Sprawozdanie z Laboratorium PSI
Zadanie 1.1 – Komunikacja UDP

Zespół nr 39
Skład zespołu:
Łukasz Szydlik (lider)
Jan Szwagierczak
Dominik Śledziewski

18 listopada 2025

1 Treść zadania

Celem zadania było stworzenie zestawu dwóch programów (klienta i serwera) komunikujących się za pomocą protokołu UDP. Jeden z komponentów został napisany w języku **C**, a drugi w języku **Python**. Aplikacje zostały uruchomione w izolowanym środowisku Docker na serwerze laboratoryjnym *bigubu*.

Główne wymagania funkcjonalne obejmowały:

- Wysłanie przez klienta datagramów o przyrastającej wielkości $(2, 4, 8, \ldots)$.
- Pomiar czasu RTT od momentu wysłania pakietu do odebrania odpowiedzi.
- Eksperymentalne wyznaczenie maksymalnej obsługiwanej wielkości datagramu UDP z dokładnością do 1 bajta.

2 Opis rozwiązania i implementacja

Rozwiązanie oparto na architekturze klient-serwer uruchamianej w dwóch osobnych kontenerach Docker, połączonych dedykowaną siecią typu *bridge* o nazwie `z39_network`.

2.1 Serwer (Język C)

Serwer został zaimplementowany przy użyciu gniazd BSD (`sys/socket.h`). Program działa w nieskończonej pętli, nasłuchując na porcie 8000. Po odebraniu datagramu funkcją `recvfrom`, serwer odsyła statyczne potwierdzenie ("ACK_OK") do nadawcy.

Fragment kodu serwera (pętla główna):

```
while (1) {
    memset(&cliaddr, 0, sizeof(cliaddr));
    len = sizeof(cliaddr);

    int n = recvfrom(sockfd, (char *)buffer, MAX_BUFFER_SIZE,
                0, (struct sockaddr *) &cliaddr, &len);
    // ... obsluga bledow ...

```

```
    // Odeslanie potwierdzenia
    sendto(sockfd, (const char *)ACK_MESSAGE, strlen(ACK_MESSAGE),
           0, (struct sockaddr *) &cliaddr, len);
}
```

2.2 Klient (Język Python)

Klient realizuje logikę testową w dwóch fazach. W pierwszej fazie wysyła pakiety o rozmiarach będących potęgami dwójki i mierzy czas odpowiedzi. W drugiej fazie, po wykryciu błędu rozmiaru pakietu, uruchamia algorytm przeszukiwania binarnego (*binary search*), aby precyzyjnie ustalić limit MTU dla payloadu UDP.

Fragment kodu klienta (obsługa błędów i wykrywanie limitu):

```
try:
    client_socket.sendto(payload, (SERVER_HOST, SERVER_PORT))
    data, server_address = client_socket.recvfrom(64)
except socket.error as e:
    print(f"Size: {len(payload)} B, Network Error: {e}")
    # Przejscie do procedury binary search...
```

3 Konfiguracja środowiska testowego

Testy przeprowadzono na serwerze `bigubu.ii.pw.edu.pl` w środowisku Docker.

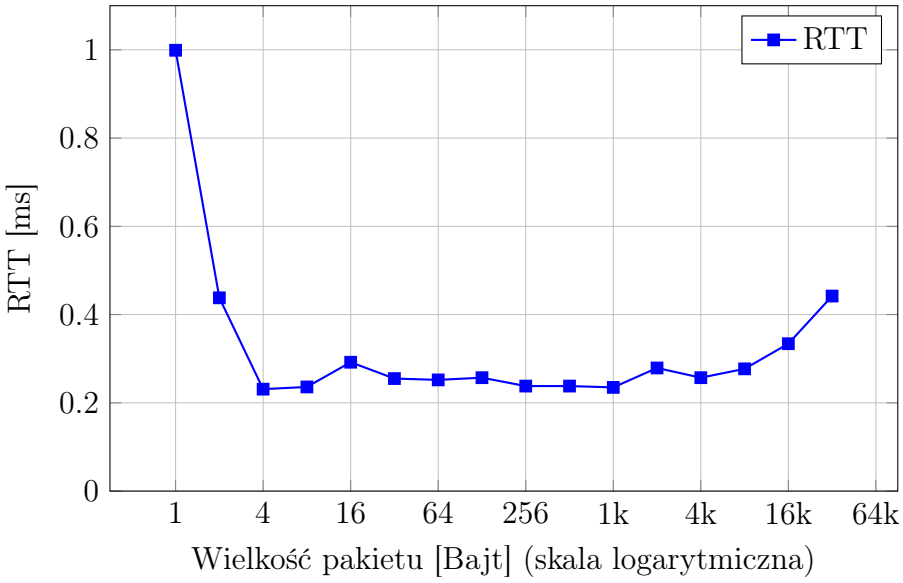
- **Sieć:** `z39_network` (driver: bridge).
- **Port nasłuchu:** 8000.
- **Nazwy kontenerów:** `z39_server_app1`, `z39_client_app1`.

