Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Marko Košmerl

Merjenje pasovne širine: metrike, metode in orodja

diplomsko delo
na univerzitetnem študiju

Mentor: doc. dr. Mojca Cliglarič

Ljubljana, 2012

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00033/2011
Datum: 01.12.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko ter Fakulteta za matematiko in fiziko izdaja naslednjo nalogo:

Kandidat: **MARKO KOŠMERL**

Naslov: **MERJENJE PASOVNE ŠIRINE: METRIKE, METODE IN ORODJA**
BANDWIDTH MEASUREMENT: METRICS, METHODS AND TOOLS

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Preglejte področje merjenja pasovne širine v računalniških omrežjih. Analizirajte metode, ki se najpogosteje uporabljajo, identificirajte metrike in primerjajte med seboj orodja, ki so na voljo. Izbrano orodje (iperf) podrobneje predstavite in opišite zlasti naprednejše možnosti uporabe in pomanjkljivosti, ki jih opažate za vaš namen uporabe. Zasnujte lastno orodje za merjenje pasovne širine, ki teh pomanjkljivosti nima. Opišite načrt in izvedbo orodja, ter njegovo ustreznost potrdite s praktičnim testiranjem.

Mentor:

M. Ciglarič

doc. dr. Mojca Ciglarič

Dekan Fakultete za računalništvo in informatiko:

prof. dr. Nikolaj Zimic

Dekan Fakultete za matematiko in fiziko:

akad. prof. dr. Franc Forstnarič

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Marko Košmerl, z vpisno številko **63040280**, sem avtor diplomskega dela z naslovom:

Merjenje pasovne širine: metrike, metode in orodja

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Mojce Ciglarič in somentorstvom univ. dipl. ing. Andreja Krevla,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 23. marca 2012 Podpis avtorja:

Zahvala

Zahvaljujem se mentorici doc. dr. Mojci Ciglarič in univ. dipl. ing. Andreju Krevlu za pomoč pri izdelavi diplomskega dela. Posebne zahvale tudi podjetju NIL Podatkovne komunikacije d.o.o., ki mi je v zadnjih študijskih letih omogočilo pridobivanje znanj in izkušenj na področju računalništva in podatkovnih komunikacij.
Zahvaljujem se tudi dragima mami in očetu za vso podporo, finančno pomoč in potrpežljivost pri študiju.

Kazalo

Seznam uporabljenih kratic in simbolov

AES - Advanced Encryption Standard
BIC - Binary Increase Congestion
BTC - Bulk Transfer Capacity
DSCP - Differentiated Services Code Point
GRE - Generic Routing Encapsulation
HTTP - The Hypertext Transfer Protocol
ICMP - Internet Control Message Protocol
IP - Internet Protocol
IPSec - Internet Protocol Security
MSS - Maximum Segment Size
MTU - Maximum Transmission Unit
RTT - Round Trip Time
SACK - Selective Acknowledgment
SHA - Secure Hash Algorithm
SNMP - Simple Network Management Protocol
TCP - Transimission Control Protocol
ToS - Type of Service
TTL - Time To Live
UDP - User Datagram Protocol
WAN - Wide Area Network

Povzetek

V paketnih omrežjih oznaka "pasovna širina" ali "prepustnost" označuje količino podatkov, ki jo lahko omrežje prenese v časovni enoti. Diplomska naloga se ukvarja z metrikami, metodami in orodji, namenjenimi merjenju pasovne širine. Cilj predstavlja izdelava končnega orodja za natančno, zanesljivo in relativno hitro merjenje pasovne širine. Najprej si pogledamo različne in najpogosteje uporabljene metrike, nato pa predstavimo metode, ki tovrstne metrike merijo. Seznanimo se z implementacijami omenjenih metod in predstavimo način merjenja pasovne širine s strani podjetij na trgu.
Podrobno se spoznamo z orodjem iperf in faktorji, ki vplivajo na prepustnost protokola TCP. Orodju iperf smo dodali podporo protokola SNMP in ga s pomočjo knjižnice API uporabili pri končnem sistemu za merjenje prepustnosti in pasovne širine, imenovanem BWMonitor.

Ključne besede:

pasovna širina, razpoložljiva pasovna širina, prepustnost protokola TCP, iperf.

Abstract

In a packet network, the terms ”bandwidth” or ”throughput” characterizes the amount of data that the network can transfer per unit of time. This thesis deals with metrics, methods and tools used in measuring bandwidth. The goal is to build a tool that could be used for measuring bandwidth in accurate, reliable and relatively fast manner. First we take a look at a different and most frequently used metrics and then we present methods that can measure such a metrics. We also present tools that implement those methods and take a look at how some companies measure bandwidth. We describe the iperf tool in detail and explain the factors that have big effects in measuring TCP throughput. Iperf has been extended with SNMP functionality which is now used as a part of the BWMonitor, a tool used for measuring TCP throughput and bandwidth.

Keywords:

bandwidth, available bandwidth, TCP throughput, iperf.

Poglavje 1

Uvod

Dandanes imajo podjetja in organizacije lokalna omrežja različnih poslovalnic med seboj povezana preko omrežij internetnih ponudnikov. Ključne lastnosti, s katerimi lahko opišemo povezavo med poljubnima poslovalnicama, so pasovna širina, latenca, variabilnost latence, dosegljivost in izkoriščenost pasovne širine. Latenca označuje čas, ki je potreben, da prispejo podatki od pošiljatelja do prejemnika, medtem ko variabilnost latence pove kakšna je variabilnost tega časa. Dosegljivost, navadno v odstotkih, pove, koliko časa je bila povezava med lokacijama prekinjena, pasovna širina pa, kakšna bo največja hitrost prenosa podatkov. Preprostemu uporabniku domačega omrežja je najverjetneje najbolj znana pasovna širina, s katero se navadno sreča pri izbiri ponudnika internetnih storitev in jo kasneje občuti pri prenosih podatkov s svetovnega spleta.

1.1 Motivacija

Od leta 2000 je bilo razvitih nemalo orodij za meritev pasovne širine. Izmed vseh razvitih je največ tistih, ki merijo razpoložljivo pasovno širino in večinoma temeljijo na tehniki razpršenosti paketov. Podajmo nekaj primerov in programov [11], kjer bi metode izpolnjevale svoj namen.

Validacija pasovne širine. Tovrsten primer je za nas najbolj pomembna veja uporabe. Pri meritvi, s katero bi opravili omenjeno validacijo, je v ospredju natančnost in zanesljivost, medtem ko ima čas potreben za izvedbo meritve, manjšo težo.

Nastavitev okna protokola TCP. Eden izmed najpomembnejših nerešenih težav pri uporabi protokola TCP je nastaviti začetno hitrost pošiljanja podatkov, ki bi vzdolž povezave zapolnila razpoložljivo pasovno širino

in s tem odpravila potrebo po fazi počasnega zagona. Pri tem je seveda pomembno, da je meritev razpoložljive pasovne širine opravljena v časovnem razponu ene sekunde, sama natančnost pa lahko do določene mere odstopa.

Hitrost prenosa pretakanja podatkov. Avdio in video aplikacije lahko pogosto nastavijo hitrost pošiljanja z uporabo različnih kodnih shem. Metode merjenja razpoložljive pasovne širine bi lahko nadomestile način nastavljanja hitrosti pošiljanja podatkov, ki je sedej precej pogojen z zaznavanjem izgube podatkov.

