ZAVOD ZA ELEKTRONIKU, MIKROELEKTRONIKU, RAČUNALNE I INTELIGENTNE SUSTAVE
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
SVEUČILIŠTE U ZAGREBU

OPTIMIZACIJA TCP PARAMETARA NA LOKALNIM GIGABITNIM MREŽAMA

Saša Mrvoš

SEMINARSKI RAD IZ PREDMETA *MREŽE RAČUNALA*

Zagreb, 2007.

Sadržaj

1. Uvod

Mjerenjem propusnosti protokola TCP ili UDP između dva računala na lokalnoj gigabitnoj mreži uočava se kako su one znatno niže očekivanih. Razlog za takvo ponašanje leži u činjenici da TCP protokol nije posebno optimiran za lokalne mreže. TCP, kao i svi parametri uz pomoć kojih se može uticati na njegovo ponašanje, su podešeni za rad i na lokalnim mrežama i na Internetu. Ipak, postoje situacije u kojim su nam važne samo performanse između dva direktno spojena računala na lokalnoj mreži. Primjer takvih situacija su grozdovi (klasteri, engl. clusters) računala koji izvršavaju neku distribuiranu aplikaciju za koju je važan brzi protok podataka između računala unutar grozda; sama mreža je, dakle, usko grlo.

Na Linux operacijskom sustavu mnogi su TCP parametri podesivi. Cilj ovog seminara bio je izmjeriti performanse u masovnom (engl. bulk, stream) prijenosu podataka između dva direktno spojena računala na gigabitnoj Ethernet lokalnoj mreži te odrediti (optimirati) za koje TCP parametre će te performanse biti maksimalne.

U masovnom prijenosu podataka propusnost podataka je performansa od interesa. Cilj seminara je također odrediti maksimalnu realnu propusnost, te time i maksimalnu iskoristivost lokalnog Ethernet kanala.

2. TCP parametri

Pod Linux-om se većina TCP parametara može podesiti uz pomoć niza parametara. Budući da su ti parametri unutar jezgre operacijskog sustava pristupa im se putem virtualnog datotečnog sustava (engl. virtual filesystem) koji se nalazi unutar `/proc` Direktorija. Pod virtualnim datotečnim sustavom podrazumijeva datotečni sustav koji ne postoji na tvrdom disku već isključivo u radnoj memoriji računala, a promjene koje se učine na virtualnim datotekama zapravo, u većini slučajeva, mijenjaju varijable u memoriji jezgre.

Ostali parametri se podešavaju odgovarajućim konfiguriranjem Ethernet sučelja naredbama `ifconfig` ili `ip`.

Čitanje i pisanje datoteka unutar `/proc` datotečnog sustava, a time i odgovarajućih parametara, izvodi se kao i čitanje/pisanje datoteke. Pomoću naredbe `cat` ispisujemo sadržaj datoteke, a time i varijable, na primjer:

```
# cat /proc/sys/net/core/rmem_default
65535
#
```

Pomoću naredbe `echo` ispisujemo tekst na standardni izlaz. Naredbama preusmjeravanja toka, > i >>, možemo tekst preusmjeriti u datoteku, a time i upisati vrijednost parametra, na primjer:

```
# echo "4096 16384 131072" > /proc/sys/net/ipv4/tcp_wmem
#
```

2.1. TCP parametri mapirani u /proc datotečni sustav

Većina TCP i IP parametara se nalazi u datotekama unutar `/proc/sys/net/core/` i `/proc/sys/net/ipv4/` direktorija. Naveden je kratak opis parametara značajnih za preformanse.

U tablici 2.1 možemo vidjeti značenje pojedinog parametra iz `/proc/sys/net/core/` direktorija.

Tablica 2.1: TCP parametri iz /`proc/sys/net/core/` direktorija

`rmem_default`	Podrazumjevana veličina prijemnog TCP prozora u oktetima.
`rmem_max`	Maksimalna veličina prijemnog TCP prozora u oktetima.
`wmem_default`	Podrazumjevana veličina predajnog TCP prozora u oktetima.
`wmem_max`	Maksimalna veličina predajnog TCP prozora u oktetima.

U tablici 2.2 možemo vidjeti značenje pojedinog parametra iz `/proc/sys/net/ipv4/` direktorija.

Tablica 2.2: TCP parametri iz /`proc/sys/net/ipv4/` direktorija

`tcp_wmem`	Vektor s 3 cjelobrojne vrijednosti: Min, Default, Max. Vrijednosti reguliraju veličinu predajnih (izlaznih) TCP međuspremnika u oktetima. TCP dinamički prilagođava veličinu pojedinih međuspremnika u ovisnosti o količini slobodne memorije. Min: Minimalna veličina predajnog (izlaznog) TCP međuspremnika rezervirana za pristupnu točku. Toliko memorije pristupna točka dobija samim svojim stvaranjem. Default: Dozvoljena, podrazumjevana veličina predajnog međuspremnika. Max: Maksimalna veličina predajnih međuspremnika, dozvoljena za automatski selektirane međuspremnike pristupnih točaka. Podrazumjevana vrijednost: 4K, 16K, 128K [Bytes]
`tcp_rmem`	Vektor s 3 cjelobrojne vrijednosti: Min, Default, Max. Vrijednosti Min, Default i Max imaju slično značenje kao i kod `tcp_wmem`, samo što se odnose na prijemne (ulazne) memorijske prostore. Ulazni memorijski prostor se dijeli na ulazni TCP međuspremnik, te na ulazni aplikacijski međuspremnik, ovisno o varijablama `tcp_adv_win_scale` i `tcp_app_win`. Podrazumijevana vrijednost: 8K, 87380, 87380*2 [Bytes] Napomena: Vrijednost 87380, zajedno sa podrazumjevanim vrijednostima varijabli `tcp_adv_win_scale` i `tcp_app_win` rezultira ulaznim TCP prozorom veličine 65535 okteta i ulaznim aplikacijskim međuspremnikom veličine 21845.
`tcp_mem`	Vektor sa 3 Cjelobrojne vrijednosti: Low, Pressure, High. Vrijednosti reguliraju količinu memorije koju jezgra operacijskog sustava koristi za TCP. Vrijednosti su mjerene u broju memorijskih stranica (engl. Pages), svaka veličine 4 KB. Low: Jezgra operacijskog sustava ne poduzima nikakve mjere ako je ukupna količina memorije alocirane za TCP ispod ove vrijednosti. Pressure: Ako količina memorije alocirana za TCP pređe ovu vrijednost, jezgra operacijskog sustava ulazi u takozvani “pressure” način rada, u kojem ostaje sve dok količina alocirane memorije ne padne ispod “Low”. U “Pressure” načinu rada TCP pokušava smanjiti “apetite” za memorijom, i smanjiti međuspremnike za sve pristupne točke. High: Najveći dozvoljeni broj memorijskih stranica alociran za TCP. Podrazumjevana vrijednost: računa se prilikom podizanja (engl. boot) sistema na temelju količine slobodne memorije.

`tcp_adv_win_scale`	Cjelobrojna vrijednost. Faktor skaliranja za računanje aplikacijskog dijela TCP prijemnog memorijskog prostora. Aplikacijski dio iznosi $\texttt{receive\_space}/2^{\texttt{tcp\_adv\_win\_scale}}$. Prijemni (engl. receive) memorijski prostor svake TCP pristupne točke dijeljen je na TCP dio i aplikacijski dio. Jezgra operacijskog sustava koristi TCP dio kao TCP prozor, i tu veličinu objavljuje drugoj strani da je spremna prihvatiti. Aplikacijski dio jezgra koristi kao "aplikacijski" međuspremnik. Tamo sprema čiste podatke aplikacijskog sloja koje treba predati aplikaciji. Time se izbjegava zastajkivanje u mrežnoj komunikaciji uzrokovano aplikacijskim kašnjenjima (latencijama), te nepovoljnim raspoređivanjem vremena za procese i dretve (engl. thread scheduling). Podrazumjevana vrijednost: 2. Napomena 1: Veličina prijemnog memorijskog prostora je zadana `tcp_rmem` varijablom. Napomena 2: Podrazumjevana vrijednost uzrokuje da se četvrtina prijemnog memorijskog prostora rezervira za aplikacijski međuspremnik, a tri četvrtine za TCP međuspremnik.
`tcp_app_win`	Cjelobrojna vrijednost. Faktor skaliranja za računanje dijela TCP prozora rezerviranog za aplikacijski međuspremnički dodatak (engl. overhead). Rezervirat će se barem $\texttt{max(MSS, TCP\_Window}/2^{\texttt{tcp\_app\_win}}\texttt{)}$ okteta za aplikacijski međuspremnički dodatak. Veličina TCP prozora rauuna se pomoću `tcp_rmem` i `tcp_adv_win_scale` varijabli. Podrazumjevana vrijednost: 31. Napomena: Podrazumjevana vrijednost uzrokuje da Window/2^{31} bude manje od jedan, te da se MSS koristi kao međuspremnički dodatak.
`tcp_window_scaling`	Logička vrijednost. Omogući/onemogući korištenje skaliranja prozora (engl. window scaling), TCP opcije definirane u RFC1323. Maksimalna veličina prozora je prema temeljnoj TCP specifikaciji ograničena na 65536 okteta. Omogućavanjem opcije "window scaling" se uklanja to ograničenje. Opcijom se zadaje faktor skaliranja koji se koristi za računanje stvarne veličine prozora: $\texttt{(window\_size*2}^{\texttt{window\_scale}}\texttt{)}$.
`tcp_timestamps`	Logička vrijednost. Omogući/onemogući korištenje vremenskih oznaka (engl. timestamps), TCP opcije definirane u RFC1323. Opcija vremenskih oznaka se koristi za preciznije utvrđivanje trenutnog vremena obilaska (engl. Round Trip Measurement).
`tcp_sack`	Logička vrijednost. Omogući/onemogući korištenje selektivnih potvrda (engl. Selective Acknowledgements, SACKs), definiranih u RFC1072. Potvrđuje se svaki segment TCP prozora. U slučaju gubitka podataka ponovo se šaljesamo segment u kojem se pojavila greška. Može povećati propusnost u kanalima u kojima se češće javljaju greške, inače uvodi redundanciju.

tcp_fack	Logička vrijednost. Omogući/onemogući korištenje unaprijednih potvrda (engl. Forward Acknowledgement, FACK). Služi za izbjegavanje zakrčenja i brzu retransmisiju, a radi iznad SACK algoritma. Ne koristi se ako SACK nije omogućen.
tcp_dsack	Logička vrijednost. Omogući/onemogući slanje duplih selektivnih potvrda (engl. Duplicate Selective Acknowledgements, DSACKs).
tcp_ecn	Logička vrijednost. Omogući/onemogući eksplicitno obavještavanje o zakrčenju (engl. Explicit Congestion Notification, ECN) u TCP spojevima. Koristi se kako bi se predajnike obavijestilo da je došlo do zakrčenja na putu u mreži, te da šalju sporijom brzinom.
tcp_low_latency	Logička vrijednost. Da li da TCP slog preferira manje kašnjenje (true) ili veću propusnost (false).
tcp_congestion_control	Varijabla sadrži naziv algoritma koji se koristi za kontrolu zakrčenja. Algoritam mora biti instaliran na sustavu. Neki od algoritama su: reno, bic, cubic, highspeed, scalable, westwood, vegas.

2.2. Konfiguracija Ethernet sučelja

Konfiguracijom Ethernet sučelja možemo izmijeniti maksimalnu veličinu Ethernet paketa te veličinu reda Ethernet paketa. Naredbom:

```
# /sbin/ifconfig $SUCELJE mtu $MTU
#
```

postavljamo maksimalnu veličinu Ethernet okvira (engl. Maximum Transfer Unit). Podrazumjevana vrijednost je 1500 okteta, a za lokalne gigabitne mreže se preporučava vrijednost od 9000 okteta (Jumbo Frames). Navodno, povećani MTU utječe pozitivno na performanse, no na žalost, na mjernim računalima na kojima se radio ovaj seminar nije bilo moguće povećati MTU.

