Data sporządzenia - 18.11

PSI - KOMUNIKACJA UDP

Katarzyna Kaniczka, Jan Mizera, Andrii-Stepan Pryimak

Treść zadania

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.

Opis rozwiązania problemu

Server:

Implementacja w C w kontenerze Dockera. Otwiera gniazdo UDP na porcie 5550 i nasłuchuje datagramy. Dla każdego otrzymanego datagramu wyświetla adres klienta i rozmiar datagramu oraz wysyła odpowiedź: "Received datagram - size: N bytes".

Klient:

Implementacja w Python, w kontenerze Dockera. Wysyła datagramy o rozmiarach 2^i dla i = 1..16. Po wysłaniu datagramu mierzy czas RTT. W przypadku napotkania błędu wykonuje wyszukiwanie binarne w celu określenia maksymalnego rozmiaru obsługiwanego przez serwer. Wyniki RTT zapisuje w słowniku i rysuje wykres z logarytmiczną skalą X.

Konfiguracja testowa

zad1-1 > client > $ run.sh

```
#!/bin/bash

docker rm -f z32-client-python
docker build -t z32-client-docker .
mkdir -p output

docker run -it --network z32_network -v "$(pwd)/output:/output" --name z32-client-python z32-client-docker $1
```

zad1-1 > server > $ run.sh

```
#!/bin/bash

docker rm -f z32-server-c
docker build -t z32-server-docker .
docker run -it --network-alias z32-server-c --network z32_network --name z32-server-c z32-server-docker $1
```

```
HOST = "z32-server-c"
PORT = 5550
BUFSIZE = 1024
```

```
#define HOST "0.0.0.0"
#define PORT 5550
#define BUFSIZE 65508
```

Serwer: z32-server-c (Docker) **Klient:** z32-client-python (Docker)

Sieć: Docker network z32_network **Port:** 5550

Bufor klienta: 1024 bajty (dla odpowiedzi)

MTU sieci: 1500 bajtów

Docker volume: mapowanie /output do hosta w celu zapisania wykresu RTT

Testowanie i wyniki

Proces uruchomienia jest opisany w pliku Readme.md. Po uruchomieniu program zrobi test dla konfiguracji testowej

Przykładowe wydruki z konsoli:

Client:

```
--------------------------------------------------
Sending datagram to server - size: 16384 bytes
Received response from server: Received datagram - size: 16384 bytes
--------------------------------------------------
Sending datagram to server - size: 32768 bytes
Received response from server: Received datagram - size: 32768 bytes
--------------------------------------------------
Sending datagram to server - size: 65536 bytes
Error: [Errno 90] Message too long
```

Wysyłamy wiadomości o rozmiarze 2^i do momentu otrzymania błędu ‘Message too large’ (errno 90)

```
Sending datagram to server - size: 65504 bytes
Received response from server: Received datagram - size: 65504 bytes
--------------------------------------------------
Sending datagram to server - size: 65520 bytes
Sending datagram to server - size: 65512 bytes
Sending datagram to server - size: 65508 bytes
Sending datagram to server - size: 65506 bytes
Received response from server: Received datagram - size: 65506 bytes
--------------------------------------------------
Sending datagram to server - size: 65507 bytes
Received response from server: Received datagram - size: 65507 bytes
--------------------------------------------------

Disconnected

Found max size: 65507
```

Wysyła wiadomości o rozmiarach pomiędzy 2^(i–1) a 2^i, zawężając zakres metodą wyszukiwania binarnego, aż do znalezienia maksymalnego rozmiaru pakietu UDP.

```
Size: 1 bytes, Round-trip time: 0.001843 seconds
Size: 2 bytes, Round-trip time: 0.000816 seconds
Size: 4 bytes, Round-trip time: 0.000663 seconds
Size: 8 bytes, Round-trip time: 0.000559 seconds
Size: 16 bytes, Round-trip time: 0.000509 seconds
Size: 32 bytes, Round-trip time: 0.000515 seconds
Size: 64 bytes, Round-trip time: 0.000468 seconds
```