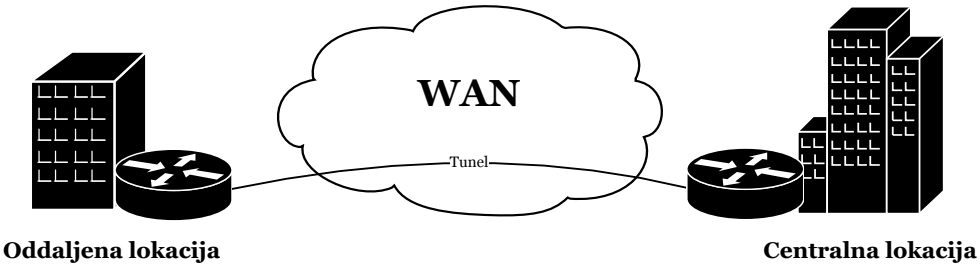
Prekrivna omrežja in multicast. Prekrivna omrežja so postavljena na enga ali več omrežij. Uporabljaljo se lahko tudi pri storitvah, ki temeljijo na prenosu podatkov do večjega števila uporabnikov istočasno, tj. multicast prenos. Obstoječa prekrivna omrežja pri usmerjanju in zagotavljanju kakovosti storitev za mero odločanja uporabljajo latenco. Če bi tovrstno mero izboljšali z metodami merjenja razpoložljive pasovne širine v realnem času, bi usmerjanje in kakovost pri tovrstnih storitvah še dodatno izboljšali.

Izbira zrcalnega strežnika. Koristnost bi se pokazala tudi pri odjemalcih, ki med večjim številom zrcalnih strežnikov poskušajo izbrati tistega, do katerega bi imeli na voljo več razpoložljive pasovne širine.

1.2 Cilj

Cilj diplomske naloge je predstavitev orodja, s pomočjo katerega je mogoče v rangu nekaj sekund kar se da natančno in zanesljivo izmeriti pasovno širino, ki jo imamo sklenjeno s ponudnikom internetnih storitev. Za samo predstavitev takšnega orodja je predpogoj, da se seznanimo z metrikami in metodami merjenja pasovne širine kot jih obravnava raziskovalno področje računalniških komunkacij in ugotovimo, ali bi bile metode za nas lahko uporabne. V precejšnji meri nas zanima tudi, kako takšne meritve kot storitev opravljajo obstoječa podjetja na trgu. S pregledom omenjenega bomo videli, zakaj smo se pri izdelavi ciljnega orodja odločili, da za osnovo vzamemo program iperf in ga kot osnovo vzamemo pri izgradnji končnega orodja, imenovanega BWMonitor.

Kot ciljni primer uporabe nam služi klasična slika podjetja v realnosti 1.1, ki jo navadno sestavljajo centralna lokacija, oddaljena lokacija in med njima tunel IPSec.



Slika 1.1: Topologija

1.3 Zgradba in terminologija

V nadaljevanju ima diplomsko delo naslednjo zgradbo: uvodno poglavje 2 je namenjeno razlagi osnovnih pojmov, povezanih s pasovno širino; podali bomo definicije metrik in opisali metode, ki merijo definirane metrike. V razdelku 2.1 bomo našteli in opisali nekaj javno dostopnih orodij, ki predstavljajo implementacije metod in so za nas potencialno uporabna. Na koncu poglavja si bomo ogledali, kako meritve pasovne širine ponujajo podjetja v tržnem svetu, pokomentirali orodja, s katerimi smo se srečali, in podali oceno predstavljenih metod in orodij. Slednje bo predstavljalo tudi argument naše odločitve izbire programa iperf, ki bo temelj našega končnega orodja BWMonitor.

Poglavje 3 je namenjeno predstavitvi programa iperf. Po splošnem opisu orodja bomo podrobneje spoznali faktorje, ki vplivajo na meritve, na poenostavljenem modelu pogledali dinamiko delovanja protokola TCP in omenili nekaj zanimivih in pomembnih dejstev glede zmogljivosti strojne opreme povprečnega domačega uporabnika. Spoznali bomo tudi zadnji algoritem, ki je v uporabi pri kontroli zamašitve omrežja, razložili problem faze počasnega zagona in predstavili in opisali rešitev. Podali bomo nekaj grafov, s katerimi bomo prikazali, kaj se zgodi s prometom protokola TCP, če povezavo obremenimo s pošiljanjem prometa UDP enake hitrosti, kot je kapaciteta povezave. V drugem delu poglavja bomo v razdelku 3.2 spoznali parametre, ki jih podamo pri uporabi orodja, pomen rezultatov in kako smo z uporabo protokola SNMP dosegli meritve pasovne širine. Poglavje bomo zaključili s primeri in komentarji rezultatov uporabe samega orodja v razdelku 3.3.

Poglavje 4 predstavlja jedro praktičnega dela diplomske naloge in opiše sistem BWMonitor. Na začetku so povzete specifikacije orodja, podrobno je razložen koncept in opis protokola, po katerem strežnik opravi meritev do odjemalca

oddaljene lokacije. V razdelku 4.3 je podrobno opisana podatkovna baza in njen pomen za delovanje orodja. V razdelku 4.4 bomo navedli razloge za izdelavo skripte `bw_manager.pl` in opisali način uporabe le-te. V 4.5 so prikazani grafi meritev znotraj sistema Cacti. Za konec bomo v zadnjem razdelku 4.6 podali oceno orodja in našteli in opisali funkcionalnosti, s katerimi bi BWMonitor naredili še bogatejši. V poglavju 4 sledi zaključek diplomskega dela.

V nadaljevanju bomo uporabljali naslednjo terminologijo. Omrežne naprave kot so stikala, usmerjevalniki ali požarne pregrade, bomo označevali z besedo "vozlišče". Povezavo med dvema poljubnima vozliščema na transportni plasti sklada TCP/IP bomo označevali kot "logična povezava" ali le "povezava", povezava med sosednjima vozliščema na povezovalni plasti bo poimenovana kot "fizična povezava". Fizični odjemalec, ki se priklopi na usmerjevalnik oddaljene lokacije, do katere se izvaja meritev, bomo poimenovali z TC (ang. "Thin Client"), sam usmerjevalnik oddaljene lokacije pa z R_{TC}. Vmesnik, ki na usmerjevalniku R_{TC} pelje v smeri ponudnika internetnih storitev, bo označen z $ifce_{WAN}$. Prepustnost protokola TCP in UDP bo označena z $perf\text{-}TCP$ in $perf\text{-}UDP$, proti oddaljeni lokaciji jo bomo označili z $perf\text{-}TCP_{down}$ ter $perf\text{-}UDP_{down}$, v obratno smer pa $perf\text{-}TCP_{up}$ ter $perf\text{-}UDP_{up}$. Pasovna širina bo v splošnem označena z BW, pasovna širina po protokolu TCP in UDP bo splošno označena $BW\text{-}TCP$ in $BW\text{-}UDP$, analogno kot pri prepustnosti še $BW\text{-}TCP_{down}$, $BW\text{-}TCP_{up}$ ter $BW\text{-}UDP_{down}$ in $BW\text{-}UDP_{up}$.

Poglavje 2

Pasovna širina

Pasovna širina je v analognih računalniških sistemih definirana kot frekvenčni razpon v herzih, ki se uporablja za prenos podatkov, medtem ko je v digitalnih komunikacijskih sistemih definirana kot količina prenesenih podatkov, merjena v bitih na sekundo (b/s, angl. bps). Današnja literatura uporablja izraz pasovna širina za tisto, kar je v [16] definirano in razloženo kot kapaciteta. V nadaljevanju bosta besedi kapaciteta in pasovna širina označevali isto, tj. količino prenesenih podatkov. Ker se podatki v računalniških omrežjih prenašajo po dogovorjenih standardih ali protokolih, je potrebno poudariti, da je pomen definicije kapacitete ali pasovne širine smiseln le na nekem danem protokolu ali protokolarni plasti.

2.1 Metrike, metode in orodja

2.1.1 Metrike

Pasovna širina ali kapaciteta (ang. capacity) [17, 16, 14]. Definicija metrike se nahaja na plasteh *L2* in *L3*. Na plasti *L2* je kapacita definirana s hitrostjo fizične povezave ali segmenta. To je lahko 100 *Mbps* pri segmentu *100BASE-TX* ali 1000 *Mbps* pri segmentu *1000BASE-X* protokola Ethernet.