Naredbom:

```
# /sbin/ifconfig $SUCELJE txqueuelen $TX_QUEUE_LEN
```

postavljamo maksimalnu veličinu paketa koji se mogu spremiti u međuspremnik Ethernet sučelja. Veći red znači da više paketa stane u međuspremnik. Prepunjavanje međuspremnika uzrokuje gubitak podataka i pokretanje algoritama za kontrolu zakrčenja.

3. Mjerenje performansi

Mjerenja su vršena na dva računala sa instaliranim Linux operacijskim sustavom, distribucije Fedora Core 6, te gigabitnim Ethernet mrežnim karticama. Računala su preko Ethernet kartica spojena na gigabitni preklopnik.

Testirali smo propusnost TCP-a i UDP-a u masovnom (engl. bulk, stream) načinu rada, te broj transakcija u sekundi u zahtjev-odziv (engl. Request-Response) načinu rada. Testiranja su vršena pomoću programa Netperf.

3.1. Netperf

Netperf je mjerni program razvijen od Hewlett-Packard kompanije, a mjeri različite aspekte mrežnih performansi. Fokusiran je na mjerenja masovnih podatkovnih prijenosa, te zahtjev-odziv prijenosa korištenjem bilo TCP-a bilo UDP-a.

Netperf je izrađen oko klijent-poslužitelj modela. Postoje dvije aplikacije: `netperf` i `netserver`. Poslužiteljska aplikacija, `netserver`, pokreće se kao servis. Klijentskom aplikacijom (`netperf`) podešava se i pokreće mjerenje. Prilikom pokretanja netperf-a prvo se ostvari kontrolna konekcija prema udaljenom računalu, na kojem mora biti pokrenut `netserver`. Njom se šalje informacija o vrsti i konfiguraciji mjerenja, te na kraju rezultati mjerenja. Kontrolna konekcija je TCP konekcija ostvarena pomoću BSD pristupnih točaka.

Kad se kontrolnom vezom prenesu konfiguracijske informacije, otvara se nova veza kojom se vrši ispitivanje. Tip (protokol) te veze, kao i korišteni API, ovise o konfiguraciji ispitivanja. Nakon završetka mjerenja, netperf ispisuje rezultate na standardni izlaz.

Ovdje će biti ukratko iznešene upute o korištenju netperf-a, a u Prilogu A se nalazi bash skripta koja je služila za automatizaciju mjerenja.

Opći oblik poziva netperf-a je:

```
$ ./netperf -H remote_host [global_options] -- [test_options]
```

gdje je `remote_host` adresa udaljenog računala.

Globalne opcije su:

`-l testlen`

Trajanje mjerenja u sekundama.

`-f [G|M|K|g|m|k]`

Prikaz rezultata u G=GB/s, M=MB/s, K=KB/s, g=10^9 bps, m=10^6 bps, k=10^3 bps.

`-i max,min`

Broj ponavljanja mjerenja. Ispitivanje će se ponoviti mimimalno `min`, a maksimalno `max` puta, da se zadovolji točnost rezultata.

`-I int,wid`

Specificira se interval pouzdanosti `int`, i širina intervala pouzdanosti `wid`. Mjerenje će se ponavljati više puta da se statistički zadovolji točnost rezultata.

`-p portnum`

Na kojem portu osluškuje netserver. Podrazumjeva se port 12865.

`-t testname`

Specifikacija ispitivanja (testa) kojeg želimo pokrenuti:

TCP_STREAM – ispitivanje masovnih strujećih prijenosa TCP-om.
UDP_STREAM – ispitivanje masovnih strujećih prijenosa UDP-om.
TCP_RR – ispitivanje zahtjev-odziv prijenosa TCP-om.
UDP_RR – ispitivanje zahtjev-odziv prijenosa UDP-om.

Test opcije (opcije ispitivanja) su:

`-s local_socksize -S remote_socksize`

Podešavanje veličine prijemnih i predajnih međuspremnika pristupnih točaka na lokalnom i udaljenom računalu.

`-m local_send_size -M remote_receive_size`

Specificiranje veličine paketa sa aplikacijske razine koji se šalju/primaju prilikom testiranja masovnih prijenosa.

`-r request_size,response_size`

Specificiranje veličine zahtjeva/odziva (sa aplikacijske razine) koji se šalju/primaju prilikom testiranja zahtjev-odziv prijenosa.

3.1.1. Korištenje netperf-a u mjerenju masovnih prijenosa TCP-om

Tipičan način pozivanja netperfa za ispitivanje masovnih strujećih prijenosa TCP-om jest:

```
$ ./netperf -H remote_host -t TCP_STREAM [global_options] -- -s
local_socksize -S remote_socksize -m local_send_size -M
remote_receive_size
```

Ispis rezultata dobiven "netperf_script" skriptom (Prilog A) koja automatizira mjerenje variranjem parametara izgleda ovako:

```
TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 161.53.65.228
(161.53.65.228) port 0 AF_INET
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    KBytes/sec
-------------------------------------------------------------
  8192   8192 524288    10.00    16336.61
 16384  16384 524288    10.00    40058.64
 32768  32768 524288    10.00    71776.36
131072 131072 524288    10.00    87662.49
262142 262142 524288    10.00    90822.97
```

Prvi, drugi i treći stupac prikazuju veličinu međuspremnika i paketa koji se šalje (zadane opcijama `-s -S -m`), četvrti stupac trajanje testa (opcija `-l`), a peti stupac rezultate testiranja.

3.1.2. Korištenje netperf-a u mjerenju masovnih strujećih prijenosa UDP-om

Tipičan način pozivanja netperfa za mjerenje masovnih prijenosa UDP-om bi bio:

```
$ ./netperf -H remote_host -t UDP_STREAM [global_options] -- -s
local_socksize -S remote_socksize -m local_send_size -M
remote_receive_size
```

Ispis rezultata dobiven "netperf_script" skriptom (Prilog A) koja automatizira mjerenje variranjem parametara izgleda ovako:

```
UDP UNIDIRECTIONAL SEND TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to
161.53.65.228 (161.53.65.228) port 0 AF_INET
Socket  Message  Elapsed      Messages
Size    Size     Time         Okay Errors   Throughput
bytes   bytes    secs            #      #   KBytes/sec
------------------------------------------------------------------
262144   32768   10.00       33383      0    106798.76
262144           10.00       33383           106798.76

262144   64000   10.00       17091      0    106809.39
262144           10.00       17091           106809.39

8388608   32768   10.00        33784      0    108096.27
8388608           10.00        33784           108096.27

8388608   64000   10.00        17412      0    108809.07
8388608           10.00        17412           108809.07
```

Prvi stupac (u dva reda) prikazuje veličinu predajnog i prijemnog međuspremnika, drugi stupac veličinu paketa koji se šalje, treći stupac trajanje testa, četvrti stupac (u dva reda) broj uspješno poslanih/primljenih paketa, peti stupac broj neuspješno prenesenih paketa, a šesti stupac rezultate testiranja. Vidimo da su kod UDP testa rezultati također prikazani u dva reda. Razlog tomu je što zbog nepouzdane datagramske UDP usluge neki paketi neće uspješno stići na odredište, pa je rezultat (propusnost) prikazan mjereno i sa pošiljateljske strane, ovisno o broju uspješno poslanih paketa, i sa primateljske strane, ovisno o broju uspješno primljenih paketa.

U optimizacijskom postupku uzeta je u obzir propusnost mjerena sa primateljske strane. Time je u analizu uvedena i mjera kvalitete prijenosa jer manje pogrešno prenesenih paketa povećava primateljsku propusnost.

3.1.3. Korištenje netperf-a u mjerenju zahtjev-odziv prijenosa TCP-om

Tipičan način pozivanja netperfa za mjerenje zahtjev-odziv prijenosa bi bio:

```
$ ./netperf -H remote_host -t TCP_RR [global_options] -- -s local_socksize
-S remote_socksize -r request_size,response_size
```

Ispis rezultata dobiven "netperf_script" skriptom (Prilog A) koja automatizira mjerenje variranjem parametara izgleda ovako:

```
TCP REQUEST/RESPONSE TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to
161.53.65.228 (161.53.65.228) port 0 AF_INET
Local /Remote
Socket Size   Request  Resp.   Elapsed  Trans.
Send   Recv   Size     Size    Time     Rate
bytes  Bytes  bytes    bytes   secs.    per sec
-------------------------------------------------------------------
8192   8192   4096     4096    10.00    3392.11
8192   8192
16384  16384  4096     4096    10.00    3307.81
16384  16384
32768  32768  4096     4096    10.00    3413.22
32768  32768
131072 131072 4096     4096    10.00    3414.45
131072 131072
262142 262142 4096     4096    10.00    3412.66
262142 262142
```

Prvi i drugi stupac prikazuju veličinu prijemnog i predajnog međuspremnika na lokalnom i udaljenom računalu, treći i četvrti stupac veličinu zahtjeva i odziva (zadane opcijom `-r`), peti stupac trajanje testa, a šesti stupac rezultate testiranja.

3.1.4. Korištenje netperf-a u mjerenju zahtjev-odziv prijenosa UDP-om

Tipičan način pozivanja netperfa za mjerenje zahtjev-odziv prijenosa bi bio:

```
$ ./netperf -H remote_host -t UDP_RR [global_options] -- -s local_socksize
-S remote_socksize -r request_size,response_size
```

Ispis rezultata dobiven "`netperf_script`" skriptom (Prilog A) koja automatizira mjerenje variranjem parametara izgleda ovako:

```
UDP REQUEST/RESPONSE TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to
161.53.65.228 (161.53.65.228) port 0 AF_INET
Local /Remote
Socket Size   Request  Resp.   Elapsed  Trans.
Send   Recv   Size     Size    Time     Rate
bytes  Bytes  bytes    bytes   secs.    per sec
-------------------------------------------------------------------
8192   8192   4096     4096    10.00    3653.21
8192   8192
16384  16384  4096     4096    10.00    3532.81
16384  16384
32768  32768  4096     4096    10.00    3486.80
32768  32768
131072 131072 4096     4096    10.00    3631.82
131072 131072
262142 262142 4096     4096    10.00    3634.68
262142 262142
```

Prvi i drugi stupac prikazuju veličinu prijemnog i predajnog međuspremnika na lokalnom i udaljenom računalu, treći i četvrti stupac veličinu zahtjeva i odziva (zadane opcijom `-r`), peti stupac trajanje testa, a šesti stupac rezultate testiranja.

4. Optimizacija TCP parametara

Teorijski maksimum propusnosti Ethernet-a se računa iz omjera korisnih podataka i dodanih podataka (engl. overhead). Standardni Ethernet okvir sadrži 1500 okteta. Dodani podaci su Ethernet preambula od 8 okteta, 6 okteta za odredišnu i 6 za izvorišnu adresu, 2 okteta za duljinu te 4 okteta za zaštitni kod (FCS, Frame Checksum). U sklopu okvira od 1500 okteta se osim korisnih podataka prenose i zaglavlje mrežnog sloja (IP) od 20 okteta i zaglavlje prijenosnog sloja (TCP) od minimalno 20 okteta. Također, na gigabitnom Ethernetu za koji je i obavljana optimizacija se ukupno 12 okteta koristi za inkapsulaciju okvira (Inter Frame Spacing).

Iz navedenih podataka možemo izračunati maksimalnu teorijsku propusnost gigabitnog Etherneta:

$$\frac{1500-20-20}{8+6+6+2+1500+4+12}\cdot 100\,\% = 94.928\,\%$$

$$94.928\,\%\cdot 1\mathrm{Gb}/s = 972.07\,\mathrm{Mb/s} = 121.51\,\mathrm{MB/s}$$

Optimizirat ćemo TCP/IP parametre da povećamo propusnost što je moguće više; u idealnom slučaju da postignemo maksimalnu teorijsku propusnost.

