PSI Sprawozdanie z zadania 1.2 v1.0

Sofiya Yedzeika
Matvii Ivashchenko
Katsyaryna Anikevich

04 grudnia 2025

Treść zadania

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).

Opis rozwiązania

W zadaniu stworzono prosty własny protokół niezawodnej transmisji działający na UDP. Klient dzieli plik na małe fragmenty i wysyła je do serwera, a serwer odbiera je w dowolnej kolejności i potwierdza każdy odebrany fragment. Brak potwierdzenia w ustalonym czasie powoduje retransmisję, dzięki czemu system radzi sobie z utratą pakietów.

Serwer buforuje otrzymane części, a po skompletowaniu wszystkich łączy je w jeden plik oraz oblicza jego hash SHA-256. Hash klienta i serwera jest następnie porównywany, co potwierdza poprawność transmisji. Całość odwzorowuje uproszczony mechanizm niezawodności podobny do TCP, ale zrealizowany ręcznie na poziomie UDP, zgodnie z wymaganiami zadania.

Klient

Klient generuje plik 10000 B (losowe dane), oblicza jego hash SHA-256 i rozpoczyna transmisję:

1. Wysyła pakiet CONTROL wielokrotnie, aż otrzyma poprawne ACK.
2. Dzieli dane na bloki po 100 B i wysyła pakiety DATA.
3. Każdy pakiet DATA jest wysyłany tak długo, aż klient otrzyma ACK o tym samym numerze sekwencyjnym.
4. Po wysłaniu wszystkich pakietów klient czeka do 8 sekund na pakiet FIN.
5. Po odebraniu FIN kończy pracę.

Mechanizm retransmisji działa poprzez timeout 0.2 sekundy i ponowne nadanie brakującego pakietu.

Serwer

1. Odbiera pakiet CONTROL, alokuje bufor total_packets, wysyła ACK.
2. Każdy pakiet DATA przetwarza następująco:
   - odczytuje numer sekwencyjny seq;
   - zapisuje fragment danych o długości plen w buforze;

- oznacza seq jako odebrany;
- odsyła ACK zawierający seq.

3. Serwer traktuje brak danych przez 5 sekund jako sygnał zakończenia transmisji.
4. Gdy wszystkie pakiety zostały odebrane:
   - scala je w plik received.bin;
   - oblicza SHA-256 rekonstrukcji;
   - wypisuje hash na konsolę;
   - wysyła FIN do klienta.
5. Serwer stosuje timeout SO_RCVTIMEO=5s w celu wyjścia z pętli oczekiwania, kiedy transmisja dobiegła końca.

Opis konfiguracji testowej:

Do testów wykorzystano docker-compose z dwoma kontenerami:

1. Serwer (udp_server)

Budowany z katalogu ./server na bazie obrazu gcc:12.
W kontenerze instalowane są biblioteki OpenSSL, kompilowany jest program serwera i uruchamiany na porcie UDP 8888.
Kontener ma cap_add: NET_ADMIN, co pozwala symulować utratę pakietów (tc netem).

2. Klient (udp_client)

Budowany z katalogu ./client na bazie python:3.
Po skopiowaniu plików do /client uruchamiany jest skrypt: python client.py udp_server 8888.
Klient również posiada NET_ADMIN.

Oba kontenery działają w tej samej sieci Docker Compose, a komunikacja odbywa się po nazwie hosta udp_server na porcie 8888.

Opis wydruków:

Otrzymane wydruki pokazują wartość SHA-256 obliczoną po stronie klienta oraz wartość obliczoną przez serwer po zrekonstruowaniu pliku. Oba hashe są identyczne, co oznacza, że:

- wszystkie pakiety dotarły poprawnie,
- nie wystąpiły błędy w kolejności ani w treści danych,
- protokół retransmisji zadziałał prawidłowo nawet w obecności strat,
- serwer odtworzył dokładnie ten sam plik, który wysłał klient.

Wynik „RESULT: MATCH” potwierdza pełną poprawność transmisji i zgodność danych.

Wnioski

Zaimplementowany protokół niezawodnej transmisji na UDP działa poprawnie: wszystkie fragmenty pliku zostały odebrane, a skróty SHA-256 po stronie klienta i serwera są identyczne. Oznacza to, że mechanizmy potwierdzeń i retransmisji skutecznie kompensują utratę pakietów, a rekonstrukcja danych przebiega bez błędów. Rozwiązanie spełnia wymagania zadania 1.2 i jest odporne na zakłócenia w środowisku testowym.