Na *L3* plasti je kapaciteta vozlišča nižja zaradi dodatne režije protokola na plasti *L2*. Recimo, da je C_{L2} kapaciteta in H_{L2} velikost protokularne enote segmenta *L2*. Čas prenosa podatkovne enote plasti *L3* je torej

$$\Delta_{L3} = \frac{L_{L3} + H_{L2}}{C_{L2}} \ ,$$

kapaciteta C_{L3} v odvisnosti od L_{L3} in H_{L2} pa ima naslednjo obliko

$$C_{L3} = \frac{L_{L3}}{\Delta_{L3}} = \frac{L_{L3}}{\frac{L_{L3}+H_{L2}}{C_{L2}}} = C_{L2}\frac{1}{1+\frac{H_{L2}}{L_{L3}}} \ .$$

Če bi imeli $L_{L3} = 100\,\text{bajtov}$ in $H_{L2} = 42\,\text{bajtov}$, bi veljalo $C_{L3} = 7.04\,Mbps$, pri $L_{L3} = 1500\,\text{bajtov}$ in istem H_{L2} pa $C_{L3} = 9.73\,Mbps$. Kapaciteto C_i vozlišča i definiramo kot tisto, pri kateri je L_{L3} največji mogoč, kapaciteta omrežne povezave C je posledično določena z minimalno kapaciteto povezave na plasti $L3$ vzdolž povezave

$$C = \min_{i=1,\ldots,H} C_i \ .$$

Fizična povezava z najmanjšo kapaciteto vzdolž povezave se imenuje najožja fizična povezava.

Razpoložljiva pasovna širina (ang. available bandwidth) [17, 11, 8]. Metrika se navezuje na neuporabljen del kapacitete povezave v nekem časovnem intervalu. V največji meri je odvisna od obremenitve povezave zaradi česar je časovno variabilna metrika. V nekem trenutku, se na povezavi prenašajo paketi s polno kapaciteto povezave ali pa je povezava nedejavna. Lahko rečemo, da je uporaba povezave v nekem trenutku bodisi 0 ali 1. Povprečna uporaba povezave $\overline{u}(t-\tau,t)$ za časovno periodo $(t-\tau,t)$ je potem

$$\overline{u}(t-\tau,t) = \frac{1}{\tau}\int_{t-\tau}^{t} u(x)\ dx \ ,$$

kjer je $u(x)$ razpoložljiva pasovna širina v časovnem trenutku x. Če je C_i kapaciteta vozlišča i in u_i povprečna obremenitev vozlišča v danem časovnem intervalu, je povprečna razpoložljiva pasovna širina A_i vozlišča i dana z neuporabljenim delom kapacitete

$$A_i = (1-u_i)C_i \ .$$

Razpoložljiva pasovna širina na povezavi vzdolž H vozlišč je torej minimalna razpoložljiva pasovna šrina izmed vseh H vozlišč,

$$A = \min_{i=1,\ldots,H} A_i \ .$$

Fizična povezava z najmanjšo razpoložljivo pasovno širino vzdolž povezave se imenuje najtesnejša fizična povezava.

TCP-prepustnost in BTC [3, 14]. TCP je daleč najbolj uporabljen protokol plasti *L4*. Posledično je TCP-prepustnost ena izmed ključnih metrik pasovne širine. BTC (ang. Bulk-Transfer-Capacity) metrika je definirana kot največja dosegljiva prepustnost v eni seji protokola TCP. Intuitivno definicija označuje dolgoročno pričakovano povprečno prepustnost idealne implementacije povezovalnega protokola s podporo kontrole zamašitve omrežja. Recimo, da je povezava kapacitete C zamašena s prometom ene TCP-seje. Razpoložljiva pasovna širina bo tedaj 0, medtem ko bi bil BTC druge seje približno enak $C/2$.

2.1.2 Metode

Preden podamo in opišemo metode merjenja različnih metrik pasovne širine, razložimo pomen polja TTL in časa RTT. TTL (ang. Time to live) protokola IP predstavlja mehanizem, s katerim dosežemo, da paket protokola IP ob morebitni zanki v omrežju ne potuje po zanki v nedogled. Navadno polje predstavlja števec, ki se pri potovanju paketa zmanjša za 1 pri vsakem usmerjevalniku, ki ga prečka. V kolikor polje TTL doseže vrednost 0, preden pride do cilja, se paket s strani usmerjevalnika zavrže, pošiljatelju pa se pošlje sporočilo o napaki z uporabo protokola ICMP.
Čas RTT (ang. Round-trip Delay Time) je čas, potreben, da segment protokola TCP prispe do ciljnega vozlišča, plus čas, potreben, da pošiljatelj prejme potrditev o prejemu paketa.

Pošiljanje spremenljive velikosti paketov

Metoda pošiljanja paketov spremenljive velikosti (ang. variable packet size probing) [17] je namenjena meritvi kapacitete vsakega vozlišča vzdolž poti. Ideja metode je s pomočjo polja TTL izmeriti RTT od izvornega vozlišča do vsakega vozlišča kot funkcijo velikosti paketov. RTT do vsakega vozlišča sestavljajo tri časovne komponente: zakasnitve serializacije (ang. serialization delays), zakasnitve razširjanja (ang. propagation delays) in čakalne zakasnitve (ang. queueing delays). Zakasnitev serializacije paketa velikosti L na segmentu kapacitete C je čas, potreben, da spravimo paket na segment, tj. L/C. Zakasnitev razširjanja je čas, ki ga potrebuje bit paketa, da prečka segment, čakalna zakasnitev pa se pojavi na usmerjevalnikih ali stikalih, če so čakalne vrste neprazne. Metoda predpostavi, da pri pošiljanju paketov dane velikosti do vseh vozlišč, vsaj en paket in pripadajoči ICMP-odgovor, nimata čakalnih zakasnitev in bo minimalni izmerjen RTT sestavljen iz zakasnitev razširjanja ter zakasnitev serializacije. Minimalen pričakovan RTT v odvisnosti velikosti

paketa L do vozlišča i vzdolž poti je

$$T_i(L) = \alpha + \sum_{k=1}^{i} \frac{L}{C_k} = \alpha + \beta_i L \ ,$$

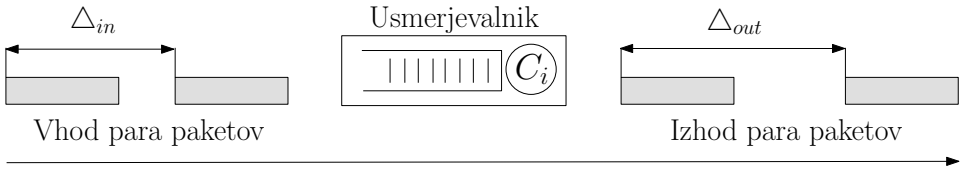
kjer je C_k kapaciteta fizične povezave do k-tega vozlišča, α zakasnitev do vozlišča i, neodvisna od velikosti L, in β_i tisti najmanjši RTT, ki ga dobimo pri meritvi do vozlišča i. Pri zaporednem izvajanju meritev in iskanju najmanjšega RTT za vsako vozlišče $i = 1, \ldots, H$, je ocena kapacitete vozlišča i naslednja:

$$C_i = \frac{1}{\beta_i - \beta_{i-1}} \ .$$

Slaba stran metode se pojavi, če imamo vzdolž povezave stikala, saj pride na njih ravno tako do zakasnitve serializacije časa L/C pa vendarle od njih ne dobimo odgovorov po protokolu ICMP.

Razpršenost parov in serije paketov

Razpršenost parov ali serije paketov [5] je metoda za merjenje kapacitete povezave med dvema vozliščema. Izvorno vozlišče pošlje par paketov prejemniku. Vsak par tvorita paketa enake velikosti. Razpršenost para paketov na določeni povezavi poti je časovna razlika med zadnjima bitoma paketov. Slika 2.1 prikazuje razpršenost para paketov pred in po tem, ko paketa prečkata vozlišče kapacitete C_i pri čemer drži predpostavka, da je na povezavi prisoten le naš promet.