Naravno, u postupku optimiranja bilo bi neefikasno isprobavati sve moguće kombinacije parametara i mjeriti propusnost. U postupku optimiranja ćemo se ipak voditi teorijskim znanjem o mogućem utjecaju parametara na funkcioniranje TCP/IP-a.

4.1. Veličina paketa s aplikacijske razine

Strujeći, masovni prijenosi mogu se prenositi u većim i manjim paketima (payload), gledano sa aplikacijske razine. Prenošenjem u manjim paketima moraju se proporcionalno češće pozivati funkcije za slanje/primanje paketa preko pristupne točke. Te funkcije su funkcije jezgre operacijskog sustava, te se pozivaju skupim sistemskim pozivom. Stoga bi mogli očekivati nešto bolje performanse ako šaljemo podatke u velikim paketima. Također, prenošenjem manjim paketima veći je i utjecaj veličine zaglavlja.

4.2. Veličina međuspremnika i TCP prozora

Dva najvažnija faktora o kojima ovise performanse TCP-a su propusni opseg veze (bandwidth) i vrijeme povrata (Round-Trip Time). Te dvije vrijednosti određuju BDP, umnožak propusnog opsega i kašnjenja (Bandwidth Delay Product):

$$\mathrm{BDP} = \frac{\mathrm{Bandwidth}\cdot \mathrm{RTT}}{2}$$

Fizički, BDP predstavlja količinu podataka (bitova) "u zraku", tj. broj poslanih bitova koji još nisu primljeni, tj. još nisu stigli na prijemnu stranu. Kod TCP-a koji garantira isporuku paketa BDP je posebno važan jer predstavlja količinu poslanih i još nepotvrđenih podataka koje predajnik mora čuvati u međuspremniku u slučaju da prijemnik zatraži ponovno slanje paketa, na primjer kada je paket izgubljen ili primljen s greškom.

Stoga, možemo pretpostaviti da bi trebalo postaviti veličine TCP prozora i svih međuspremnika na barem dvostruku vrijednost BDP-a. Tada je, u idealnim uvjetima kad nema pogreški, mrežni cjevovod

između predajnika i prijemnika cijelo vrijeme pun podataka, te postižemo najbolje moguće performanse.

4.3. Količina TCP memorijskog prostora

Svaki operacijski sustav, pa tako i Linux ima mehanizme za kontrolu veličine sistemske memorije, pa tako i za kontrolu količine sistemske memorije alocirane za svaku TCP konekciju.

Poželjno je osigurati da TCP ima dovoljno sistemske memorije za sve međuspremnike i sve veze. Stoga treba, u ovisnosti o količini fizičke memorije, povećati količinu sistemskog memorijskog prostora za TCP. To će spriječiti nepotrebre alokacije/dealokacije u toku rada, a također će spriječiti i posebne sistemske algoritme za rukovanje memorijom koji se uključe u "pressure" načinu rada.

Ne treba pretjerivati da se ne dogodi da ostane premalo memorije za ostatak jezgre operacijskog sustava, te za aplikacije, te da se ne počne koristiti virtualna memorija na tvrdom disku.

4.4. Duljina ulaznog i izlaznog reda

Veličina izlaznog (interface queue size) reda je veličina izlaznog reda na granici jezgre Linux operacijskog sustava i Ethernet sučelja (txqueuelen). Određuje maksimalnu veličinu međuspremnika paketa na Ethernet sučelju.

Veličina ulaznog (`netdev_max_backlog`) reda je veličina ulaznog reda na granici jezgre Linux operacijskog sustava i Ethernet sučelja. Linux jezgra u redovnitim intervalima (scheduling mechanism) prebacuje podatke u TCP ulazni međuspremnik.

Povećavanjem ovih veličina trebali bismo, do neke mjere, povećati performanse.

4.5. Upotreba TCP opcije skaliranja prozora

Trebamo omogućiti skaliranje TCP prozora, jer bi u suprotnosti TCP prozor bio ograničen na 64KB, a na gigabitnim mrežama je preporuka zbog većeg BDP-a povećati TCP prozor.

4.6. Upotreba TCP opcije vremenskih oznaka

Opcija uvodi dodatnih 12 okteta u TCP zaglavlje, a na lokalnim mrežama nema velikog smisla. Služi za precizno mjerenje vremena povrata (RTT), te pomaže algoritmima za kontrolu zakrčenja.

Pretpostavka je da će onemogućavanje ove opcije dovesti do neznatnog povećanja performansi.

4.7. Upotreba TCP opcije SACK

Selektivne potvrde (SACK, Selective Acknowledgments) omogućavaju TCP prijemniku da obavijesti predajnika koji se točno okviri trebaju ponovo poslati. Obavlja se selektivna retransmisija samo oštećenog okvira. Inače bi prijemnik morao odbaciti sve okvire primljene nakon okvira s pogreškom (protokol s vraćanjem).

Teško je predvidjeti utjecaj SACK-a na performanse na lokalnoj mreži. Za usnopljene pogreške bolji je protokol s vraćanjem, dok SACK pomaže kod izoliranih pogreški.

4.8. Upotreba kontrole zakrčenja

Imati kontrolu zakrčenja na lokalnoj mreži nema smisla, a smanjuje performanse. Nažalost, algoritam za kontrolu zakrčenja mora postojati, ne može se potpuno isključiti, a podrazumijevani algoritam je *reno*. Stoga treba pronaći najprikladniji algoritam, a to će vjerojatno biti onaj najjednostavniji, koji ne usporava jezgru operacijskog sustava.

Postoji više algoritama za kontrolu zakrčenja, svaki sa svojim prednostima i nedostacima, a slijedeći su korišteni u ispitivanjima:

TCP Reno je standardni algoritam u Linux jezgrama verzije 2.4. Podešava prozor zagušenja u ovisnosti o omjeru izgubljenih i uspješno prenesenih paketa. Implementira polagani start (engl. slow start). Propusnost polagano raste dok ne ostane stabilna; prozor se povećava za fiksnu vrijednost nakon svake primljene potvrde. Za izgubljeni paket prozor se multiplikativno umanjuje.

TCP BIC je podrazumjevani algoritam u Linux jezgrama verzije 2.6. Modifikacija je Reno algoritma, a prilagođen jako brzim mrežama s velikim kašnjenjem (engl. Long Fat Networks). Prozor zakrčenja se aditivno inkrementira ako je već velik; ako je mali tada se uvećava binarnim pretraživanjem (engl. Binary Search Increase). Veličina prozora zakrčenja prije i nakon umanjenja, tj. nakon redukcije uzrokovane izgubljenim paketom, ulaze kao parametri u algoritam binarnog pretraživanja. Prozor se umanjuje multiplikativno.

TCP Vegas mjeri vrijeme povrata (RTT) i tom mjerom evaluira kvalitetu veze. Koristi aditivno povećanje i aditivno umanjenje prozora zakrčenja. Pokušava izbjeći gubitak paketa.

TCP Westwood je modifikacija Reno algoritma, prilagođena bežičnim mrežama. Analizira dinamične promjene u opterećenju mreže promatranjem paketa potvrde.

5. Rezultati mjerenja

Mjerenja su vršena na dva računala sa instaliranim Linux operacijskim sustavom, distribucije Fedora Core 6, te gigabitnim Ethernet mrežnim karticama. Računala su preko Ethernet kartica spojena na gigabitni preklopnik. Inicijalne vrijednosti TCP parametara na računalima prije postupka optimizacije prikazane su u tablici 5.1.

Tablica 5.1: Inicijalne vrijednosti TCP parametara na testnim računalima

rmem_default	131071
rmem_max	131071
wmem_default	131071
wmem_max	131071
tcp_mem	38912 51882 77824
tcp_wmem	4096 16384 1660224
tcp_rmem	4096 87380 1660224
tcp_window_scaling	1
tcp_timestamps	1
tcp_sack	1
tcp_fack	1
tcp_dsack	1
tcp_congestion_control	bic
tcp_ecn	0
tcp_low_latency	0
ip_no_pmtu_disc	0
tcp_mtu_probing	0
tcp_base_mss	512
netdev_max_backlog	1000
txqueuelen	1000
MTU	1500

5.1. Mjerenje vremena povrata

Prvo se u seminaru pristupilo mjerenju vremena povrata (RTT) koji je potreban za izračunavanje BDP-a, te na taj način i optimalne veličine međuspremnika.

Vrijeme povrata se mjerilo *ping* naredbom, što možda i nije najpreciznije mjerenje. Stoga je mjerenje vršeno u serijama u raznim vremenskim trenucima. Serija se sastojala od deset mjerenja, a ukupno su izvršene dvadeset i tri serije.

Ukupni rezultati su slijedeći:

RTT				
Min [ms]	Avg [ms]	Max [ms]	MDev [ms]	Median [ms]
0,105	0,213	1,432	0,206	0,147

Median je izračunat nad srednjim vrijednostima serija, dok su minimum, maksimum i srednja vrijednost računati nad svim mjerenjima. S obzirom na relativno visoku srednju devijaciju, sumjerljivu sa samom srednjom vrijednošću, te na nakošenost krivulje kojoj je median manji od srednje vrijednosti, pri računanju BDP-a je za RTT uzeta dvostruka srednja vrijednost.

$$RTT = 2 \cdot RTTavg = 0{,}426\,ms$$

$$BDP = Bandwidth \cdot RTT = 1073741824\,bit/s \cdot 0{,}426\,ms = 457414{,}017\,bit$$

$$BDP = \frac{457414{,}017}{8 \cdot 1024} = 55{,}837\,KB$$

Teorija kaže da bi optimalna veličina TCP prozora, te veličina međuspremnika trebala iznositi dvostruku vrijednost BDP-a:

$$OPT = 2 \cdot BDP = 111{,}673\,KB$$

Količina memorije zadana TCP varijablama `tcp_rmem`, `rmem_default`, i drugim dijeli se na TCP i aplikacijski međuspremnik. Stoga moramo alocirati veću količinu memorije, a s obzirom na pretpostavljene vrijednosti varijabli `tcp_adv_win_scale` i `tcp_app_win`, potrebna količina jezgrene memorije za optimalne međuspremnike iznosi:

$$OPT' = \frac{4}{3} \cdot OPT = 148{,}898\,KB$$

5.2. Ovisnost propusnosti o veličini TCP prozora

Izračunali smo da teorijska optimalna veličina međuspremnika i TCP prozora iznosi 148.898 KB. Ovdje valja naglasiti da iste TCP varijable `tcp_rmem`, `tcp_wmem`, `rmem_default`, `wmem_default`, `rmem_max`, `rmem_max` reguliraju veličinu međuspremnika pristupnih točaka (engl., socket buffers), te posredno veličinu TCP prozora. Stoga ćemo nadalje koristiti pojam *veličina međuspremnika* za vrijednosti upisane u TCP varijable koje utječu i na TCP prozor i na sve međuspremnike. Prema temeljnoj TCP specifikaciji veličina TCP prozora je ograničena na 65536 okteta. Opcijom Window Scale, tj varijablom `tcp_window_scaling` se uklanja to ograničenje. Već po teoriji se vidi da je opcija Window Scale potrebna za optimalan prozor (148KB > 64KB). Naravno, ovo vrijedi samo za TCP, UDP je uvijek ograničen na 65507 okteta.