4 Wyniki eksperymentów

W tabeli nr 1 przedstawiono wyniki pomiarów czasu RTT dla poszczególnych wielkości pakietów. Dane pochodzą z logów aplikacji klienta.

Wielkość pakietu [B]	RTT [ms]
1	0.999
2	0.438
4	0.231
8	0.236
16	0.292
32	0.255
64	0.252
128	0.257
256	0.238
512	0.238
1024	0.235
2048	0.279
4096	0.257
8192	0.277
16384	0.334
32768	0.442

Tabela 1: Zestawienie czasów RTT w zależności od wielkości datagramu



Rysunek 1: Wykres zależności czasu RTT od wielkości datagramu UDP

Podczas testów granicznych aplikacja ustaliła, że maksymalny rozmiar danych, jaki udało się poprawnie przesłać, wynosi **65507 bajtów**. Próba wysłania pakietu o rozmiarze 65508 bajtów i większych kończyła się błędem systemowym `[Errno 90] Message too long`. Fragment logu z działania procedury wyszukiwania binarnego przedstawiono poniżej:

```
--- Starting binary search for maximum payload size ---
Payload size: 49152 B, Success. Response: b'ACK_OK'
Payload size: 57344 B, Success. Response: b'ACK_OK'
Payload size: 61440 B, Success. Response: b'ACK_OK'
Payload size: 63488 B, Success. Response: b'ACK_OK'
Payload size: 64512 B, Success. Response: b'ACK_OK'
Payload size: 65024 B, Success. Response: b'ACK_OK'
Payload size: 65280 B, Success. Response: b'ACK_OK'
Payload size: 65408 B, Success. Response: b'ACK_OK'
```

```
Payload size: 65472 B, Success. Response: b'ACK_OK'
Payload size: 65504 B, Success. Response: b'ACK_OK'
Payload size: 65520 B, Network Error: [Errno 90] Message too long
Payload size: 65512 B, Network Error: [Errno 90] Message too long
Payload size: 65508 B, Network Error: [Errno 90] Message too long
Payload size: 65506 B, Success. Response: b'ACK_OK'
Payload size: 65507 B, Success. Response: b'ACK_OK'
Payload max size : 65507 B
Maximum datagram size : 65535 B (including IP and UDP headers)
```

5 Analiza problemów i wnioski

5.1 Maksymalny rozmiar datagramu

Otrzymany wynik eksperymentalny (65507 B) jest zgodny z teoretycznymi ograniczeniami protokołu IPv4. Maksymalny rozmiar pakietu IP (pole *Total Length*) wynosi $2^{16} - 1 = 65535$ bajtów. Nagłówek IP zajmuje minimalnie 20 bajtów, a nagłówek UDP 8 bajtów.

$$MaxPayload = 65535 - 20_{IP} - 8_{UDP} = 65507 bajtow$$

5.2 Analiza czasów RTT

Zaobserwowano zjawisko tzw. "zimnego startu". Pierwszy pakiet (1 bajt) miał czas RTT równy 0.999 ms, co jest wartością znacznie wyższą niż dla kolejnych pakietów (0.23 ms). Było to spowodowane koniecznością rozwiązania adresu IP na adres fizyczny (protokół ARP) oraz uzupełnieniem tablic routingu przy pierwszej transmisji.

Wraz ze wzrostem wielkości pakietu powyżej 1500 bajtów (domyślne MTU dla sieci Docker bridge), zauważalny jest stopniowy wzrost opóźnień (od 0.235 ms dla 1 kB do 0.442 ms dla 32 kB). Wynika to z konieczności fragmentacji pakietów IP na warstwie sieciowej oraz narzutu związanego z kopiowaniem większych bloków pamięci.