Slika 2.1: Razpršenost para paketov

Če se povezava začne na segmentu kapacitete C_0 in je velikost paketov L, je razpršenost para paketov na prvem segmentu $\Delta_0 = L/C_0$. V splošnem je razpršenost para paketov na segmnetu kapacitete C_i enaka Δ_{in}, razpršenost po vstopu v nov segment pa

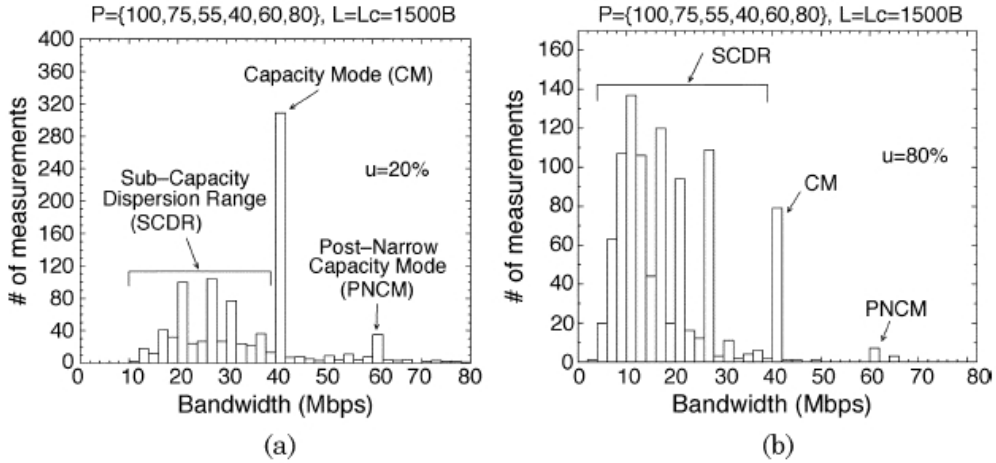
$$\Delta_{\text{out}} = \max(\Delta_{\text{in}}, \frac{L}{C_i}) \ .$$

Razpršenost paketov, ki jo bo prejemnik izmeril pri sprejemu paketov, je

$$\Delta_R = \max_{i=0,\ldots,H}(\frac{L}{C_i}) = \frac{L}{\min_{i=0,\ldots,H}(C_i)} = \frac{L}{C} \ ,$$

kjer je C kapaciteta omrežne povezave. Prejemnik lahko torej oceni kapaciteto povezave kot $C = L/\Delta_R$. Seveda se moramo zavedati, da je predpostavka, da je na omrežni povezavi prisoten le naš promet, vse prej kot realistična. Promet, ki se na povezavi nahaja poleg našega, ima velik vpliv na našo oceno. Kapaciteto C povezave lahko podcenimo, če se med paroma paketov najde ostali promet, ki posledično dvigne razpršenost in s tem zniža našo oceno C. Lahko se zgodi, da je vpliv ostalega prometa takšen, da povzroči večjo zakasnitev prvega paketa kot drugega in smo s tem precenili C.

Metoda oceni porazdelitev pasovne širine iz velikega števila parov paketov, nato pa rezultate meritev razdeli na lokalna območja. Statistično metodo, ki na vzorčnih rezultatih poišče lokalna območja si lahko bralec prebere v dodatku članka [5]; povzemimo le na kratko, da predstavlja lokalno območje razpon, v katerem se nahaja lokalni maksimum vzorčne porazdelitve. Moč lokalnega območja predstavlja število meritev, ki je v bližnji okolici lokalnega maksimuma, medtem ko je širina lokalnega območja dejanska širina obsega. Globalno območje predstavlja lokalno območje z največjo močjo.



Slika 2.2: Histogram meritev pri (a) 20 % in (b) 80 % izkoriščeni povezavi [5]

Slika 2.3 prikazuje histogram porazdelitve meritev za povezavo $P = \{100, 75, 55, 40, 60, 80\}$. Najožja fizična povezava je torej 40 *Mbps* . Ko je obremenjenost povezave majhna, tj. 20 % izkoriščenosti kapacitete povezave, globalno območje sovpada z območjem kapacitete. Meritve, pri katerih se je med pari paketov našel ostali promet in povečal razpršenost, se nahajajo v območjih podcenjene razpršenosti kapacitete (ang. sub-capacity dispersion range), medtem ko je pri meritvah levo od območja kapacitete prišlo do zakasnitve prvega od parov paketov, zaradi česar se je razpršenost zmanjšala in posledično ocena kapacitete

povečala. Slednja območja imenujemo predel precenitve razpršenosti kapacitete (ang. post-narrow capacity mode). Na povezavah visoke obremenitve je verjetnost, da se ostali promet pojavi med parom paketov še večja in vidimo, da območje kapacitete povezave ne predstavlja več globalnega območja. Pogosto se tudi ne pojavi kot lokalno območje.
Ključna točka, ki jo lahko razberemo iz opisanega, je dejstvo, da kapaciteta povezave ne more biti ocenjena kot najpogostejša meritev ali najpogostejše območje.

Posplošitev metode pošiljanja parov paketov predstavlja pošiljanje serije paketov. Naj δ^k označuje razpršenost paketa k in $k+1$ serije paketov dolžine N. Razpršenost serije paketov predstavlja $\Delta(N) = \sum_{k=1}^{N-1} \delta^k$. Ocena kapacitete povezave je enaka:

$$b(N) = \frac{(N-1)L}{\Delta(N)} \ .$$

Ker velja, da je pri daljši seriji paketov večja verjetnost, da pride do motenj ostalega prometa, kar ima za posledico manjšo oceno kapacitete povezave od dejanske, se izkaže, da ko se N poveča, se moč območja kapacitete in območja desno od le-tega zmanjšajo, dokler ne izginejo in prevladajo območja znotraj sub-capacity dispersion range.

Metrika ADR, ki pri zaporedju serije paketov pripada popvrečni vrednosti razpršenosti serij paketov $E[\Delta(N)]$, se imenuje povprečna stopnja razpršenosti:

$$ADR = \frac{(N-1)L}{E[\Delta(N)]} \ .$$

Trditev 1. *ADR je spodnja meja kapacitete in zgornja meja razpoložljive pasovne širine povezave.*

Dokaza trditve 1, s katero se bralec najverjetneje intuitivno strinja, na tem mestu ne bomo podali, nahaja se v [5].

Samodejni periodični tokovi in serija parov paketov

Metoda samodejni periodični tokovi ali krajše SLoPS [11, 17] (ang. self-loading periodic streams) in metoda serija parov paketov, krajše TOPP (ang. trains of packet pairs) [5] sta namenjeni merjenju razpoložljive pasovne širine.

Razložimo idejo metode SLoPS: pošiljatelj periodično pošlje tok K paketov

velikosti L do prejemnika s hitrostjo R_0 vsakih $T = L/R_0$ časovnih enot. Enosmerna zakasnitev (OWD) D^k od pošiljatelja do prejemnika paketa k je

$$D^k = \sum_{i=1}^{H}(\frac{L}{C_i} + \frac{q_i^k}{C_i}) = \sum_{i=1}^{H}(\frac{L}{C_i} + d_i^k) \ ,$$

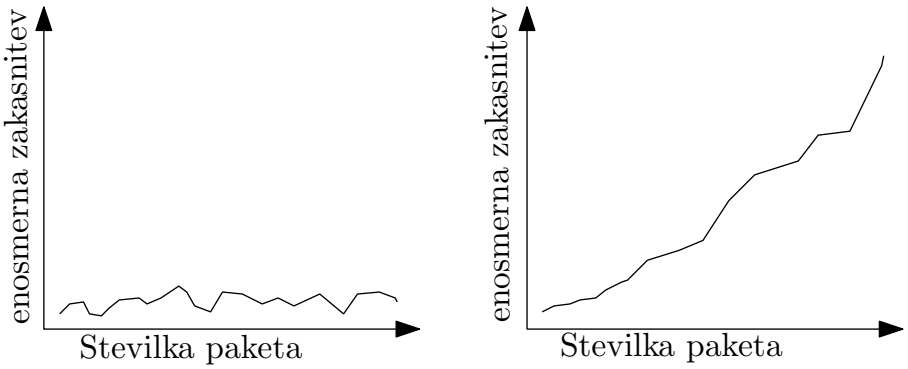
kjer je q_i^k velikost čakalne vrste na povezavi i po sprejemu paketa k in d_i^k čakalna zakasnitev paketa k na povezavi i. OWD-razlika med zaporednima paketoma k in $k+1$ je

$$\Delta D^k = D^{k+1} - D^k = \sum_{i=1}^{H} \frac{\Delta q_i^k}{C_i} = \sum_{i=1}^{H} \Delta d_i^k \ ,$$

kjer je $\Delta q_i^k = q_i^{k+1} - k_i^k$ in $\Delta d_i^k = \Delta q_i^k / C_i$.

