PSI laboratorium Z 1.1

Lider - Miłosz Andryszczuk
Aleksander Paliwoda
Maksymilian Zieliński

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.

Z 1.1:

Stworzyliśmy programy serwera w C oraz klienta w pythonie. Oba komponenty uruchamiane są w osobnych kontenerach Dockera, połączonych w jednej sieci wirtualnej. Serwer nasłuchuje na porcie 8888 i odpowiada przez ACK. Klient wysyła kolejne pakiety o rozmiarze zwiększanym dwukrotnie.

Serwer w C tworzy gniazdo UDP i wiąże je z adresem 0.0.0.0:8888, za pomocą funkcji recvfrom() odbiera w pętli datagramy i odsyła odpowiedź ACK. Serwer jest uruchamiany automatycznie w kontenerze zbudowanym z obrazu gcc:4.9.

Fragment kodu odpowiedzialny za obsługę odbioru i odpowiedzi:

```
for (;;) {
    ssize_t n =
        recvfrom(s, buf, sizeof(buf), 0, (struct sockaddr *)&cli, &clen);
    if (n < 0) {
```

```
      perror("recvfrom");
      continue;
    }

    printf("rx=%zd B from %s:%d -> ACK\n", n, inet_ntoa(cli.sin_addr),
           ntohs(cli.sin_port));
    fflush(stdout);

    if (sendto(s, &ack, 1, 0, (struct sockaddr *)&cli, clen) < 0)
      perror("sendto");
  }
```

Klient w Pythonie tworzy gniazdo UDP i wysyła pakiety o kolejno zwiększanym dwukrotnie rozmiarze (2, 4, 8, 16… bajtów), aż napotka błąd transmisji lub przekroczy limit MTU. Dla poprawnie obsłużonego datagramu mierzy czas round trip time. W przypadku braku odpowiedzi w czasie TIMEOUT = 2s test jest uznawany za nieudany. Po pierwszym błędzie rozmiar datagram jest doprecyzowany metodą bisekcji z dokładnością do jednego bajta. Wyniki RTT są zapisywane i wyświetlane na wykresie results.png, który przedstawia zależność czasu RTT od długości danych.

Fragment kodu odpowiedzialny za pomiar czasu i obsługę błędów:

```
data = b"A" * size
    t0 = time.perf_counter()

    try:
        s.sendto(data, (HOST, PORT))
        s.recvfrom(1)

        rtt_ms = (time.perf_counter() - t0) * 1000.0
        s.close()

        return True, rtt_ms, None
```

Uwagi dotyczące napotkanych błędów:

-Sztywny limit na wielkości przesyłanych danych
-Nie wyłączenie fragmentacji

Opis konfiguracji testowej:

Compose:

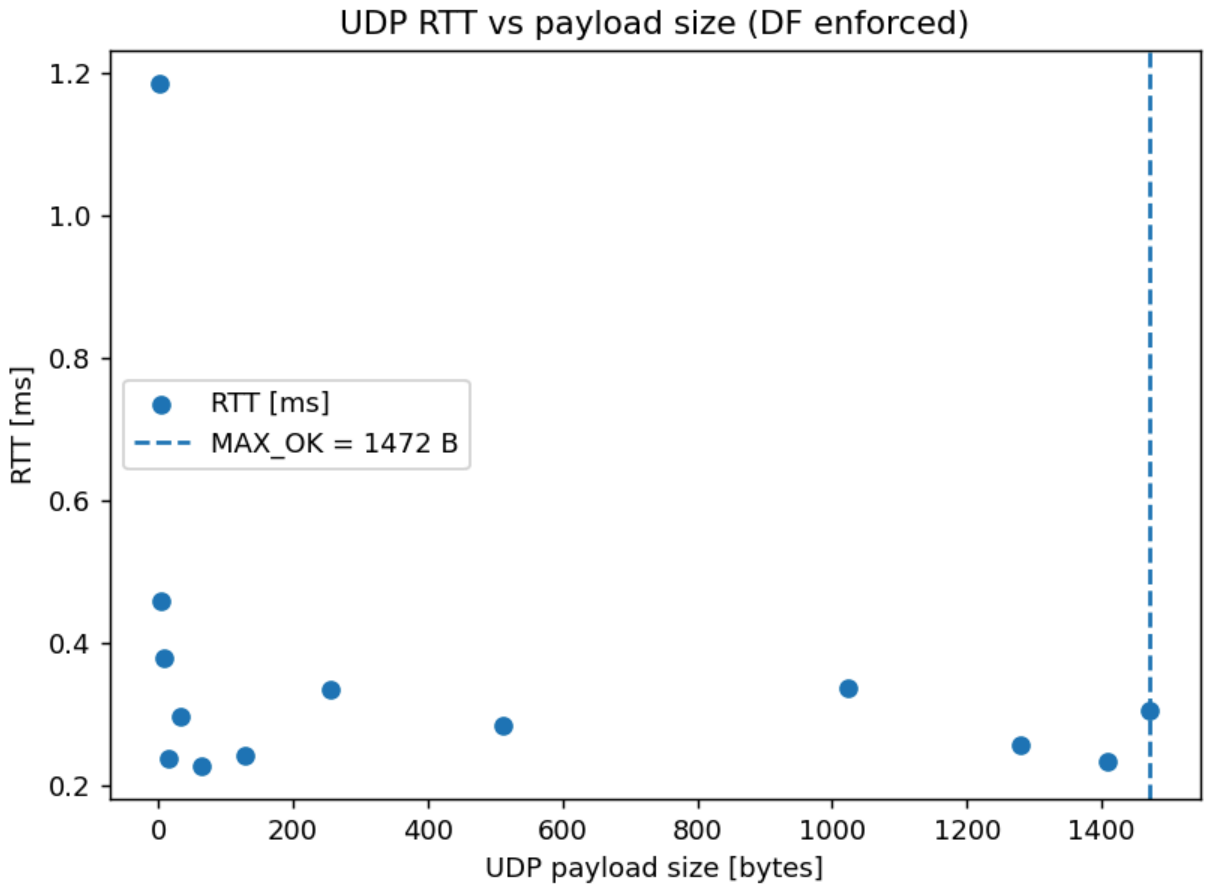
- Z34_udp_server - kontener serwera, obraz z34_server_image, port 8888/UDP
- Z34_udp_client - kontener klienta, obraz z34_client_image
- Z34_network - sieć mostkowa Docker

Adresacja sieci została nadana automatycznie przez Dockera (pula 172.18.0.0/16). Komunikacja zachodziła wewnątrz tej sieci.

Opis testowania i wyniki:

Dla każdej kolejnej długości datagramu klient rejestruje status transmisji (OK/FAIL) oraz czas RTT.
Po wystąpieniu pierwszego błędu (EMSGSIZE albo timeout) klient przechodzi w tryb binarnego wyszukiwania granicy rozmiaru.

Uzyskane wyniki:
-Maksymalny rozmiar przesyłanego datagramu UDP wynosi 1472 bajtów [1500 (standardowy MTU ethernet) -20 (nagłówek IPv4), -8 (nagłówek UDP)]
-Wysyłanie większych pakietów wymagałoby fragmentacji IP, którą wyłączyliśmy
-RTT pozostał stabilny i niski



Wnioski z pomiarów:

Mimo iż teoretyczny limit protokołu UDP wynosi (65507 B), w praktyce maksymalny rozmiar ładunku UDP w typowej sieci Ethernet i środowisku Docker wynosi 1472B.