PSI – Sprawozdanie do zad 1.2

Autorzy:
Alesia Filinkova
Weronika Maślana
Diana Pelin

Z 1.2 Komunikacja UDP

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).

1. Opis rozwiązania problemu

Celem zadania było zaimplementowanie uproszczonego protokołu niezawodnej transmisji działającego w oparciu o datagramy UDP oraz obsługującego gubienie pakietów. Klient ma odczytać plik o rozmiarze 10 000 bajtów i przesłać go w paczkach 100-bajtowych do serwera. Serwer odbiera datagramy, rekonstruuje plik i oblicza jego sumę kontrolną SHA-256. Aby udowodnić poprawność transmisji, należy porównać hash obliczony przez serwer z hashem pliku źródłowego po stronie klienta. Transmitowana komunikacja miała działać w środowisku symulującym losowe gubienie pakietów UDP. Ponieważ UDP może gubić pakiety, należało zaprojektować własny mechanizm potwierdzeń ACK i retransmisji. Środowisko testowe umożliwiało symulowanie strat pakietów poprzez parametr --loss w serwerze.

2.1. Koncepcja protokołu

Zaimplementowany został prosty protokół niezawodnej transmisji typu *Stop-and-Wait ARQ*:

1. Klient wysyła datagram zawierający:

    a. numer sekwencyjny (4 B),
    b. długość ładunku (2 B),
    c. ładunek do 100 B.
2. Serwer po odebraniu poprawnego segmentu odsyła ACK (4 B) zawierający numer sekwencyjny.
3. Klient oczekuje na ACK z limitem czasu 3 s.
    a. Brak ACK → retransmisja (maks. 10 prób).
    b. ACK z innym numerem – ignorowanie i retransmisja.
4. Serwer buforuje segmenty według numeru sekwencyjnego.
5. Po odebraniu 100 segmentów serwer odtwarza cały plik i oblicza SHA-256.

2.2. Realizacja w Pythonie (serwer)

Serwer UDP został zaimplementowany w Pythonie z użyciem `socket`.

```
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((args.host, args.port))
print(f"Server listening on {args.host}:{args.port}, loss={args.loss}%")
```

Najważniejszy fragment odpowiedzialny za obsługę strat pakietów:

```
if random.randrange(100) < args.loss:
        print(f"Simulated drop of packet from {addr}")
        continue
```

Obsługa segmentu:

```
    seq, plen = struct.unpack("!IH", data[:6])
    payload = data[6:6+plen]

    if len(payload) != plen:
        print(f"Payload length mismatch for seq {seq}")
        continue

    if seq not in chunks:
        chunks[seq] = payload
        print(f"Received seq={seq} len={plen} total_received={len(chunks)}")

    ack = struct.pack("!I", seq)
    sock.sendto(ack, addr)
```

Po odebraniu 100 segmentów serwer zapisuje plik i oblicza SHA-256.

2.3. Realizacja w C (klient)

Klient używa nazw kontenerów Docker jako nazw hostów w tej samej sieci bridge:

```
struct addrinfo hints, *res;
    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_DGRAM;

    int err = getaddrinfo(server_name, port_str, &hints, &res);
    if(err) { fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(err)); return
1; }
```

Każdy fragment jest pakowany do struktury:

```
char packet[6 + CHUNK_SIZE];
        *((unsigned int*)packet) = htonl(seq);
        *((unsigned short*)(packet+4)) = htons(n);
        memcpy(packet+6, buffer, n);
```

Mechanizm niezawodności:

```
fd_set fds;
            struct timeval tv = {3,0};
            FD_ZERO(&fds);
            FD_SET(sock, &fds);

            int r = select(sock+1, &fds, NULL, NULL, &tv);
```

- jeśli `select` zwróci 0 → timeout → retransmisja
- jeśli serwer odeśle ACK ze zgodnym numerem seq → przejście do kolejnego fragmentu

Każdy pakiet jest zatem transmitowany dopóki nie zostanie potwierdzony.

2. Opis konfiguracji testowej

Do testów rozwiązania wykorzystano dwa kontenery Docker uruchomione na serwerze bigubu i połączone we wspólnej sieci Docker z31_network, zgodnie z wymaganiami środowiska laboratoryjnego.

- Serwer TCP
  - nazwa kontenera: z31_server12
  - obraz: z31_server12
  - język: Python 3
  - port nasłuchujący: 5005/UDP
  - sieć: z31_network

- Klient TCP
    - nazwa kontenera: z31_client12
    - obraz: z31_client12
    - język: C (gcc)
    - sieć: z31_network
    - serwer dostępny pod hostname: z31_server12

3. Opis wyników

Serwer poprawnie odebrał całkiem plik

```
afilinko@bigubu:~/zad12$ docker cp z31_server12:/app/received.bin ./received.bin
Successfully copied 11.8kB to /home/users/afilinko/zad12/received.bin
afilinko@bigubu:~/zad12$ sha256sum plik.bin received.bin
50e8ea46e83be16ff02941cd87c56c6d52fbcd381cb074955e0938f043e3b992  plik.bin
50e8ea46e83be16ff02941cd87c56c6d52fbcd381cb074955e0938f043e3b992  received.bin
```