Trditev 2. *Če je $R_0 > A$, velja, da je $\Delta D^k > 0$ za $k = 1, \ldots, K-1$. Nasprotno, če je $R_0 \leq A$, velja, da je $\Delta D^k = 0$ za $k = 1, \ldots, K-1$.*

Dokaz trditve 2 bomo ravno tako izpustili in si ga lahko bralec ogleda v [11].



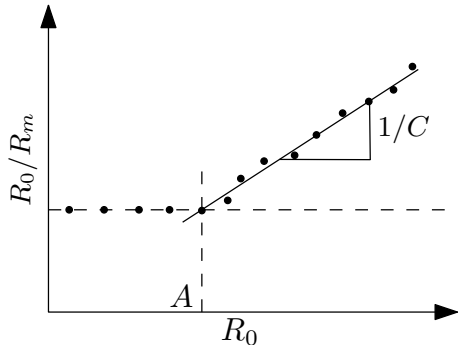
Slika 2.3: SLoPS

Opišimo sedaj iterativni algoritem, ki temelji na trditvi 2. Recimo, da pošiljatelj v iteraciji n pošlje tok paketov s hitrostjo $R(n)$. V skladu s trditvijo 2 prejemnik določi, ali je $R(n) > A$ ali ne, in obvesti prejemnika o tem, kakšna je relacija med A in $R(n)$. Če je $R(n) > A$, pošiljatelj pošlje naslednji tok paketov $n+1$ s hitrostjo $R(n+1) < R(n)$, v nasprotnem primeru pa s hitrostjo $R(n+1) > R(n)$. $R(n+1)$ se izračuna na naslednji način:

$$\begin{aligned} &\text{če } R(n) > A: \quad R^{\max} = R(n); \\ &\text{če } R(n) \leq A: \quad R^{\min} = R(n); \\ &R(n+1) = (R^{\max} + R^{\min})/2. \end{aligned}$$

$R^{\min}$ in $R^{\max}$ sta spodnja in zgornja meja za A po poslanem n-tem toku paketov. Na začetku je $R^{\min} = 0$ in $R^{\max} = R_0^{\max} > A$. Algoritem se konča, ko velja $R^{\max} - R^{\min} \leq \omega$, kjer je ω ocenitveni interval podan kot vhod s strani uporabnika. Če A skozi čas ni spremenljiv, bo prejšnji algoritem poiskal interval $[R^{\min}, R^{\max}]$, ki vsebuje A po $\lceil \log_2(R_0^{\max}/\omega) \rceil$ iteracijah.

Metoda TOPP analogno prvi metodi pošlje serijo parov paketov in nato opazuje zakasnitve med pari. Večina razlik v primerjavi z metodo SLoPS predstavlja statistična obdelava meritev. Razlika je tudi v spreminjanju hitrosti pošiljanja parov paketov, ki je linearna, ter v dejstvu, da lahko metoda TOPP poleg razpoložljive pasovne širine povezave oceni tudi njeno kapaciteto.



Slika 2.4: Razmerje med R_0 in R_m

Za ilustracijo poglejmo pot, pri kateri imamo le eno fizično povezavo kapacitete C, povprečno razpoložljivo pasovno širino A in povprečno hitrost pošiljanja ostalega prometa $R_c = C - A$. TOPP pošlje pare paketov z linearno naraščajočo hitrostjo R_0. Ko postane R_0 večji od A, bo izmerjena hitrost pošiljanja para paketov R_m pri prejemniku naslednja:

$$R_m = \frac{R_0}{R_0 + R_c} C \ .$$

Enakovreden zapis enačbe:

$$\frac{R_0}{R_m} = \frac{R_0 + R_c}{C} \ .$$

TOPP oceni največjo razpoložljivo pasovno širino A z največjim R_0, pri katerem velja $R_0 \approx R_m$.

2.1.3 Orodja

Naštejmo in na kratko opišimo orodja za meritev kapacitete povezave, ki so javno na voljo in uporabljajo metode, ki smo jih predhodno na kratko opisali.

Bprobe [**17**] uporablja metodo razpršenosti parov paketov, pri meritvah pa uporablja protokol ICMP. Bprobe-meritev je sestavljena iz 7 serij vrste paketov dolžine 10, pri čemer ima vsaka naslednja serija večje pakete od prejšnje. V vseh vrstah se izračuna po 9 razpršenosti parov paketov in teh 7 množic izračunov se nato poda fazi filtriranja. Faza filtriranja vsako množico razdeli v intervale, pogleda preseke izračunov vseh množic, nato pa še unijo intervalov. Srednjo vrednost intervala se nato vrne kot oceno kapacitete povezave.

Nettimer [**17**] je orodje, ki oceni kapaciteto povezave s pošiljanjem parov paketov, pri tem pa definira funkcijo gostote, na podlagi katere da večjo težo dobrim vzorcem in izloči vzorce, pri katerih vrednosti zaradi vpliva ostalega prometa niso merodajne.

Pathrate [**17, 5**] za oceno kapacitete uporabi pare datagramov kot tudi serijo datagramov protokola UDP. Za uporabo je potrebno orodje pognati v vlogi prejemnika in v vlogi pošiljatelja. Meritev sestavljajo 3 faze. Cilj prve faze je odkriti največjo dolžino serije paketov, ki jo je povezava še zmožna prenesti brez težav, preden zapolnimo medpomnilnike omrežnih vozlišč in s tem povzročimo izgubo paketov. Za tem pathrate začne s pošiljanjem serij paketov naraščajoče dolžine in preveri, ali vzdolž povezave ni ostalega prometa in bo posledično kapaciteto lažje oceniti. V kolikor je temu tako, pathrate poda končno oceno in zaključi z izvajanjem meritve. V drugi fazi pathrate generira veliko število (1000) parov paketov različne velikosti ter izračuna statistično porazdeljenost meritev. V zadnji fazi pathrate generira dolge serije paketov in izračuna njihove pripadajoče razpršenosti. Na podlagi meritev v zadnjih dveh fazah, pathrate nato poda razpon kapacitete, ki naj bi jo povezava imela.

Sprobe [**17**] za oceno kapacitete povezave uporablja protokol TCP. Pri meritvi BW_{down} prvemu, TCP SYN, paketu, ki je namenjen vzpostavitvi povezave, pripnemo naključne podatke velikosti 1460 bajtov in ga pošljemo na ciljno napravo. Ciljna naprava, v kolikor so ciljna vrata zaprta, odgovori z TCP RST-paketom, kar pošiljatelju omogoča izračunati razpršenost parov paketov. Pri meritvi se uporabi 6 TCP SYN paketov izmed katerih sta srednja dva velikosti 1460, ostali pa 40 bajtov.
Pri meritvi BW_{up} je potrebno uporabiti uspešno TCP povezavo. Lahko se uporabi povezava na spletni strežnik, na katerega se pošlje poljubno poizvedbo. Strežnik nato pošlje toliko segmentov, kot je začetna vrednost TCP okna. Prejemnik izračuna razpršenost parov prejetih paketov in na podlagi tega oceni kapaciteto BW_{up}.

2.1.4 Meritve pasovne širine kot storitev

V podrazdelku, ki je pred nami, bomo na hitro spoznali dve podjetji, ki med drugim, ponujata tudi meritve pasovne širine bodisi kot storitev bodisi kot produkt.