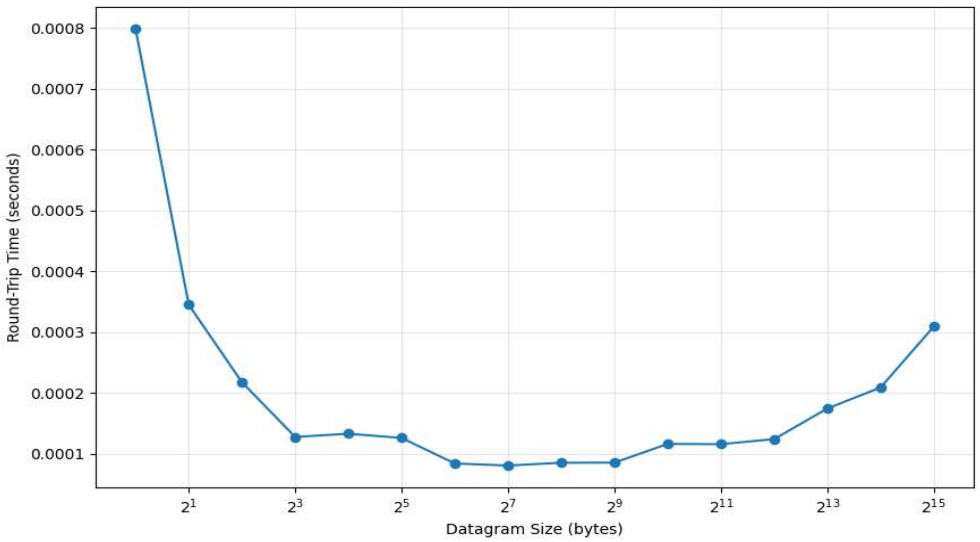
Wypisuje RTT oraz generuje wykres na podstawie tych danych

Server:

```
--------------------------------------------------
Received datagram from 172.18.0.3:40503, size: 65408 bytes
Sending response to the client
--------------------------------------------------
Received datagram from 172.18.0.3:40503, size: 65472 bytes
Sending response to the client
--------------------------------------------------
Received datagram from 172.18.0.3:40503, size: 65504 bytes
Sending response to the client
--------------------------------------------------
Received datagram from 172.18.0.3:40503, size: 65506 bytes
Sending response to the client
--------------------------------------------------
Received datagram from 172.18.0.3:40503, size: 65507 bytes
Sending response to the client
--------------------------------------------------
```

Wpisy otrzymanych wiadomości (adres, rozmiar)

Tabela wyników RTT + wykres



Widać minimalny wzrost RTT dla małych datagramów i lekkie zwiększenie dla dużych datagramów

Rozmiar	RTT	Liczba fragmentów	Przepustowość	Rozmiar Headera/Rozmiar
1		1		96.5%
2		1		93.3%
4		1		87.5%
8		1		77.7%
16		1		63.6%
32		1		46.6%
64		1		30.4%
128		1		17.9%
256		1		9.85%
512		1		5.18%
1024		1		2.66%
2048		2		1.34%
4096		3		0.67%
8192		6		0.34%
16384		11		0.17%
32768		22		0.08%
65536		44		0.04%

Problemy napotkane i rozwiązania

W trakcie realizacji zadania nie wystąpiły istotne problemy techniczne ani trudności implementacyjne. Zarówno kod serwera w języku C, jak i kod klienta w Pythonie działały poprawnie

Wnioski

Ustalono eksperymentalnie maksymalny rozmiar pakietu UDP - 65507 bajtów. Co potwierdza oczekiwania maksymalny rozmiar pakietu IP = 2^16 = 65535

IP header + UDP header + dane UDP ≤ 65535 bajtów

Max(dane UDP) = 65535 – 20 -8 = 65507

Fragmentacja pakietów w warstwie IP nie powodowała błędów. Dla pakietów większych od MTU = 1500 bajtów nie obserwowaliśmy zgubienia fragmentów. RTT nie wzrastało drastycznie. Serwer zawsze poprawnie odbierał zrekonstruowane datagramy

Wykres RTT pokazał bardzo niewielki wzrost czasu dla dużych datagramów

Serwer jest stabilny i nie zawiesza się w losowych momentach.