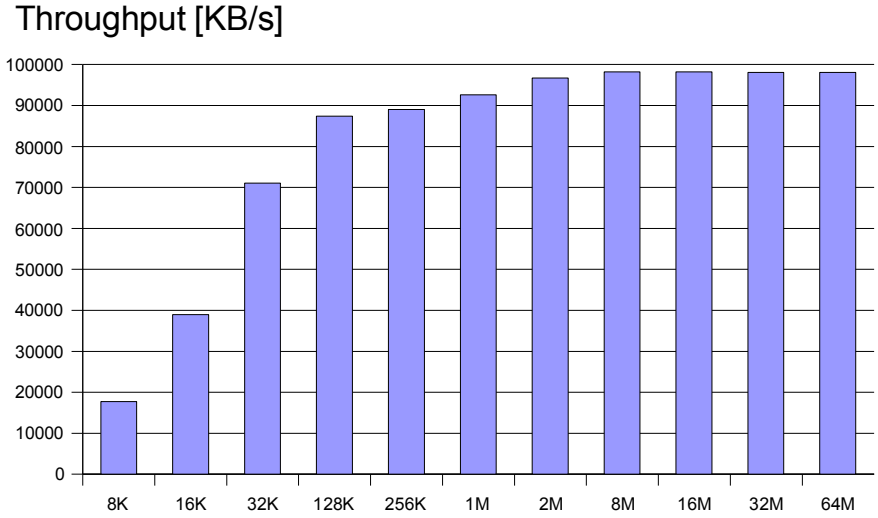
Povećavanjem TCP prozora i međuspremnika se također mora osigurati i da jezgra operacijskog sustava alocira dovoljno velik memorijski prostor za TCP. Varijablom `tcp_mem` treba se osigurati dovoljno memorijskih stranica da stanu svi TCP međuspremnici za svaku otvorenu pristupnu točku.

U nastavku, među mjerenjima, podrazumjevano je da su varijable `tcp_window_scaling` i `tcp_mem` pratile promjene varijabli `tcp_rmem`, `tcp_wmem`, `rmem_default`, `wmem_default`, `rmem_max`, `rmem_max` gdje god je potrebno. U suprotnom, promjene tih parametara se ne bi mogle u potpunosti očitovati.

Time su za dva parametra nađene optimalne vrijednosti kao funkcije parametara `tcp_rmem`, `tcp_wmem`, `rmem_default`, `wmem_default`, `rmem_max` i `rmem_max`.

5.2.1. TCP u masovnim prijenosima

Ovisnost propusnosti o veličini međuspremnika i TCP prozora vidi se na slici 5.1. Može se primjetiti da je u početku velika ovisnost propusnosti o veličini međuspremnika , a potom dolazi do zasićenja. Također se vidi da je granica otprilike na 128 KB, a to je upravo blizu izračunatog teorijskog optimalnog prozora od 148 KB.



Slika 5.1: TCP Stream - Ovisnost propusnosti o veličini međuspremnika

Na 128 KB propusnost iznosi 87417,64 KB/s, dok je maksimalna izmjerena propusnost na 64MB iznosila 98245,89 KB/s. Znači, povećanje TCP prozora od 500% je prouzročilo povećanje propusnosti od 12%. Naravno, sa stanovišta fizičke memorije koja je ograničena, ne treba postavljati nerazumno velike TCP prozore i međuspremnike jer se može negativno odraziti na rad cijelog sustava.

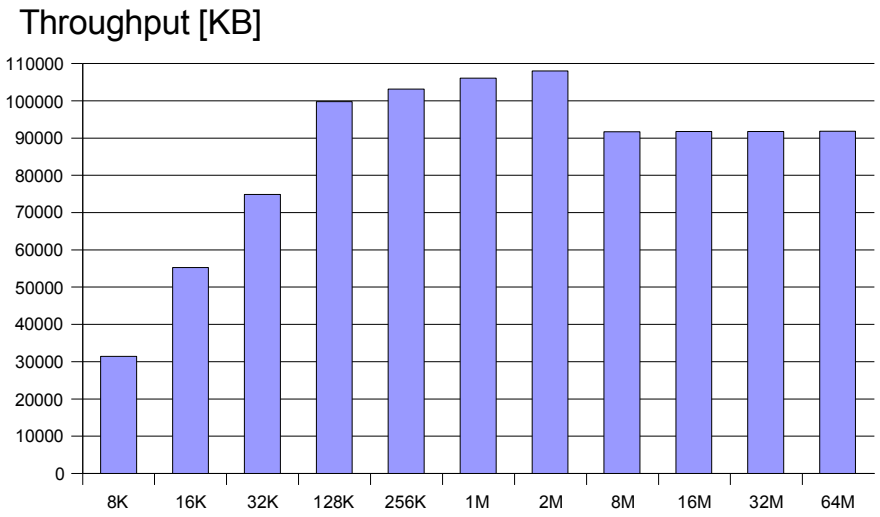
Ranije je izračunato da je maksimalna teoretska propusnost TCP-a na gigabitnoj mreži 121510 KB/s. Znači, postignuto je 80% od maksimalne teorijske propusnosti.

5.2.2. UDP u masovnim prijenosima

Sa slike 5.2 se uočava da i UDP prijenos ovisi o veličinama koje se reguliraju TCP postavkama, zato jer te veličine određuju veličinu ulaznih i izlaznih međuspremnika pristupnih točaka, a također i veličinu aplikacijskog međuspremnika sa strane jezgre operacijskog sustava.

Ipak, kvalitativno gledano, graf nema isti oblik kao i kod TCP Stream mjerenja. Povećavanjem međuspremnika iznad 2MB protok počne znatno padati. Naime, UDP je nepouzdana datagramska usluga, i valja naglasiti da su kod mjerenja protoka uzete veličine mjerene sa prijemne strane, dakle u obzir je uzet samo onaj dio podataka koji je primljen bez pogreške. Jedino kod međuspremnika od 256KB, 1MB i 2MB nije bilo nikakvih pogreški u prijenosu, niti jedan datagram nije bio izgubljen ili primljen s greškom. Znači, povećavanjem međuspremnika iznad 2MB su se očito usporavali drugi dijelovi sustava, najvjerojatnije jezgra operacijskog sustava, zbog gladovanja za memorijom. Međuspremnici od 2MB su očito gornja granica na mjernim računalima.

Maksimalan protok izmjeren UDP-om iznosi 108489,02 KB/s, što iznosi 89% teorijski maksimalnog protoka. Razlog u, naizgled, boljem ponašanju UDP-a od TCP-a (98245,89 KB/s, 80%) u masovnim prijenosima je vjerojatno u tome što je UDP "lakši" protokol (engl. lightweight), te ima i kraće zaglavlje od TCP-a.

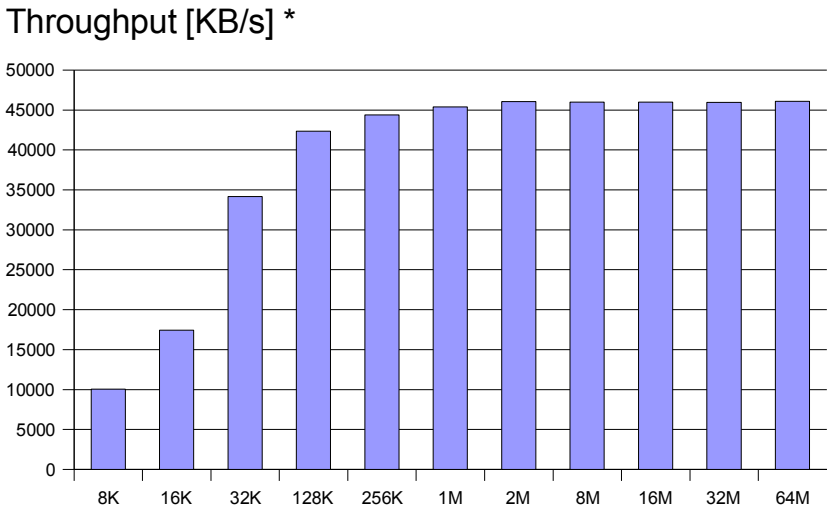


Slika 5.2: UDP Stream - Ovisnost propusnosti o veličini međuspremnika

Optimalna veličina međuspremnika je negdje oko 128KB, baš kao i u TCP Stream mjerenju, te se poklapa sa teorijskim optimumom od 148KB.

5.2.3. TCP u zahtjev-odziv prijenosima

Performanse zahtjev odziv prijenosa mjere se u transakcijama u sekundi. S obzirom da broj transakcija u sekundi najviše ovisi o veličini zahtjeva i odziva (u oktetima), a tek onda o ostalim parametrima, to je performansa prikazana u kilobajtima u sekundi. Rezultat svakog mjerenja, u transakcijama po sekundi, pomnožen je sa veličinom zahtjeva i odziva u tom mjerenju. Time je zapravo dobiven protok.



Slika 5.3: TCP RR - Ovisnost propusnosti o veličini međuspremnika

* Throughput = TransactionRate[1/s] * Payload[KB]

Razlika u odnosu na prethodna mjerenja (masivna strujeća) se najviše očituje u činjenici da sada oba računala i šalju i primaju podatke. To će se direktno primjetiti i u rezultatima koji sada pokazuju otprilike dvostruko manje iznose; transakcija je jedan zahtjev uspješno poslan i jedan odziv uspješno

primljen, pa kad se to pomnoži sa veličinom paketa dobijemo veličinu jednaku protoku u jednom smjeru i protoku u drugom smjeru.

Rezultati mjerenja protoka mogu se vidjeti na slici 5.3. Vidi se ovisnost o veličini međuspremnika i TCP prozora slična kao i kod TCP Stream mjerenja. Također, optimum u samoj veličini međuspremnika leži negdje između 128KB i 256KB.

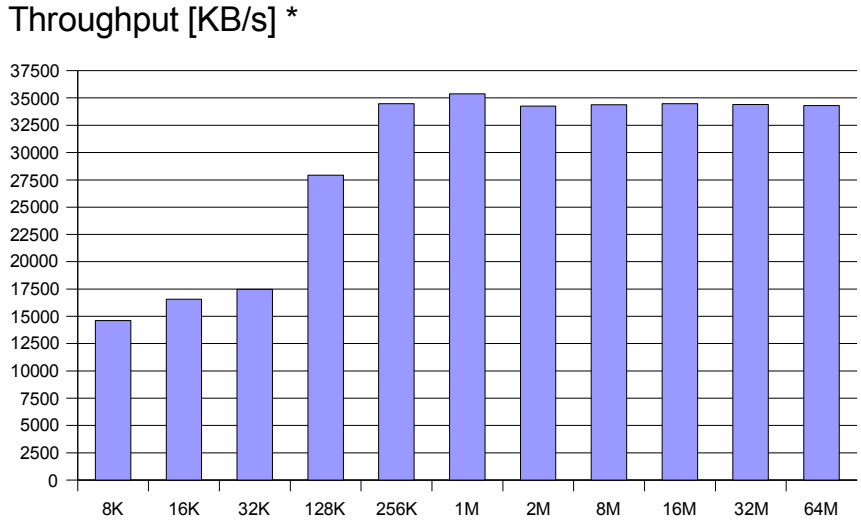
Maksimalan protok izmjeren u TCP zahtjev-odziv testu iznosi 46090,24 KB/s. Da usporedimo ovu vrijednost s onima iz prošlih testova, moramo ju pomnožiti sa dva da dobijemo ukupan protok podataka koji je ostvaren, dakle 92180.48 KB/s, ili 76% od maksimalnog. TCP daje nešto lošije rezultate u zahtjev-odziv prijenosu, nego u strujećim prijenosima. Očiti razlog tomu je to što se razbija cjevovod, tj. sam tok (stream) podataka tokom svake promjene smjera prijenosa.

5.2.4. UDP u zahtjev-odziv prijenosima

Ovdje također vrijede napomene iz TCP zahtjev-odziv mjerenja, protok podataka računamo množenjem broja transakcija u sekundi koje smo izmjerili sa veličinom zahtjeva i odziva. Naravno, s obzirom na zahtjev-odziv mjerenja imamo obosmjerni prijenos, te je sam protok prikazan iz perspektive jednog smjera (jednog toka). Da dobijemo onaj ukupni protok, moramo izračunati protok pomnožiti sa dva.

Sa slike *5.4* je vidljiv kvalitativno isti oblik krivulje kao u prethodnim testovima, pogotovo kao u UDP Stream mjerenju. Optimum u veličini međuspremnika je opet između 128KB i 256KB, mada je ovaj put naizgled veća razlika između rezultata u tim točkama, čemu je vjerojatno uzrok šum u mjerenjima.