Ookla

Ookla [18, 1] je vodilno svetovno podjetje pri izvajanjih meritev hitrosti širokopasovnega dostopa do interneta. Metode in načini merjenja so bile sprejete s strani večine svetovnih ponudnikov internetnih storitev, dnevno pa naj bi programsko opremo, ki je namenjena meritvam in jo je s plačilom licence mogoče gostovati tudi na lastnem strežniku, uporabljalo več kot tri milijone ljudi.

Podjetje meritve ponuja preko spleta na strani `http://www.speedtest.net/`. Uporabnik ima na izbiro seznam testnih strežnikov, ki se nahajajo na različnih lokacijah po svetu in jih je pri meritvi mogoče izbrati. Navadno se izbere najbližji strežnik in do tega se nato opravi meritev pasovne širine z uporabo protokola HTTP. Poleg pasovne širine je mogoče opraviti tudi test izgube paketov po protokolu UDP s pomočjo aplikacije, napisane v programskem jeziku java. V nadaljevanju je opisano, kako se izvede meritev.

Meritev BW_{down} se izvede na naslednji način:

- iz strežnika se prenese majhno binarno datoteko s katero se naredi grobo oceno BW_{down};
- na podlagi ocene se izbere eno izmed binarnih datotek različnih velikosti;
- izvede se test s preprečevanjem uporabe medpomnenja (ang. caching);
- za test se lahko uporabi do 8 vzporednih HTTP niti;
- podatke se prejme do 30-krat na sekundo;
- podatke se združi v 20 rezin (ena rezina predstavlja 5 %);
- najhitrejših 10 % in najpočasnejših 30 % rezin se nato zavrže;
- povprečje ostalih rezin se uporabi za rezultat ocene BW_{down}.

Meritev BW_{up} se izvede na naslednji način:

- na strani odjemalca se generira majhna kočina naključnih podatkov in pošlje do spletnega strežnika za grobo oceno hitrosti;

- na podlagi ocene se izbere eno izmed binarnih datotek različnih velikosti;
- s pošiljanjem podatkov se izvede meritev;
- za test se lahko uporabi do 8 vzporednih HTTP niti;
- testne podatke se uredi po velikosti, povprečje najhitrejše prenešene polovice podatkov se uporabi za rezultat meritve.

Tehnične razlage, zakaj se pri meritvah uporabi povprečje 60 % ali 50 % določenih podatkov ni moč najti, lahko sklepamo, da se povezava skriva v sami dinamiki protokola TCP. Vir [18] navaja, da je so v podjetju Ookla 60 % in 50 % določili na podlagi testiranj. Tehnično gledano velja, da rezultat meritev ne predstavlja pasovne širine ampak meritev prepustnosti protokola HTTP, ki na transportni plasti uporablja protokol TCP.

Ixia

Podjetje Ixia je vodilno svetovno podjetje na področju izvajanja meritev zmogljivosti storitev in programov računalniških omrežij. Njihova programska oprema IxChariot je uporabljena v več kot 5000 IT organizacijah in podjetjih, med katerimi je nemalo svetovno znanih. Ključna elementa programske opreme sta konzola in končna točka (ang. end-point), ki meritve opravi s pomočjo skript. Končne točke namestimo na računalniški sistem v omrežju centralne in oddaljenih lokacij, medtem ko je konzola nameščena le na centralni lokaciji. Test meritve je definiran s pari končnih točk, izbrano kombinacijo skript, ki posnemajo promet določenih programov ali storitev (Oracle, VoIP, elektronska pošta, ...), in časi, ki določajo, kdaj se meritev začne ter koliko časa traja. Na ta način lahko izmerimo, kako se bodo posamezen aplikacije v omrežju obnašale, preverimo ali implementacija QoS storitve pripomore k boljšemu delovanju določenih storitev in programov ter v končni fazi tudi preverimo, ali meritve odražajo kapaciteto pasovne širine, ki jo imamo sklenjeno s ponudnikom internetnih storitev.

2.2 Ocena obstoječih metod, orodij in storitev

Orodja, ki smo jih opisali v 2.1.3, so se izkazala za neuporabna, saj smo pri testiranju le-teh naleteli na nemalo težav že pri sami prevedbi programa v strojno kodo. Bprobe, Sprobe in Nettimer so bili izdelani na starih verzijah jedra Linux in bi jih bilo potrebno prej popraviti, da bi jih bilo mogoče uporabiti tudi na novih verzijah. Ker ni bilo mogoče najti virov, ki bi govorili o

uporabnosti omenjenih orodij, in ker smo pri spoznanju samih metod videli, da ob prisotnosti ostalega prometa meritve vsebujejo nezanemarljive napake, se za prevedbo izvorne kode na novo verzijo jedra Linux nismo odločili.
Orodje pathrate je odražalo dejstva, ki smo jih spoznali v 2.1.2 in je v večini primerov podalo manjšo oceno kapacitete povezave od dejanske.

Pri študiju izbrane teme smo dobili tudi idejo, da bi v času meritve izmerili razpoložljivo pasovno širino in hitrost pošiljanja ostalega prometa s pomočjo protokola SNMP. Ocena kapacitete bi torej predstavljala vsoto omenjenih meritev. Natančnost meritev bi bila torej odvisna od natančnosti metod, ki merijo razpoložljivo pasovno širino. Članek [8] podaja oceno uporabnosti metod in nekaterih orodij za merjenje razpoložljive pasovne širine gledano po naslednjih kriterijih: natančnost, zanesljivost, čas in količina generiranega prometa, potrebnega za meritev. Meritve so bile opravljene na kontrolnem omrežju, sestavljenem iz štirih strežnikov v funkciji usmerjevalnikov in dveh končnih strežnikov, do katerih so potekale meritve. Testi so bili izvedeni z različno nastavitvijo parametrov, ki navadno opisujejo omrežje (kapaciteta fizičnih povezav, stopnja izgube paketov, velikost medpomnilnikov usmerjevalnikov, zakasnitev paketov), s čimer se je pokazal vpliv parametra na samo meritev. Bralec, ki ga rezultati in grafi omenjenih testov posebej zanimajo, si le-te lahko ogleda v samem članku, podajmo le ugotovitev, ki je v samem viru za nas najbolj pomembna. Napaka meritev vseh orodij je, podobno kot smo videli pri metodah za merjenje kapacitete, odvisna od količine ostalega prometa in je navadno, v kolikor je na povezavi le-tega dosti, precenjena.

Nenatančnost predstavljenih metod in orodij ter način merjenja pasovne širine kot ga izvajata podjetji Ookla in Ixia 2.1.4 nam tako predstavljajo zadosten argument, da meritev pasovne širine izvedemo na podoben način.
Ker je orodij za merjenje prepustnosti protokola TCP kar nekaj (glej 2.1.3), smo se odločili, da pri izdelavi našega končnega sistema, uporabimo orodje iperf verzije 3. Dejstvo je, da je iperf najpogosteje uporabljeno orodje in predstavlja celo neuraden standard pri meritvah prepustnosti protokola TCP.

Poglavje 3

Iperf

Aplikacija, ki uporablja transportno plast sklada TCP/IP, lahko uporabi protokola TCP ali UDP, odvisno od samega namena in potreb aplikacije. V splošnem: če aplikacija potrebuje zanesljiv prenos podatkov, uporablja protokol TCP, medtem ko je uporaba protokola UDP mogoča, če lahko toleriramo nekaj izgube podatkov. Protokol TCP je na račun zanesljivosti počasnejši od protokola UDP, medtem ko je protokol UDP hitrejši in ga navadno uporabimo pri aplikacijah, ki potrebujejo prenose podatkov v realnem času (IP telefonija in IP televizija). Iperf nam omogoča, da izmerimo zmogljivost in delovanje enega in drugega protokola in s tem bodisi vidimo kakšne zmogljivosti lahko pričakujemo iz strani uporabljenih aplikacij bodisi ugotovimo težave, ki jih imamo pri uporabi aplikacij.

