PSI Zadanie 1.1

18.11.2025, wersja 1

Paweł Spirydowicz, Hanna Zarzycka, Krzysztof Wojtaszko

Terść Zadnia

Z 1 Komunikacja UDP

Napisz zestaw dwóch programów – klienta i serwera wysyłające datagramy UDP. Proszę napisać jedno zadanie w
konfiguracji klient/server Python/C, a drugie w konfiguracji klient/server C/Python – do wyboru.

Z 1.1

Klient wysyła, a serwer odbiera datagramy oraz odsyła ustaloną odpowiedź. Klient powinien wysyłać kolejne datagramy o przyrastającej wielkości, tj. 2, 4, 8, 16, 32, itd. bajtów. Ustalić eskperymentalnie z dokładnością do jednego bajta jak duży datagram jest obsługiwany. Wyjaśnić. Zmierzyć czas pomiędzy wysłaniem wiadomości a odebraniem
odpowiedzi po stronie klienta i zestawić wyniki na wykresie.

Rozwiązanie Zadania

Program serwera został napisany w pythonie, a klienta w C.
Serwer odsyła liczbę bajtów w otrzymanej wiadomości

Fragment programu udp_client.c, pętla wysyłająca coraz dłuższe wiadomości do serwera:

```
  int i;
  for (i=0; i<atol(argv[3]); i++ ) {
      struct timeval send_time, recieve_time;

      memset( databuf, '*', dgram_size );

      // Send datagram
      if ((send( sock, databuf, dgram_size, 0 )) <0 ) {
          perror("Error while sending\n");
      }
  gettimeofday(&send_time, NULL);
      // Server responds with number of bytes in datagram
      int response;
      int rec = recv(sock, &databuf, MAX_DGRAMSIZE, 0);
  gettimeofday(&recieve_time, NULL);
  databuf[rec] = '\0';
  response =  atoi(databuf);
      if (rec < 0) {
          bailout("Error while receving a response\n");
      }
      else if(response != dgram_size) {
          bailout("Invalid server response\n");
```

```
        }

        printf("Datagram size: %i Response delay: %lf\n", dgram_size,
timediff(send_time, recieve_time));
    fflush(stdout);
    dgram_size = dgram_size * 2;
    }
```

Fragment udp_server.py, nasłuchiwanie i odpowiadanie na komunikaty od klienta:

```
 while True:
    data_address = s.recvfrom( BUFSIZE )
    data = data_address[0]
    address = data_address[1]
    print( "Message from Client:{}".format(data) )
    print( "Client IP Address:{}".format(address) )

    if not data:
      print("Error in datagram?")
      break

    response = str(len(data)).encode()
    s.sendto(response, address )
    print(f'Sending number of bytes recieved ({len(data)})')
```

Opis konfiguracji

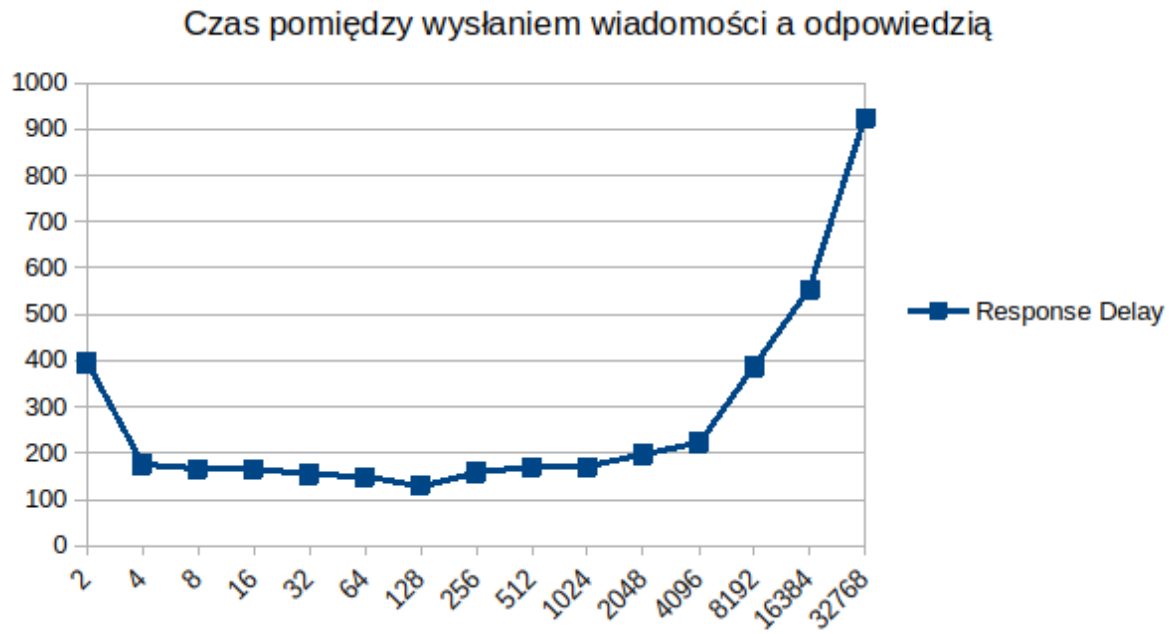
Użyliśmy dwóch kontenerów w Dockerze, z37_server na serwer, z37_client na klienta. Serwer nasłuchiwał na porcie 8000 w sieci z37_network.

Wydruki z testów

Datagram size: 2 Response delay: 396.000000
Datagram size: 4 Response delay: 176.000000
Datagram size: 8 Response delay: 165.000000
Datagram size: 16 Response delay: 166.000000
Datagram size: 32 Response delay: 155.000000
Datagram size: 64 Response delay: 148.000000
Datagram size: 128 Response delay: 130.000000
Datagram size: 256 Response delay: 159.000000
Datagram size: 512 Response delay: 170.000000
Datagram size: 1024 Response delay: 170.000000
Datagram size: 2048 Response delay: 198.000000
Datagram size: 4096 Response delay: 223.000000
Datagram size: 8192 Response delay: 387.000000
Datagram size: 16384 Response delay: 553.000000
Datagram size: 32768 Response delay: 924.000000

Error while sending
: Message too long

Wykres



Wnioski

Pierwsza wiadomość wymagała więcej czasu na dostanie odpowiedzi niż kolejne. Mogło być to spowodowane z czasu wymaganego na inicjalizację buforów, planowania wątku przez system operacyjny i rozgrzewania cache procesora. Wyniki maleją aż do rozmiaru 128 B, potem rosną przez wykładniczy wzrost rozmiaru.

Próba wysłania pakietu o datagramie 64 KiB kończy się błędem. Wiadomość jest zbyt długa. Po zbadaniu różnych długości datagramu maksymalna długość datagramu to 65507 B.
Wytłumaczenie:
64 KiB to maksymalna długość wiadomości łącznie z długością, nagłówkiem i IP
Pole length ma 2 bajty, więc można zakodować liczby od 0 do 65535
Nagłówek UDP zajmuje 8 bajtów
Nagłówek IPv4 zajmuje 20 bajtów
65535 B - 2 B - 8 B - 20 B = 65507 B