Najbolji postignut rezultat, onaj na 1MB, iznosi 37613,75 KB/s, što pomnoženo sa dva daje 75227,50 KB/s, ili 62% od maksimalnog teorijskog protoka.



Slika 5.4: UDP RR - Ovisnost propusnosti o veličini međuspremnika

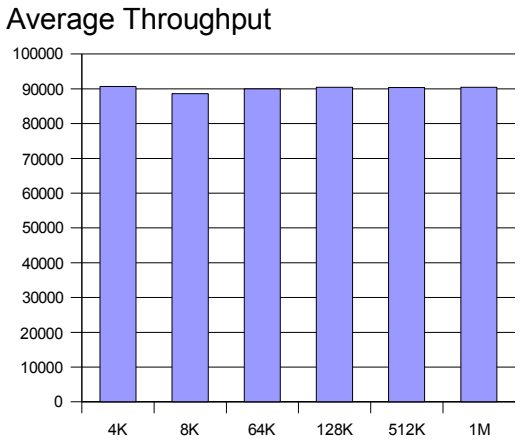
* Throughput = TransactionRate[1/s] * Payload[KB]

Ovdje vidimo da je UDP u zahtjev-odziv testu pokazao najslabije rezultate, dok je UDP u masovnom prijenosu pokazao najbolje rezultate. Očiti uzrok tome je činjenica da uspješnu transakciju čini uspješno prenesen zahtjev i uspješno prenesen odziv. Ako dođe do pogreške bilo u prijenosu zahtjeva, bilo odziva, cijela transakcija je neuspješna. To uvelike narušava broj uspješnih transakcija, a time i broj transakcija u sekundi. S druge strane, UDP u masovnim strujećim prijenosima daje najbolje rezultate zbog svoje jednostavnosti, zbog koje ne opterećuje sustav. Zato se mnogo algoritama za

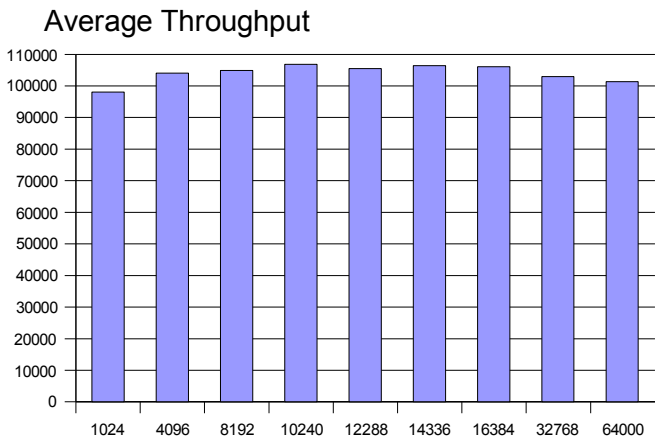
prijenos podataka između računala na klasteru (engl. cluster) bazira na UDP-u sa malom TCP vezom koja služi za prijenos kontrolnih podataka. UDP-om se prenose podaci, dok se kontrolnom TCP vezom prenose potvrde i podaci kontrole toka.

5.3. Veličina paketa s aplikacijske razine

Na slikama 5.6 i 5.5 možemo vidjeti ovisnost propusnosti o veličini paketa u TCP Stream i UDP Stream testovima. S obzirom na veličinu paketa s aplikacijske razine, uočavamo da testovi UDP-a i TCP-a u masovnim prijenosima ne pokazuju zavisnost. Graf UDP-a pokazuje blagu zakrivljenost sa izbočinom (maksimumom) između 8K - 16K, no to može biti posljedica šuma u mjerenjima.

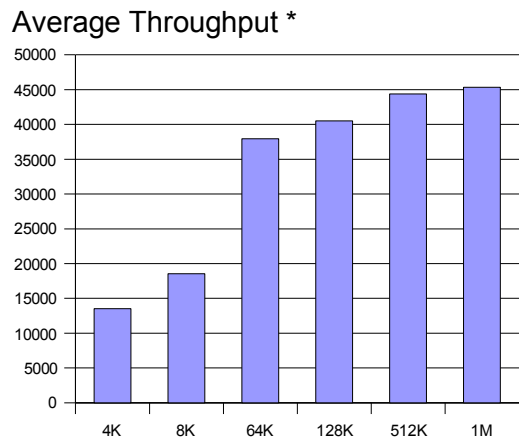


Slika 5.6: TCP Stream - Ovisnost propusnosti o veličini paketa

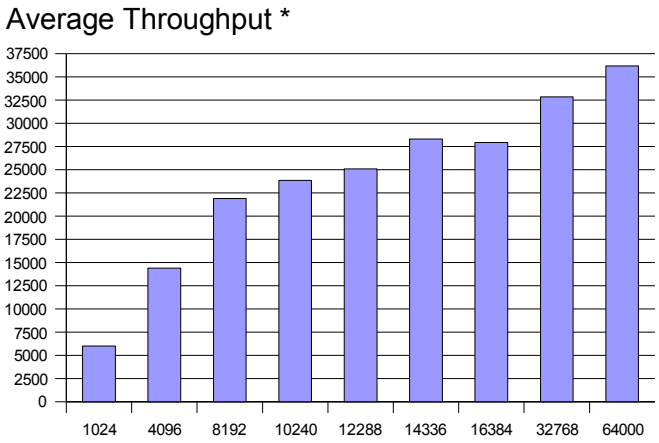


Slika 5.5: UDP Stream - Ovisnost propusnosti o veličini paketa

Zapravo je takvo ponašanje i očekivano; s obzirom da jezgra operacijskog sustava za svaku pristupnu točku alocira i aplikacijske međuspremnike u koje se smještaju podaci koje spora aplikacija ne uspjeva pročitati. Jezgra alocira aplikacijske međuspremnike kao posljedicu postavki `tcp_adv_win_scale` i `tcp_app_win` varijabli.



Slika 5.8: TCP RR - Ovisnost propusnosti o veličini paketa



Slika 5.7: UDP RR - Ovisnost propusnosti o veličini paketa

Za razliku od masovnih prijenosa, TCP i UDP u zahtjev-odziv prijenosima pokazuju drugačije ovisnosti o veličini paketa, kao što se vidi sa slika 5.8 i 5.7.

Naravno, broj transakcija u sekundi je obrnuto proporcionalan veličini paketa. No za performanse je važna mjera količine prenesenih podataka u jedinici vremena. Na grafovima se vide ovisnosti protoka o veličini paketa. Puna propusnost se dobije množenjem prikazane propusnosti sa dva. To je stvarna propusnost u oba smjera.

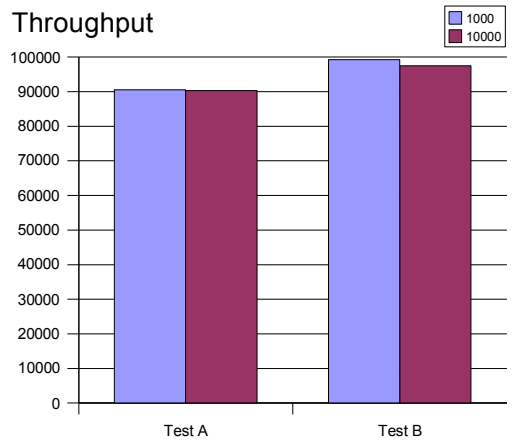
Vidi se da zapravo propusnost raste sa veličinom paketa, što je i logično jer ima manje promjena smjera prijenosa iz slanja u primanje, i obratno. Naravno, povećavanjem veličine paketa gubi se smisao zahtjev-odziv prijenosa; počinje sve više ličiti na masovni strujeći prijenos.

5.4. Ovisnost o duljini izlaznog i ulaznog reda sučelja

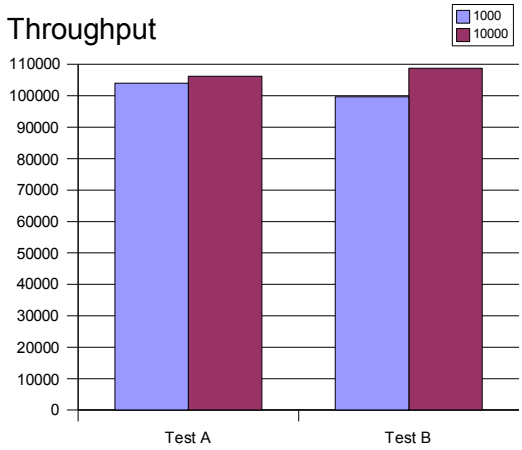
Txqueuelen (`ipconfig` *`interface`* `txqueuelen` *`value`*) određuje duljinu izlaznog reda sučelja u paketima; u taj red se spremaju paketi koje jezgra operacijskog sustava prebrzo šalje mrežnoj kartici, tj. fizičkom sloju.

Netdev_max_backlog određuje duljinu ulaznog reda sučelja u paketima; kad sučelje prima pakete brže nego što ih jezgra operacijskog sustava stigne obraditi i poslati aplikaciji.

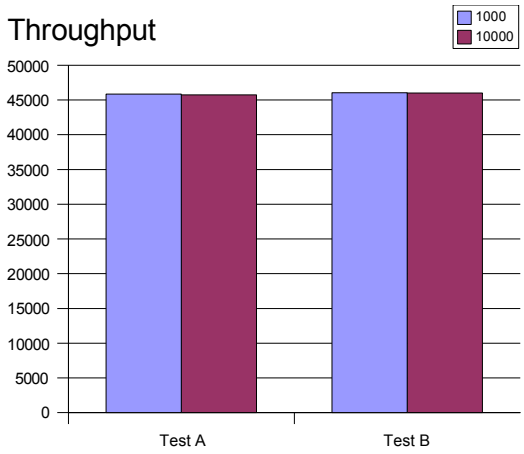
U svim mjerenjima, svi testovi su pokazali da propusnost ne ovisi o duljini ovih redova (slike 5.9, 5.10, 5.11 i 5.12). Male razlike su tu samo usljed šuma. Ovakvi rezultati bili su očekivani; međuspremnici na sučelju ne utječu direktno na TCP/UDP performanse. Prilikom dužih mjerenja, tj. slanja ogromne količine podataka, međuspremnici ne mogu kompenzirati bilo kakva uska grla. Međuspremnici na sučelju dobro dođu samo u slučajevima kratkih i sporadičnih zastoja ili problema u prijenosu.