3.1 Predstavitev orodja

Orodje iperf so izdelali v laboratoriju NLANR/DAST (ang. "National Laboratory for Applied Network Research, Distributed Applications Support") in deluje kot običajna TCP storitev — na eni strani povezave je strežnik, na drugi odjemalec. Strežnik je v vlogi prejemnika prometa, odjemalec pa generira in pošilja promet. Iperf izmeri količino prenešenih podatkov in na podlagi znanega trajanja meritve izračuna hitrost prenosa podatkov po protokolu TCP/IP.

Deluje lahko tudi kot generator in prejemnik UDP prometa. Odjemalec pošilja promet po protokolu UDP z neko predpisano hitrostjo, strežnik pa ta promet sprejema. Oba spremljata poslane in prejete UDP pakete in ugotavljata morebitno izgubo paketov (ang. "packet loss"), sposobna sta zaznati tudi spremembo v zaporedju testnih paketov (ang. "packet reordering").
S postopnim povečevanjem hitrosti lahko ugotovimo mejno prepustnost lokal-

nega omrežja, pod katero še ni izgub paketov.

Orodje uporabimo tako, da na oddaljeni lokaciji najprej poženemo iperf v vlogi strežnika 3.1,

```
root:/ # iperf3 -s
```

Izpis 3.1: Iperf strežnik

takoj za tem pa lokalno še v vlogi odjemalca 3.2.

```
root:/ # iperf3 -c 192.168.10.3
```

Izpis 3.2: Iperf odjemalec

3.1.1 Faktorji, ki vplivajo na meritev

Faktorji, ki vplivajo na *perf-TCP* so velikost prenosa podatkov, tip in količina ostalega prometa (UDP ali TCP), število TCP povezav, velikost izravnalnega pomnilnika na strani pošiljatelja in prejemnika, uporabljen algoritem kontrole zamašitve prometa, sami dogodki zamašitve prometa kot tudi velikosti medpomnilnika in obremenitve vmesnih vozlišč vzdolž povezave.
Prav tako na *perf-UDP* vplivajo velikost datagrama, število ostalega TCP ter UDP prometa in s tem posledično velikost medpomnilnikov vmesnih vozlišč. UDP temelji na principu prenosa po najboljših močeh in je posledično *perf-UDP* najbolj odvisna od velikosti in načina upravljanja medpomnilnika vozlišč omrežja. V nadaljevanju bomo podrobno pogledali faktorje vpliva na *perf-TCP*.

Velikost izravnalnega pomnilnika

Pri velikosti izravnalnega pomnilnika je pomemben čas, ki ga internetni paket (TCP/IP) potrebuje za pot od enega sistema do drugega in nazaj (RTT). Velja namreč, da je v pošiljanju lahko toliko nepotrjenih segmentov kot jih je lahko največ v izravnalnem pomnilniku, kar pomeni, da mora biti velikost izravnalnega pomnilnika najmanj tolikšna, kolikor znaša produkt BDP (ang. Bandwidth Delay Product):

$$BDP = BW \cdot RTT \ ,$$

če ne želimo, da je prepustnost protokola omejena s strani izvornega ali ciljnega vozlišča. Tabela 3.1 prikazuje nekaj kombinacij pasovne širine BW in parametra RTT, ki določata minimalno potrebno velikost izravnalnega pomnilnika

označeno z min TCP WND.

BW (*Mbps*)	*RTT* (*ms*)	min TCP WND (*KB*)
1	20	3
10	300	375
44	25	138
100	5	63
100	50	625
100	200	2500
1000	100	12500
10000	150	187500

Tabela 3.1: Tabela produkta *BDP*

Jedro sistema Linux privzeto od verzije 2.6.6 naprej, skrbi za velikost izravnalnega pomnilnika samodejno (ang. "autotunning") in nam za to pri povezavah majhne pasovne širine ni potrebno skrbeti. V kolikor pa želimo meriti prepustnost prometa TCP med napravami, ki imajo pasovno širino 1 *Gbps* ali več, so po [6] priporočljive vrednosti, podane v izpisu 3.3. Podane nastavitve je potrebno nastaviti v datoteki */etc/sysctl.conf*.

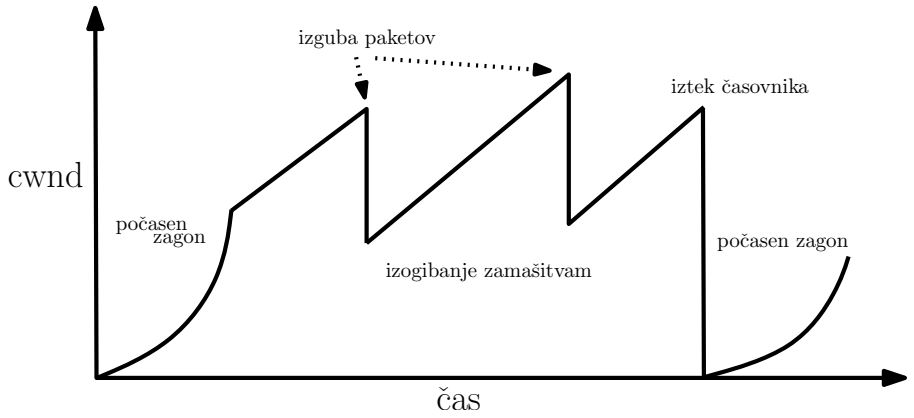
```
# increase TCP max buffer size setable using setsockopt()
# 16 MB with a few parallel streams is recommended for most 10G paths
# 32 MB might be needed for some very long end-to-end 10G or 40G paths
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
# increase Linux autotuning TCP buffer limits
# min, default, and max number of bytes to use
# (only change the 3rd value, and make it 16 MB or more)
net.ipv4.tcp_rmem = 4096 87380 16777216
net.ipv4.tcp_wmem = 4096 65536 16777216
# recommended to increase this for 10G NICS
net.core.netdev_max_backlog = 30000
# these should be the default, but just to be sure
net.ipv4.tcp_timestamps = 1
net.ipv4.tcp_sack = 1
```

Izpis 3.3: Nastavitev parametrov protokola TCP

Kontrola zamašitve omrežja

Kontrola zamašitve omrežja je pri protokolu TCP določena s štirimi prepletenimi algoritmi [13]: počasni zagon (ang. slow start), izogibanje zamašitvam (ang. congestion avoidance), ponovno pošiljanje (ang. fast retransmit) in hitro odpravljanje zamašitve (ang. fast recovery). Pri uporabi omenjenih algoritmov se uporabljajo spremenljivke `cwnd`, `rwnd` in `ssthresh`. `cwnd` določa največjo količino podatkov (v bajtih), ki jo pošiljatelj lahko pošlje, medtem ko predsta-

vlja `rwnd` količino podatkov, ki jo prejemnik lahko še prejme. Spremenljivka `ssthresh` določa ali se pri kontroli pošiljanja podatkov uporabi fazo izogibanja zamašitvam ali počasnega zagona. Počasen zagon predstavlja prvo fazo in si jo bomo podrobno ogledali v naslednjem podrazdelku, omenimo sedaj le, da gre za fazo, v kateri pošiljatelj z eksponentnim povečevanjem hitrosti pošiljanja podatkov poišče razpoložljivo pasovno širino pri čemer poskrbi, da sam ne zamaši omrežja. Ko vrednost `cwnd` preseže `ssthresh`, preide pošiljatelj v fazo izogibanja zamašitvam, v kateri hitrost pošiljanja podatkov povečuje linearno.



