Raport o projekcie #1 z sieci komputerowych
Natalia Junkiert 448 267

W następującym raporcie, opiszę testy przeprowadzone na klientach TCP, UDP, i UDP z retransmisją oraz serwerach TCP i UDP. Opisane będą procedury użyte do testowania, ustawienia programów oraz wyniki przeprowadzonych testów. Każdy test będzie patrzył na to jak jeden czynnik wpływa na pewien aspekt wydajności programów aby móc wyciągnąć wiarygodne i użyteczne konkluzje. Wszystkie testy były przeprowadzone na maszynach wirtualnych przez interfejs eth1.

1. **Jak przepustowość łącza wpływa na prędkość przesyłu danych?**

Procedura testowania przebiega następująco. Po stronie serwera wprowadzane są komendy:

```
sudo tc qdisc del dev eth1 root
sudo tc qdisc change dev eth1 root netem loss 0%
sudo tc qdisc add dev eth1 root tbf rate 10kbps burst 100kbit latency 50ms
```

które najpierw usuwa główne dyscyplinowanie kolejek (qdisc) na eth1, ustawia żeby nie było gubienia pakietów, po czym ustawia przepustowość (tbf rate) na 100, 1000 i 10000 kbps w kolejnych testach. Następnie, serwer jest uruchamiany i odpowiedni klienci wysyłają plik wielkości 10 megabajtów (10141756 bajtów dokładnie). Za pomocą funkcji gettimeofday() z pliku nagłówkowego <sys/time.h> mierzony jest czas potrzebny na przesłanie danych. Proces ten jest powtarzany 3 razy dla każdego protokołu i obliczana jest prędkość przesyłu danych po czym wyliczamy z tych trzech wartości średnią prędkość przesyłu danych.
Klient wysyłał paczki po 1000 bajtów, MAX_WAIT = 20 oraz MAX_RETRANSMITS = 5.

Protokół	**TCP**			**UDP**			**UDPR**		
Przepustowość (kbps)	100	1000	10000	100	1000	10000	100	1000	10000
Prędkość przesyłu danych (Mbit/s)	18.728	21.87	25.45	602.067	1575.23	1787.67	11.579	14.638	16.647

Wyniki testów pokazują że wraz ze zwiększającą się przepustowością, prędkość przesyłu danych również wzrasta. Widać, że klient UDP najszybciej przesyłał dane do serwera, jednak to wynika z tego że taki klient wysyła wszystkie dane bez czekania na potwierdzenie od serwera ani połączenia z serwerem. To jednak powoduje że nie jest gwarantowana kolejność danych odbieranych po stronie serwera w przeciwieństwie do TCP i UDPR które pilnują kolejności przesyłanych danych. Widocznie jest również to, że TCP jest (trochę) szybsze niż UDPR, co może być spowodowane tym, że dla TCP socket z “typem” sock_stream gwarantuje nam że kolejność danych jest taka jak klient przesłał, a serwer UDPR, musi zweryfikować poprawną kolejność danych przed potwierdzeniem; również, UDPR musi odfiltrować dane od innych klientów i powtórki poprzednich pakietów od aktualnego klienta co resetuje oczekiwanie na dane, co jest wolniejsze od TCP który wie że komunikuje się z jednym klientem i tylko weryfikuje kolejność i poprawność danych które przychodzą.

2. **Jak procent zagubionych pakietów wpływa na program serwerów i klientów oraz na ilość przesyłanych danych?**

Aby przetestować zachowanie programów gdy zmienia się procent zagubionych pakietów, wpisuję komendę

```
sudo tc qdisc change dev eth1 root netem loss 0%
```

z procentami 10%, 20%, 30% (dla 0%, widzimy dane powyżej również). Przepustowość sieci jest ustawiona na 1000kbps a opóźnienie na 50ms. Dla każdego ustawienia procentu zagubionych pakietów i każdego protokołu, wysyłam plik wielkości 0.1MB (100793 bajtów), z tymi samymi ustawieniami co wyżej. Każdy test powtarzam trzy razy.
(Nie jest to zaskakujące że) Dla klienta i serwera TCP, dla wszystkich procentów pakietów zagubionych, wszystkie dane były przesłane w całości i klient otrzymał pakiet RCVD oraz prędkość przesyłu danych był 20.36kbps (czyli