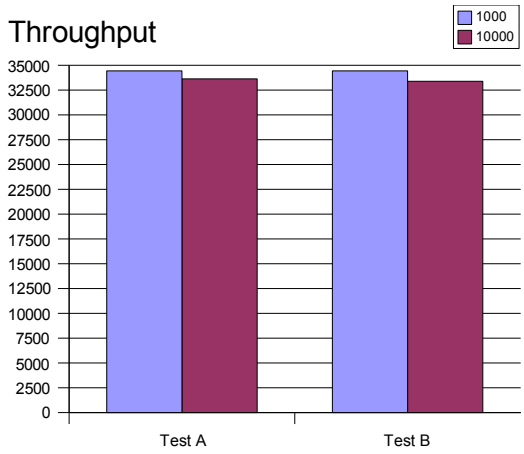
Slika 5.9: TCP Stream - Ovisnost propusnosti o duljini redova sučelja



Slika 5.10: UDP Stream - Ovisnost propusnosti o duljini redova sučelja



Slika 5.11: TCP RR - Ovisnost propusnosti o duljini redova sučelja

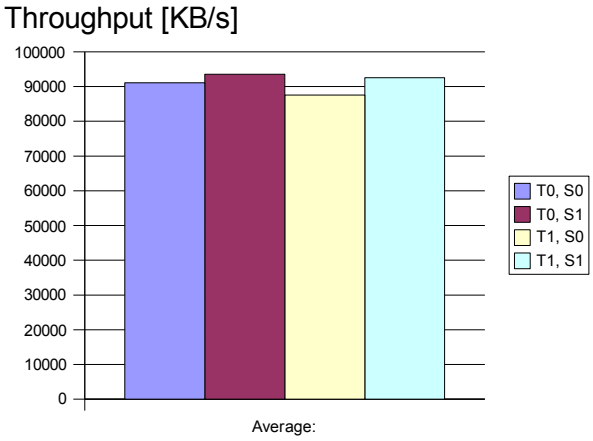


Slika 5.12: UDP RR - Ovisnost propusnosti o duljini redova sučelja

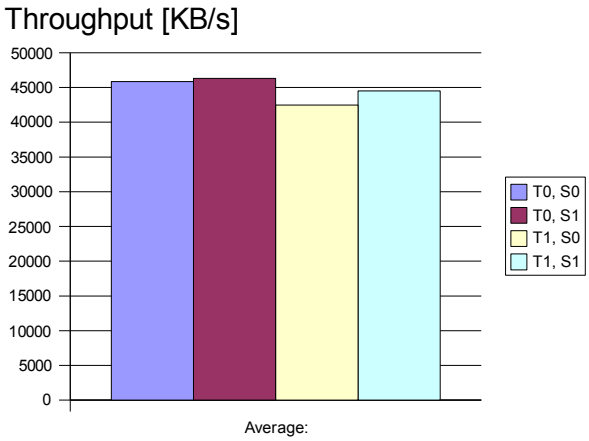
5.5. Upotreba selektivnih potvrda i vremenskih oznaka

TCP opcije selektivnih potvrda (engl. Selective Acknowledge, SACK) i vremenskih oznaka (engl. timestamps) također utječu na performanse. Selektivne potvrde omogućavanju potvrđivanje podataka izvan sljeda, što je nemoguće sa osnovnom kumulativnom TCP potvrdom. Vremenske oznake služe preciznom mjerenju vremena povrata (Round-Trip Measurement), a dodaju 12 okteta u TCP zaglavlje.

Na slikama 5.13 i 5.14 se vidi utjecaj tih opcija na protok podataka TCP-om u masovnim i zahtjev-odziv prijenosima. U oba je slučaja propusnost najveća kada je selektivno potvrđivanje uključeno i kada su vremenske oznake isključene. I upravo suprotno, najslabija je propusnost u slučaju uključenih vremenskih oznaka i isključenog selektivnog potvrđivanja. U oba slučaja, povećanje propusnosti podešavanjem opcija "Timestamps" i "SACK" iznosi 9%.

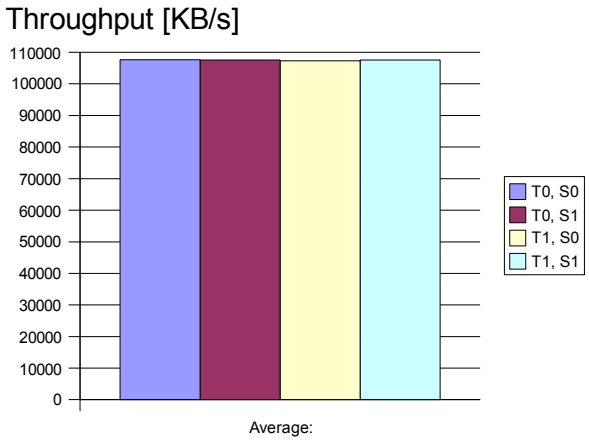


Slika 5.13: TCP Stream - utjecaj opcija "Timestamps" i "SACK" na propusnost

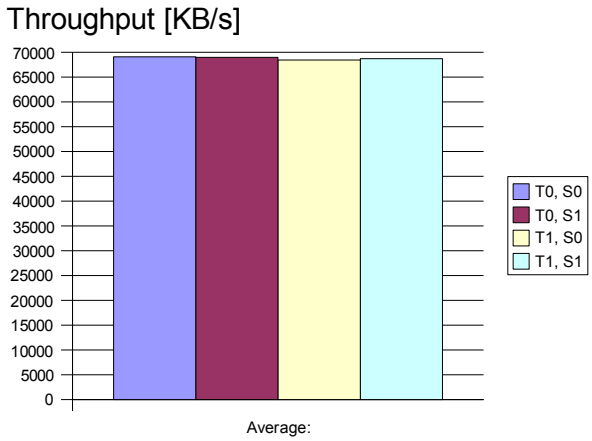


Slika 5.14: TCP RR - utjecaj opcija "Timestamps" i "SACK" na propusnost

Kao što je i za očekivati, ove opcije nemaju utjecaja na UDP performanse (slike 5.15 i 5.16).



Slika 5.15: UDP Stream - utjecaj opcija "Timestamps" i "SACK" na propusnost

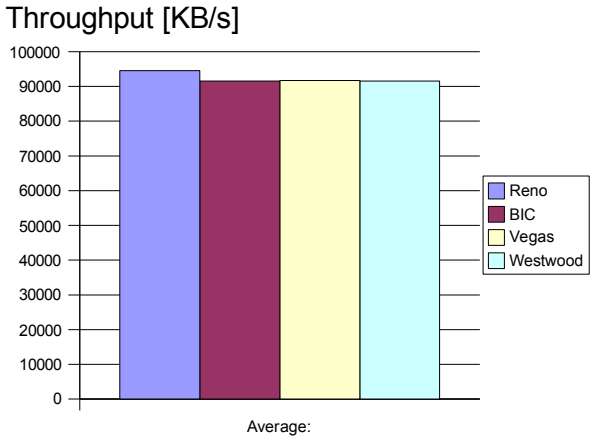


Slika 5.16: UDP RR - utjecaj opcija "Timestamps" i "SACK" na propusnost

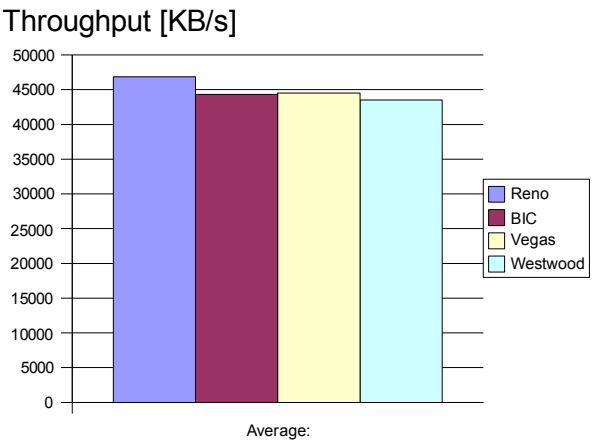
5.6. Upotreba kontrole zakrčenja

Kontrola zakrčenja se općenito ne može isključiti. No možemo odabrati jedan od instaliranih i uključivih (engl., pluggable) algoritama kontrole zakrčenja. Na ispitnim računalima bila su instalirana četiri algoritma: Reno, Vegas, BIC i Westwood.

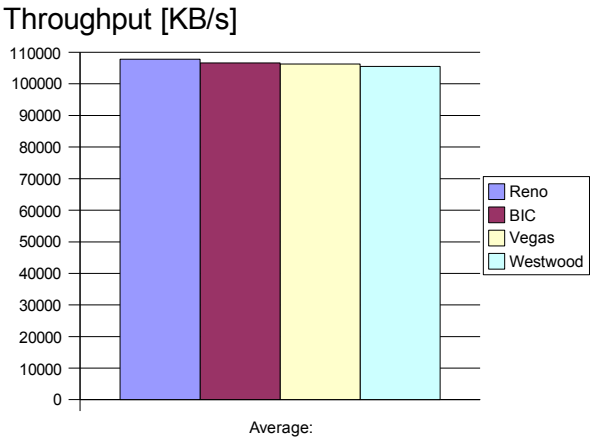
Na slikama 5.17, 5.18, 5.19 i 5.20 se vidi ovisnost propusnosti o upotrebljenom algoritmu za kontrolu zakrčenja. Iako su ispitna računala spojena direktno preko preklopnika, te nijedno drugo računalo nije bilo spojeno na preklopnik, pokazalo se da postoji blaga ovisnost propusnosti o algoritmu za kontrolu zakrčenja, pa čak i u UDP komunikaciji. Izrenađujuće, upravo standardni i već pomalo zastarjeli Reno je pokazao najbolje rezultate, vjerojatno zbog svoje jednostavnosti, a i zato jer na lokalnoj mreži ni ne treba kontrola zakrčenja. Ostali korišteni algoritmi su vjerojatno svojom većom složenošću malo jače opteretili jezgru operacijskog sustava i time doprinjeli nešto slabijoj propusnosti.



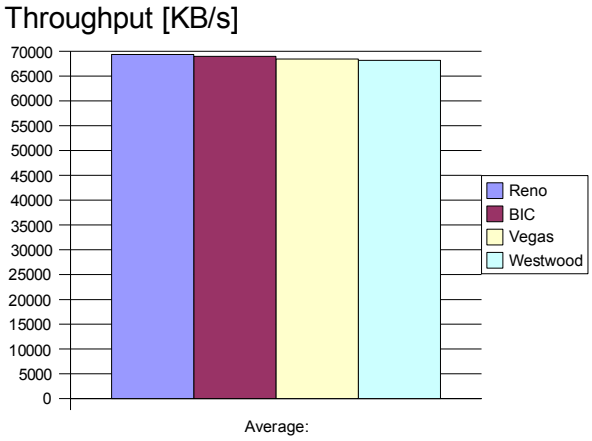
Slika 5.17: TCP Stream - utjecaj algoritma za kontrolu zakrčenja na propusnost



Slika 5.18: TCP RR - utjecaj algoritma za kontrolu zakrčenja na propusnost



Slika 5.19: UDP Stream - utjecaj algoritma za kontrolu zakrčenja na propusnost



Slika 5.20: UDP RR - utjecaj algoritma za kontrolu zakrčenja na propusnost

5.7. Multipleksiranje sa aplikacijske razine

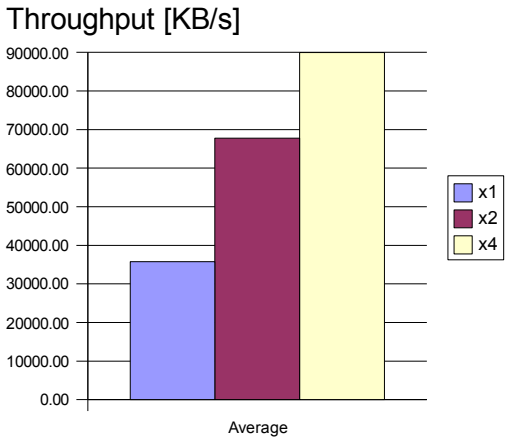
Mnogo je slučajeva kad iz raznih razloga nismo u mogućnosti promjeniti neke TCP parametre, na primjer nemamo administratorske privilegije. Također, naći optimalnu kombinaciju parametara nije lak, a ni jednoznačan problem. Optimalnost parametara ovisi i o vanjskim utjecajima, hardverskoj konfiguraciji radnih strojeva, verziji operacijskog sustava, količini aplikacijskih procesa koji se izvršavaju te njihovom korištenju procesora (engl. CPU Usage). Također, u komunikaciji, svako računalo ovisi i o svom partnerskom računalu, te o vanjskim utjecajima na njihovu fizičku vezu. Zato su za masovne prijenose između partnerskih lokalno umreženih računala (npr. za klastere) smišljeni mnogi novi algoritmi i protokoli, te modifikacije istih, koji optimiraju propusnost veze. Većina tih algoritama je aplikacijske razine i bazira se na multipleksiranju otvaranjem više veza transportnog sloja. Zatim dijeli tok podataka na nekoliko tokova i svaki dio šalje zasebnom vezom transportnog sloja. To naravno znači da aplikacija otvara više pristupnih točaka umjesto jedne. Mnogi optimizatori skidanja datoteka sa interneta se baziraju na ovom principu.

Jedan od takvih, češće korištenih algoritama je RBUDP (Reliable Blast UDP). RBUDP je agresivan algoritam masovnog prijenosa, namjenjen visokoj propusnosti, ali i pouzdanosti prijenosa. Algoritam otvara više UDP veza kojima šalje po dio toka. Na prijemnoj strani se tokovi ponovo spajaju u jedan. Algoritam također otvara i jednu TCP vezu koju koristi za kontrolne podatke: kontrola toka, slanje potvrde i poruke o pogrešci, itd.

