pythonie oraz klienta w C działały poprawnie.

Wnioski

Zaimplementowany mechanizm Stop-and-Wait zapewnia poprawną i kompletną transmisję pliku w środowisku stratnym. Choć protokół ten nie jest wydajny (ze względu na konieczność oczekiwania na potwierdzenie każdego pakietu), jest prosty w implementacji i skuteczny dla małych plików przy umiarkowanym opóźnieniu sieci (RTT). Zastosowanie sum kontrolnych MD5 jednoznacznie potwierdziło integralność przestanych danych.