niewiele odbiegający od prędkości przy braku zagubionych pakietów). Klientowi UDP udało się wysłać wszystkie dane, ale zakończył się błędem dla każdego procentu testowanego, ponieważ dane nie przyszły do serwera w odpowiedniej kolejności (ponieważ protokół UDP nie gwarantuje, że dane przyjdą w poprawnej kolejności).
Dla UDPR, klientowi udało się przesłać dane przy 10% z prędkością 0.004384Mbit/s, przy 20% z prędkością 0.00171045Mbit/s, a przy 30% z prędkością 0.0009376Mbit/s.
Jak widać, procent zagubionych paczek najbardziej wpływa na wydajność klienta UDPR który aktywnie komunikuje się z serwerem, więc aby klient UDPR mógł wysłać kolejną paczkę danych (która może się zgubić), musi odebrać odpowiednią paczkę ACC która również może się zagubić. Proces retransmisji bardzo spowalnia klienta UDPR i serwera który go obsługuje, bardziej niż niektóre inne czynniki które widać w innych testach ponieważ z podwyższeniem procentu zagubionych paczek o 10%, klient około 2 razy wolniej wysyła dane, ponieważ proces retransmisji jest bardzo drogi.

3. **Jak opóźnienie wpływa na prędkość przesyłu danych?**

Testujemy to za pomocą komendy

sudo tc qdisc add dev eth1 root netem delay 100ms

z następującymi wartościami dla opóźnienia: 100ms, 1000ms i 5000ms. Ustawienia programu są jak wyżej.
Dla TCP, prędkość przesyłania danych była kolejno 3.877, 0.7837 i 0.16015 Mbit/s dla tych wartości. Za każdym razem jak zwiększał się delay, (około) 5-ciokrotnie wolniej działał klient TCP. Dla UDP, było to 55.61, 54.307 i 54.013 Mbit/s, czyli niezauważalna zmiana. Dla UDPR było to kolejno 0.07093, 0.007712 (~10 razy wolniej) oraz 0.001565 Mbit/s (~5 razy wolniej). To pokazuje dość nieciekawą konkluzję że im większe opóźnienia tym wolniej dane się przesyłają, ponieważ każda paczka jest opóźniona i trzeba na nią czekać, co spowalnia prędkość przesyłu danych.

4. **Jak rozmiar przesyłanej paczki wpływa na prędkość przesyłu danych?**

Dotychczas, wszystko było testowane ustawiając maksymalny rozmiar paczki na 1000 bajtów, tak aby się mieściło w MTU dla eth1. Jak mamy większe pakiety, to dla UDP, są one dzielone na mniejsze pakiety i potem składane po stronie serwera. To jednak zwiększa ryzyko na zagubienie się danej paczki, ponieważ jeśli co najmniej jedna część paczki się zgubiła to cały pakiet nie dojdzie. W tym teście, zmieniam MAX_MESSAGE_SIZE na kolejno 10000, 32000 oraz 64000, po czym sprawdzam jak wpływa to na prędkość i dostarczenie danych. Procent zagubionych pakietów ustawiony jest na 10%. Tych komend użyłam:

sudo tc qdisc change dev eth1 root netem loss 10%

Rozmiar przesyłanych paczek zmieniłam bezpośrednio w odpowiednim pliku klienta.
Dla rozmiaru pakietu równym 1000, TCP osiągnęło prędkość przesyłu danych 15.396Mbit/s, UDPR 0.004365Mbit/s, a UDP 170.1Mbit/s ale udało się przesłać w poprawnej kolejności jedynie 294 pakietów, czyli 294000 bajtów danych. Dla rozmiaru paczki 10000, TCP wysyłało z prędkością 22.128 Mbit/s, UDPR z prędkością 0.013029 Mbit/s, a UDP wysłało po kolei 43 pakiety czyli 430000 bajtów danych z prędkością 244.42Mbit/s. Dla 32000, TCP osiągnęło prędkość 20.905Mbit/s, UDPR 1.553Mbit/s a UDP udało się przesłać średnio 11 pakietów po kolei przed zagubieniem się paczki albo niepoprawnej kolejności, czyli udało się przesłać 352000 bajtów danych. Dla 64000, TCP miało prędkość 75.66Mbit/s, UDPR 0.3797Mbit/s a UDP wysyłało (mniej więcej) tak samo szybko co w innych testach ale udało się przesłać 256000 bajtów przed zagubieniem się paczki.