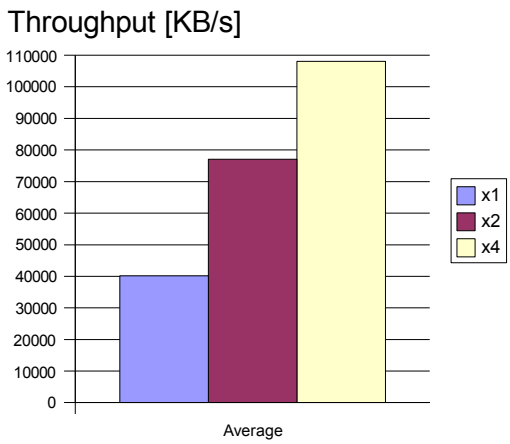
Takvi algoritmi, zbog dijeljenja toka, nemaju toliko problema sa memorijom, veličinom međuspremnika i veličinom TCP prozora. Također, izbjegava se ručno podešavanje TCP parametara, odnosno varijabli jezgre operacijskog sustava.

Pristupilo se mjerenju takvom, agresivnijom vrstom prijenosa. Interesantno je bilo izmjeriti koliko se može postići takvim multipleksiranjem. Nije korišten konkretno RBUDP, a niti neki drugi algoritam, već opet netperf u TCP i UDP načinu rada. Samo, ovaj put je pokretano istovremeno nekoliko netperf mjerenja, paljenjem netperf-a kao pozadinskog procesa Linux naredbom "**&**". Time je odsimulirano takvo multipleksiranje i izmjerena takva komunikacija.

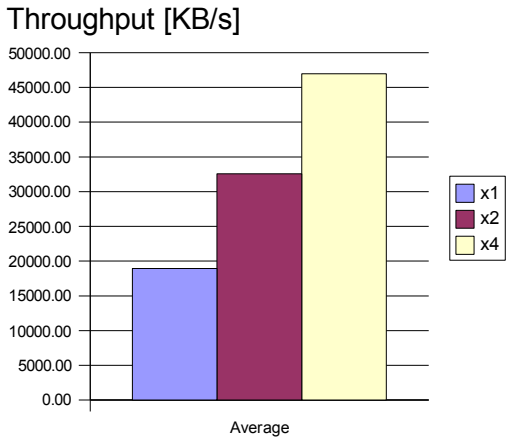
Prije mjerenja, svi parametri su vraćeni na svoje početno, podrazumjevano stanje, koje je zateknuto na testnim računalima prije optimizacije i mjerenja. Sa slika 5.21, 5.22, 5.23 i 5.24 je vidljivo povećanje performansi (propusnosti) dijeljenjem toka i slanjem kroz više transportnih veza. Mjerenja prikazuju performanse običnim prijenosom, te sa dva i četiri paralelno pokrenuta netperf-a (dvije i četiri transportne veze).



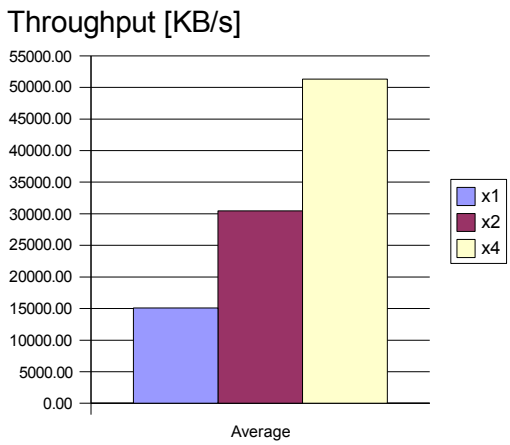
Slika 5.21: TCP Stream - ovisnost propusnosti o multipleksiranju



Slika 5.22: UDP Stream - ovisnost propusnosti o multipleksiranju

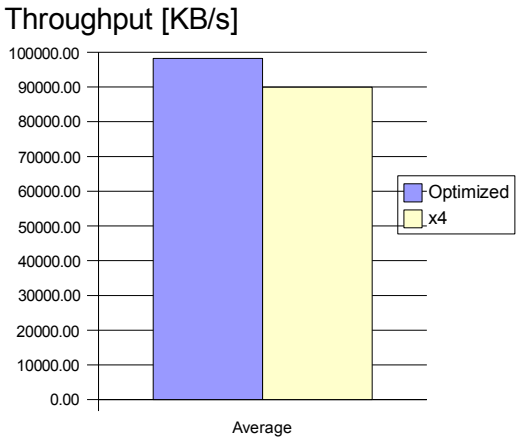


Slika 5.23: TCP RR - ovisnost propusnosti o multipleksiranju

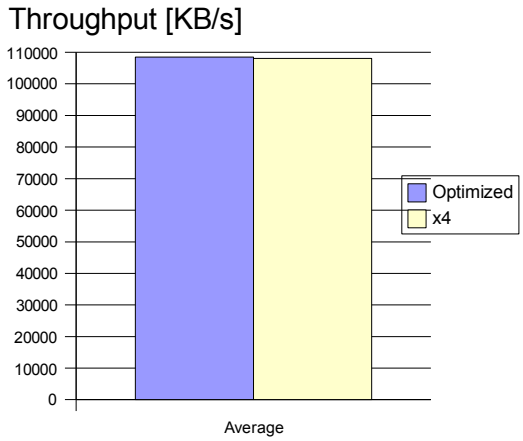


Slika 5.24: UDP RR - ovisnost propusnosti o multipleksiranju

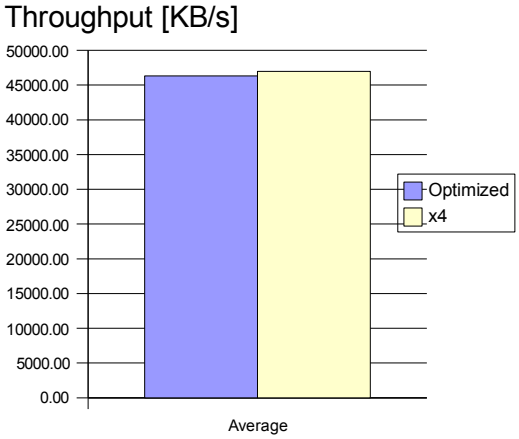
Također, napravljena je usporedba rezultata dobivenih početnim, neoptimiranim parametrima i multipleksiranjem sa rezultatima dobivenim optimalnim parametrima bez multipleksiranja. Na slikama 5.25, 5.26, 5.27 i 5.28 vidimo rezultate te usporedbe. Za primjetiti je sumjerljivost rezultata, dakle, multipleksiranjem se doista može puno postići. Nadalje, vidljivo je da u masovnim prijenosima nešto bolje rezultate daje prijenos optimalnim parametrima, dok je u zahtjev-odziv prijenosima bolje rezultate dao prijenos multipleksiranjem.



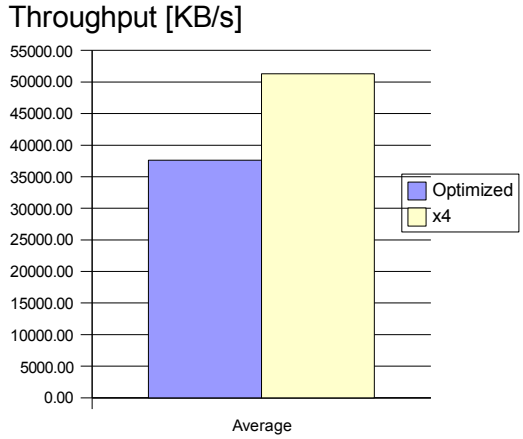
Slika 5.25: TCP Stream - usporedba prijenosa optimalnim parametrima i multipleksiranjem



Slika 5.26: UDP Stream - usporedba prijenosa optimalnim parametrima i multipleksiranjem



Slika 5.27: TCP RR - usporedba prijenosa optimalnim parametrima i multipleksiranjem



Slika 5.28: UDP RR - usporedba prijenosa optimalnim parametrima i multipleksiranjem

6. Zaključak

Isprobane su četiri vrste prijenosa podataka: TCP masovni prijenos, UDP masovni prijenos, TCP zahtjev-odziv prijenos i UDP zahtjev-odziv prijenos. Također, isprobane su i različite kombinacije parametara, te izmjeren njihov utjecaj na propusnost pojedinom vrstom prijenosa. Mjerilo se na testnim računalima lokalno povezanim gigabitnim Ethernet-om preko preklopnika. Možemo izvesti nekoliko zaključaka:

1. Općenito je bolje koristiti masovne prijenose od zahtjev-odziv prijenosa za prijenos velikih količina podataka između lokalno povezanih računala.
2. Općenito, UDP daje bolje rezultate od TCP-a, tj. veća propusnost je postignuta UDP-om. Propusnost je, naravno, mjerena s prijemne strane. Time je uračunato da su neki paketi UDP-om stigli s pogreškom, te ih treba prenijeti ponovo. Stoga se pretežno UDP i koristi u posebno prilagođenim (engl., custom) algoritmima za masovne prijenose. UDP-om je postignuto 88% maksimalno moguće teoretske propusnosti, dok je TCP-om postignuto 80% maksimalne teoretske propusnosti.
3. Multipleksiranje transportnim slojem, tj. otvaranje više TCP/UDP pristupnih točaka, daje odlične rezultate bez ikakve optimizacije. Stoga posebno prilagođeni (custom) algoritmi za masovne prijenose često koriste multipleksirani UDP.
4. TCP parametrima jezgre Linux operacijskog sustava možemo znatno utjecati na propusnost. U tablici 6.1 vidimo njihove optimizirane vrijednosti.

Tablica 6.1: Optimizirani TCP parametri

`tcp_window_scaling`	1	da bi uopće mogli koristiti velike TCP prozore (>64K)
`tcp_mem`	*`LOW PRESSURE HIGH`*	povećati Low, Pressure i High da jezgra operacijskog sustava ima dovoljno memorije, npr. na 1048576

`rmem_default`
`rmem_max`
`wmem_default`
`wmem_max`
	`MAX`	povećati TCP prozore i međuspremnike. High povećati, npr na 33554432

`tcp_rmem` `tcp_wmem`	`4096` *`MAX MAX`*	povećati TCP prozore i međuspremnike. High povećati, npr na 33554432
`tcp_timestamps`	`0`	ne koristiti vremenske oznake (timestamps)

`tcp_sack`
`tcp_fack`
`tcp_dsack`
	1	koristiti selektivne potvrde (i njihove ekstenzije)

`tcp_congestion_control`	`reno`	Reno je dao najbolje rezultate od isprobanih algoritama kontrole zakrčenja
`tcp_low_latency`	0	Preferiraj veću propusnost na uštrb manjeg kašnjenja
`netdev_max_backlog` `txqueuelen`	`10000`	povećaj ulazne i izlazne međuspremnike na sučelju

7. Literatura

1. A. Kuznetsov: *Linux Kernel Documentation :: networking : ip-sysctl*
   URL: http://www.mjmwired.net/kernel/Documentation/networking/ip-sysctl.txt (07/02/2007).
2. V. Jacobson, R. Braden, D. Borman: *RFC 1323 - TCP Extensions for High Performance*
   URL: http://www.ietf.org/rfc/rfc1323.txt (05/1992).
3. V. Jacobson, R. Braden: *RFC 1072 - TCP Extensions for Long-Delay Paths*
   URL: http://www.ietf.org/rfc/rfc1072.txt (10/1988).
4. *Netperf Manual*, Hewlett-Packard Company 1995.
   URL: http://www.netperf.org/netperf/training/Netperf.html.
5. E. He, J. Leigh, O. Yu, T. DeFanti: *Reliable Blast UDP : Predictable High Performance Bulk Data Transfer,* accepted paper, IEEE Cluster Computing 2002, Chicago, Illinois, Sept, 2002.
6. Boumenot, Christopher M., *The Performance of a Linux NFS Implementation*, Magistarski rad, Faculty of the Worcester Polytechnic Institute, 2002.