Nano plik.bin	Nano received.bin
	

afilinko@bigubu:~/zad12$ docker logs -f z31_server12 Server listening on 0.0.0.0:5005, loss=0%

Received seq=0 len=100 total_received=1 Received seq=1 len=100 total_received=2
Received seq=2 len=100 total_received=3 Received seq=3 len=100 total_received=4
Received seq=4 len=100 total_received=5 Received seq=5 len=100 total_received=6
Received seq=6 len=100 total_received=7 Received seq=7 len=100 total_received=8
Received seq=8 len=100 total_received=9 Received seq=9 len=100 total_received=10
Received seq=10 len=100 total_received=11 Received seq=11 len=100 total_received=12
Received seq=12 len=100 total_received=13 Received seq=13 len=100 total_received=14
Received seq=14 len=100 total_received=15 Received seq=15 len=100 total_received=16
Received seq=16 len=100 total_received=17 Received seq=17 len=100 total_received=18
Received seq=18 len=100 total_received=19 Received seq=19 len=100 total_received=20
Received seq=20 len=100 total_received=21 Received seq=21 len=100 total_received=22

Received seq=22 len=100 total_received=23 Received seq=23 len=100 total_received=24
Received seq=24 len=100 total_received=25 Received seq=25 len=100 total_received=26
Received seq=26 len=100 total_received=27 Received seq=27 len=100 total_received=28
Received seq=28 len=100 total_received=29 Received seq=29 len=100 total_received=30
Received seq=30 len=100 total_received=31 Received seq=31 len=100 total_received=32
Received seq=32 len=100 total_received=33 Received seq=33 len=100 total_received=34
Received seq=34 len=100 total_received=35 Received seq=35 len=100 total_received=36
Received seq=36 len=100 total_received=37 Received seq=37 len=100 total_received=38
Received seq=38 len=100 total_received=39 Received seq=39 len=100 total_received=40
Received seq=40 len=100 total_received=41 Received seq=41 len=100 total_received=42
Received seq=42 len=100 total_received=43 Received seq=43 len=100 total_received=44
Received seq=44 len=100 total_received=45 Received seq=45 len=100 total_received=46
Received seq=46 len=100 total_received=47 Received seq=47 len=100 total_received=48
Received seq=48 len=100 total_received=49 Received seq=49 len=100 total_received=50
Received seq=50 len=100 total_received=51 Received seq=51 len=100 total_received=52
Received seq=52 len=100 total_received=53 Received seq=53 len=100 total_received=54
Received seq=54 len=100 total_received=55 Received seq=55 len=100 total_received=56
Received seq=56 len=100 total_received=57 Received seq=57 len=100 total_received=58
Received seq=58 len=100 total_received=59 Received seq=59 len=100 total_received=60
Received seq=60 len=100 total_received=61 Received seq=61 len=100 total_received=62
Received seq=62 len=100 total_received=63 Received seq=63 len=100 total_received=64
Received seq=64 len=100 total_received=65 Received seq=65 len=100 total_received=66
Received seq=66 len=100 total_received=67 Received seq=67 len=100 total_received=68
Received seq=68 len=100 total_received=69 Received seq=69 len=100 total_received=70
Received seq=70 len=100 total_received=71 Received seq=71 len=100 total_received=72
Received seq=72 len=100 total_received=73 Received seq=73 len=100 total_received=74
Received seq=74 len=100 total_received=75 Received seq=75 len=100 total_received=76
Received seq=76 len=100 total_received=77 Received seq=77 len=100 total_received=78
Received seq=78 len=100 total_received=79 Received seq=79 len=100 total_received=80
Received seq=80 len=100 total_received=81 Received seq=81 len=100 total_received=82
Received seq=82 len=100 total_received=83 Received seq=83 len=100 total_received=84
Received seq=84 len=100 total_received=85 Received seq=85 len=100 total_received=86
Received seq=86 len=100 total_received=87 Received seq=87 len=100 total_received=88
Received seq=88 len=100 total_received=89 Received seq=89 len=100 total_received=90
Received seq=90 len=100 total_received=91 Received seq=91 len=100 total_received=92
Received seq=92 len=100 total_received=93 Received seq=93 len=100 total_received=94
Received seq=94 len=100 total_received=95 Received seq=95 len=100 total_received=96
Received seq=96 len=100 total_received=97 Received seq=97 len=100 total_received=98

Received seq=98 len=100 total_received=99 Received seq=99 len=100 total_received=100 Server: reconstructed file 'received.bin' Server SHA256: 50e8ea46e83be16ff02941cd87c56c6d52fbcd381cb074955e0938f043e3b992

4. Wnioski

1. Mimo użycia nieniezawodnego protokołu UDP udało się stworzyć w pełni niezawodny mechanizm transmisji typu Stop-and-Wait.
2. Implementacja handshake i ACK pozwoliła na poprawne wysłanie i odtworzenie pliku w 100% przypadków.
3. Zastosowanie `select()` umożliwiło obsługę timeoutów bez blokowania.

Zaprojektowany protokół działa poprawnie, a wyniki testów potwierdzają jego niezawodność.