5. **Co się dzieje jak mamy poprawny serwer obsługujący niepoprawnych klientów? Co się dzieje jak poprawny klient komunikuje się z niepoprawnym serwerem?**

Następujące testy się skupiają na error-handlingu i jak zachowują się programy jak są niepoprawne scenariusze.
Dla TCP, jeżeli klient zakończy się przed wysłaniem wszystkich danych, dostajemy po stronie serwera ERROR: connection closed, tak samo w drugą stronę.
Aby zasymulować niepoprawną kolejność wysyłania pakietów, zrobiłam aby z prawdopodobieństwem 50%, wysyłał się dany pakiet po stronie klienta, i po obu stronach wypisywane na stdout były również informacje o tym które pakiety były wysłane oraz które odebrane po stronie serwera. We wszystkich przypadkach, był moment gdzie klient faktycznie nie wysłał danej paczki i odebrał wtedy RJT od serwera i zakończył działanie. To jest prawidłowe postępowanie. Te testy powtarzałam co najmniej trzykrotnie dla każdego protokołu, a dla UDPR, raz

zdarzyło się że nie wysłał klient danej paczki MAX_RETRANSMITS razy, więc zakończył się z ERROR: retransmits exceeded, a serwer z tego powodu przestał go obsługiwać.
Jeżeli klient się połączył ale nie wysłał żadnych danych, serwer TCP zamykał połączenie po odpowiednim czasie, a UDP wypisywał na stderr ERROR: timeout albo że retransmisja była zrobiona więcej niż maksymalna liczba dozwolonych retransmisji.
Do serwera TCP, nadchodzące połączenia czekają aż serwer zakończy obsługiwać aktualnego klienta a serwer UDP poprawnie wysyła CONRJT do nadchodzących, obcych klientów. UDP poprawnie utrzymywał komunikację z klientem UDPR pomimo nadchodzących conn pakietów od innych klientów.
Jeżeli serwer UDP nie będzie wysyłał potwierdzeń, klient udpr w pewnym momencie zakończy się albo errorem że pakiet został odrzucony ponieważ serwer znowu czeka na conn pakiety, albo że za dużo retransmisji było wykonanych, i zakończy się błędem.
Ostatni test który warty jest wspomnienia jest to że jak mamy serwer UDP, obsługujący klienta, i inni klienci bez otrzymania CONACC, wysyłają dane (bez czekania na odpowiedź), to serwer wysyła RJT odpowiednim klientom a obsługiwany dalej może komunikować się z serwerem. Używałam tutaj 5 klientów na maszynach wirtualnych (na więcej miejsca nie mam niestety), więc działa to dla jednego poprawnego klienta i czterech którzy przeszkadzają w komunikacji. Odpowiedni klient zakończył się sukcesem, inne dostały pakiet RJT. To jest poprawne zachowanie programów. Przypuszczam że jak będzie dostatecznie dużo klientów wysyłających dane, może to rezultować tym że poprawny klient nie otrzyma od serwera ACC w MAX_WAIT * MAX_RETRASMITS czasu ponieważ serwer filtruje przez niepoprawne pakiety.

6. **Czy wielkość pliku ma efekt na prędkości przesyłania i wydajności programów?**

TCP oraz UDPR klienci wysyłali pliki wielkości 108,632,485 (a), 65,536,615 (b), i 1,985,958 (c) bajtów. Dla TCP, prędkości w Mbit/s w kolejności (a) 24.435, (b) 18.48 i (c) 21.314 a dla UDPR (a)13.863, (b) 14.903 i (c) 14.75. Różnica w wielkości pliku nie ma zatem znaczenia na prędkość przesyłania danych ponieważ nie ma żadnego tren du. Warto wspomnieć że poprawnie przesyłane są pliki wielkości 3 bajty (czyli ten edge case również został rozpatrzony). Widać że o ile jest miejsce na dany plik na maszynie, można go przesłać bez problemu.