8. Prilog A

Ispis datoteke (bash skripta) "*netperf_script*" koja služi za automatizaciju mjerenja:

```
#!/bin/sh
#
# Skripta za automatiziranje mjerenja performansi tcp-a i udp-a
# u bulk prijenosu (streaming i request-reply).
# Prvo ispisuje vrijednosti /proc/sys/net/core/* i /proc/sys/net/ipv4/*
# koje bi mogle utjecati na performanse, a koje variramo prije
# pokretanja skripte.
#
# Usage:
# netperf_script
#         - ispituje ip_sysctl varijable bez pokretanja testa.
#
# Usage:
# netperf_script hostname
#         - ispituje ip_sysctl varijable i pokrece grupu netperf testova.
#


# #############
# # PARAMETRI #
# #############

# provjera broja parametara
if [ $# -gt 1 ]; then
          echo "Usage:"
          echo "netperf_script hostname"
          exit 1
fi

echo ""
echo "------------------------------"
echo "Script executed"

case $# in
0)
          echo "Without testing"
          JUST_READ_IPSYSCTL_VARIABLES=1;;
1)
          echo "With testing"
          REMOTE_HOST=$1
          echo "Remote host:" $REMOTE_HOST
          JUST_READ_IPSYSCTL_VARIABLES=0;;
esac

echo "------------------------------"


# #############
# # KONSTANTE #
# #############
```

```
#NETHOME=/usr/local/netperf
NETHOME=.

#PORT="-p some_other_portnum"
PORT=""

# The test length in seconds
TEST_LENGTH=10

# Measuring unit
# G=2^30, M=2^20, K=2^10, g=10^9, m=10^6, k=10^3
MEASURING_UNIT="-f K"

# Verbosity
#VERBOSITY="-v 2"
VERBOSITY="-v 1"

# -i max,min
# Set the maximum and minimum number of iterations when trying to reach
# certain confidence levels.
# -I lvl,[,intvl]
# Specify the confidence level (either 95 or 99 - 99 is the default) and
# the width of the confidence interval as a percentage (default 10)
STATISTICS="-i 10,2 -I 99,5"
#STATISTICS="-i 3,1"


# ##################
# # Grupa testova #
# ##################

# The socket sizes that we will be testing
# The send sizes that we will be using
TCP_STREAM_SOCKET_SIZES="4K 16K 64K 128K 1M"
TCP_STREAM_SEND_SIZES="4K 16K 64K 128K 1M"

UDP_STREAM_SOCKET_SIZES="4K 16K 64K 128K 1M"
UDP_STREAM_SEND_SIZES="4K 16K 64K 128K 1M"

TCP_RR_SOCKET_SIZES="4K 16K 64K 128K 1M"
TCP_RR_SEND_SIZES="4K 16K 64K 128K 1M"

UDP_RR_SOCKET_SIZES="4K 16K 64K 128K 1M"
UDP_RR_SEND_SIZES="4K 16K 64K 128K 1M"


# #############
# # IP_SYSCTL #
# #############

# ip sysctl varijable koje zelimo ispitati
IP_SYSCTL_VARIABLES="core/rmem_default core/rmem_max core/wmem_default
core/wmem_max "
IP_SYSCTL_VARIABLES+="ipv4/tcp_mem ipv4/tcp_rmem ipv4/tcp_wmem "
IP_SYSCTL_VARIABLES+="ipv4/tcp_window_scaling ipv4/tcp_timestamps ipv4/tcp_sack
ipv4/tcp_fack ipv4/tcp_dsack "
IP_SYSCTL_VARIABLES+="ipv4/tcp_congestion_control ipv4/tcp_ecn
ipv4/tcp_low_latency "
```

```
IP_SYSCTL_VARIABLES+="ipv4/ip_no_pmtu_disc ipv4/tcp_mtu_probing ipv4/tcp_base_mss
"
IP_SYSCTL_VARIABLES+="core/netdev_max_backlog ipv4/route/flush"

# petlja ispisuje vrijednosti svih varijabli
echo ""
echo "IP SYSCTL VARIABLES:"
for VARIABLE in $IP_SYSCTL_VARIABLES
  do
          echo /proc/sys/net/$VARIABLE
          cat /proc/sys/net/$VARIABLE
  done
echo ""


# ############
# # ifconfig #
# ############
# ispisuje podatke o suceljima (zanima nas MTU i txqueuelen)
/sbin/ifconfig


#########################################################
# Kraj, ako samo zelimo ispitati ip_sysctl varijable
# bez pokretanja testa.
if [ $JUST_READ_IPSYSCTL_VARIABLES -eq 1 ]; then
          exit 0
fi



# ############
# # MJERENJA #
# ############

#         -ping-
#         ------
echo "1) ping test"
echo "------------"
echo ""

ping -c 5 $REMOTE_HOST

echo ""

#         -netperf-
#         ---------
echo "2) netperf test"
echo "---------------"
echo ""

echo "2.A) TCP_STREAM"
echo "---------------"
echo ""

# Ispisi Header, bez da doista mjeris
$NETHOME/netperf $PORT -H $REMOTE_HOST -l 0 -t TCP_STREAM -v 1 $MEASURING_UNIT
echo
echo ----------------------------------------------------------------
```

```
for SOCKET_SIZE in $TCP_STREAM_SOCKET_SIZES
  do
  for SEND_SIZE in $TCP_STREAM_SEND_SIZES
    do

    # Pokreni mjerenje
    $NETHOME/netperf $PORT -H $REMOTE_HOST -l $TEST_LENGTH -t TCP_STREAM\
    $STATISTICS $VERBOSITY $MEASURING_UNIT -P 0  --\
    -m $SEND_SIZE -M $SEND_SIZE -s $SOCKET_SIZE -S $SOCKET_SIZE

    done
  done

echo ""

echo "2.B) UDP_STREAM"
echo "---------------"
echo ""

# Ispisi Header, bez da doista mjeris
$NETHOME/netperf $PORT -H $REMOTE_HOST -l 1 -t UDP_STREAM -v 1 $MEASURING_UNIT
echo
echo ----------------------------------------------------------------

for SOCKET_SIZE in $UDP_STREAM_SOCKET_SIZES
  do
  for SEND_SIZE in $UDP_STREAM_SEND_SIZES
    do

    # Pokreni mjerenje
    $NETHOME/netperf $PORT -H $REMOTE_HOST -l $TEST_LENGTH -t UDP_STREAM\
    $STATISTICS $VERBOSITY $MEASURING_UNIT -P 0 --\
    -m $SEND_SIZE -M $SEND_SIZE -s $SOCKET_SIZE -S $SOCKET_SIZE

    done
  done

echo "2.C) TCP_RR"
echo "---------------"
echo ""

# Ispisi Header, bez da doista mjeris
$NETHOME/netperf $PORT -H $REMOTE_HOST -l 0 -t TCP_RR -v 1 $MEASURING_UNIT
echo
echo ----------------------------------------------------------------

for SOCKET_SIZE in $TCP_RR_SOCKET_SIZES
  do
  for SEND_SIZE in $TCP_RR_SEND_SIZES
    do

    # Pokreni mjerenje
    $NETHOME/netperf $PORT -H $REMOTE_HOST -l $TEST_LENGTH -t TCP_RR\
    $STATISTICS $VERBOSITY $MEASURING_UNIT -P 0  --\
    -r $SEND_SIZE,$SEND_SIZE -s $SOCKET_SIZE -S $SOCKET_SIZE

    done
  done
```

```
echo ""

echo "2.D) UDP_RR"
echo "---------------"
echo ""

# Ispisi Header, bez da doista mjeris
$NETHOME/netperf $PORT -H $REMOTE_HOST -l 0 -t UDP_RR -v 1 $MEASURING_UNIT
echo
echo ---------------------------------------------------------------------

for SOCKET_SIZE in $UDP_RR_SOCKET_SIZES
  do
  for SEND_SIZE in $UDP_RR_SEND_SIZES
    do

    # Pokreni mjerenje
    $NETHOME/netperf $PORT -H $REMOTE_HOST -l $TEST_LENGTH -t UDP_RR\
    $STATISTICS $VERBOSITY $MEASURING_UNIT -P 0  --\
    -r $SEND_SIZE,$SEND_SIZE -s $SOCKET_SIZE -S $SOCKET_SIZE

    done
  done

echo ""

echo ""
echo "------------------"
echo "Testing completed!"
echo "------------------"
echo ""
```

9. Prilog B

Ispis datoteke (bash skripta) "*ipsysctl_script*" koja služi za automatizaciju mjerenja:

```
#!/bin/sh
#
# Skripta za postavljanje  /proc/sys/net/core/* i /proc/sys/net/ipv4/*
# varijabli (IP SYSCTL varijabli).
#
#

# memory pages
TCP_MEMORY_PAGE=1048576
TCP_MEMORY_PAGES="$TCP_MEMORY_PAGE $TCP_MEMORY_PAGE $TCP_MEMORY_PAGE"

# 1000000 bi trebalo biti sasvim dovoljno
SEND_SOCKET_SIZE=33554432
RECEIVE_SOCKET_SIZE=33554432

# For socket (min, default, max)
SEND_SOCKET="4096 $SEND_SOCKET_SIZE $SEND_SOCKET_SIZE"
RECEIVE_SOCKET="4096 $RECEIVE_SOCKET_SIZE $RECEIVE_SOCKET_SIZE"

USE_WINDOW_SCALING=1
USE_TIMESTAMPS=1
USE_SACK=1
USE_FACK=1
USE_DSACK=1

# westwood, bic, vegas
CONGESTION=bic
CONGESTION_ECN=0
LOW_LATENCY=0

NO_PMTU_DISCOVERY=0
MTU_PROBING=0

BASE_MSS=512

NETDEV_MAX_BACKLOG=1000
TX_QUEUE_LEN=1000
MTU=1500

######################################################################
echo $RECEIVE_SOCKET_SIZE > /proc/sys/net/core/rmem_max
echo $RECEIVE_SOCKET_SIZE > /proc/sys/net/core/rmem_default
echo $SEND_SOCKET_SIZE > /proc/sys/net/core/wmem_max
echo $SEND_SOCKET_SIZE > /proc/sys/net/core/wmem_default

echo $TCP_MEMORY_PAGES > /proc/sys/net/ipv4/tcp_mem
echo $SEND_SOCKET > /proc/sys/net/ipv4/tcp_wmem
echo $RECEIVE_SOCKET > /proc/sys/net/ipv4/tcp_rmem

echo $USE_WINDOW_SCALING > /proc/sys/net/ipv4/tcp_window_scaling
echo $USE_TIMESTAMPS > /proc/sys/net/ipv4/tcp_timestamps
```

```
echo $USE_SACK > /proc/sys/net/ipv4/tcp_sack
echo $USE_FACK > /proc/sys/net/ipv4/tcp_fack
echo $USE_DSACK > /proc/sys/net/ipv4/tcp_dsack

echo $CONGESTION > /proc/sys/net/ipv4/tcp_congestion_control
echo $CONGESTION_ECN > /proc/sys/net/ipv4/tcp_ecn
echo $LOW_LATENCY > /proc/sys/net/ipv4/tcp_low_latency

echo $NO_PMTU_DISCOVERY > /proc/sys/net/ipv4/ip_no_pmtu_disc
echo $MTU_PROBING > /proc/sys/net/ipv4/tcp_mtu_probing

echo $BASE_MSS > /proc/sys/net/ipv4/tcp_base_mss

echo $NETDEV_MAX_BACKLOG > /proc/sys/net/core/netdev_max_backlog

/sbin/ifconfig eth0 txqueuelen $TX_QUEUE_LEN
/sbin/ifconfig eth0 mtu $MTU

echo 1 > /proc/sys/net/ipv4/route/flush
```