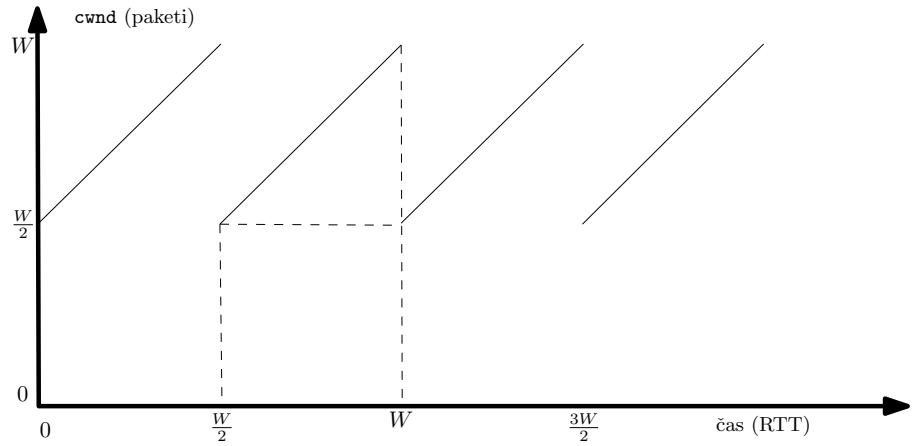
Slika 3.1: TCP kontrola zamašitve omrežja

V nekem času pride pri zaporedju paketov, ki so v prenosu, do izgube enega ali več paketov. Recimo, da pride do izgube i-tega paketa po vrsti. Prejemnik bo po prejemu $(i+1)$-tega paketa pošiljatelju poslal ACK zadnjega v zaporedju še zveznega paketa, tj. ACK paketa $(i-1)$. Pošiljatelj bo po prejemu tretjega podvojenega ACK paketa $(i-1)$ vzel to kot indikator, da je prišlo do izgube paketa i in prešel v fazo ponovnega pošiljanja ter hitrega odpravljanja zamašitve v kateri izgubljen paket ponovno pošlje in hkrati nastavi vrednosti `cwnd` in `ssthresh` na polovico tiste vrednosti `cwnd`, pri kateri je zaznal izgubo. Drugi indikator izgube paketa predstavlja časovnik, ki meri čas, v katerem pošiljatelj še čaka na ACK paketa i. Po izteku časovnika, bo pošiljatelj prešel v fazo slow start.

Za predstavo vpliva velikosti segmenta in časa RTT povzemimo v [15] predstavljen model faze izogibanja zamašitvam protokola TCP, ki uporablja algoritem TCP Reno. Model predpostavlja konstanten RTT in naključno izgubo paketa konstantne verjetnosti p.
V fazi izogibanja zamašitvam se v času RTT `cwnd` poveča za število bajtov, ki jih ima MSS [1]. Pri predpostavkah, ki jih imamo, velja, da se spremenljivka

[1]Pri implementaciji je to doseženo z majhnim povečevanjem spremenljivke cwnd pri

`cwnd` spreminja v obliki periodične žage – slika 3.2.



Slika 3.2: TCP izogibanje zamašitvam

Naj bo največja velikost okna W paketov. Torej velja, da bo v času faze izogibanja zamašitvam najmanjša velikost okna $W/2$ paketov. Če prejemnik pošlje ACK paket za vsak prejeti paket, se okno poveča na vsake RTT časa. To pomeni, da je ena perioda sestavljen iz $W/2$ RTT časov ali $RTT \cdot W/2$ sekund. Količina prenešenih podatkov na periodo je $(\frac{W}{2})^2 + 1/2(\frac{W}{2})^2 = \frac{3}{8}W^2$. Po začetni predpostavki velja, da v vsaki periodi prenesemo $\frac{1}{p}$ paketov.Velja torej:

$$W = \sqrt{\frac{8}{3p}} \cdot$$

Če omenjene izračune vstavimo v izračun *perf-TCP*, dobimo:

$$\textit{perf-TCP} = \frac{\text{podatki na periodo}}{\text{čas na periodo}} = \frac{\text{MSS} \cdot \frac{3}{8}W^2}{RTT \cdot \frac{W}{2}} = \frac{\text{MSS}/p}{RTT\sqrt{\frac{2}{3p}}} = \frac{\text{MSS}}{RTT}\frac{C}{\sqrt{p}} ,$$

kjer je $C = \sqrt{3/2}$.

Vidimo, da je *perf-TCP* sorazmeren z velikostjo MSS in obratno sorazmeren z RTT in p. Brez uporabe implementacije kakovosti storitev, torej velja, da ima promet z majhnimi paketi (IP telefonija), veliko slabši položaj od prometa velikih paketov (FTP prenosi). Prav tako pri paketih iste velikosti velja, da je pri dveh TCP povezavah iz istega vozlišča v prednosti tista, ki ima bližje

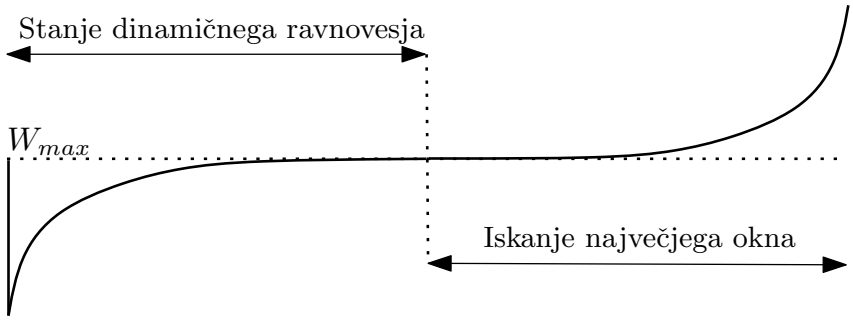
vsakem prejetem ACK paketu. Naj bo W velikost cwnd v paketih. Po prejetem ACK paketu se W poveča za $1/W$. Ker je $W = \text{cwnd}\,/\,\text{MSS}$, je sprememba okna naslednja: $\text{cwnd} = \text{cwnd} + \text{MSS} \cdot \text{MSS}\,/\,\text{cwnd}$.

ciljno vozlišče in manj zamašitev vzdolž same povezave.

Zadnji algoritem kontrole zamašitve omrežja, ki je v uporabi v jedru sistema Linux od verzije 2.6.19 naprej, je CUBIC [10] in predstavlja izboljšavo predhodnega algoritma BIC (ang. Binary Increase Control). Pomembna značilnost algoritma je, da je njegovo delovanje definirano le na podlagi stopnje izgube paketov, torej neodvisno od parametra RTT. Slednje močno vpliva na boljšo zmogljivost na povezavah, kjer je čas RTT nekaj 100 ms. Prav tako naj bi v primerjavi z algoritmom BIC imel boljšo kontrolo okna ter boljšo prijaznost do ostalega TCP prometa. Okno `cwnd` je določeno z naslednjo funkcijo:

$$W_{\text{cubic}} = C(t - K)^3 + W_{\max} \ ,$$

kjer je C skalarni faktor, t čas od zadnje redukcije okna, $W_{\max}$ velikost okna pred zadnjo redukcijo, β multiplikativni faktor, uporabljen pri redukciji okna v času izgube paketa, in $K = \sqrt[3]{W_{\max}\beta/C}$



Slika 3.3: CUBIC - rast okna

Slika 3.3 prikazuje rast funkcije W_{cubic}. Gre za kombinacijo konveksne in konkavne rasti – po redukciji je rast okna intenzivna, a ko se približa vrednosti $W_{\max}$, se upočasni in postane skoraj 0. Počasna rast okoli $W_{\max}$ izboljša stabilnost in uporabo razpoložljive pasovne širine, medtem ko nagla rast stran od $W_{\max}$ protokolu TCP zagotavlja razširljivost.
Podrobna razlaga in meritve prepustnosti, stabilnosti in odnosa do ostalega prometa so lepo opisane in prikazane v [10], omenimo le še, da je pri majhni vrednosti parametra RTT rast funkcije W_{cubic} počasnejša od linearne rasti algoritma TCP Reno, zaradi česar se po izgubi paketov prav tako izračuna in uporabi linearno rast, če je večja od rasti W_{cubic}.

TCP počasni zagon

Ocenimo, koliko časa bo protokol TCP v fazi počasnega zagona pri tem pa, kot velja pri nekaterih implementacijah kontrole zamašitve, vzemimo, da je

začetna vrednost `ssthresh` enaka vrednosti `rwnd` (rwnd $>$ *BDP*), vrednost `cwnd` pa enaka $4 \cdot \mathrm{MSS}$. Posledično bo faze konec, ko bo `cwnd` dosegla velikost *BDP*. Ker se `cwnd` eksponentno povečuje na vsake *RTT* časa, je potreben čas, da `cwnd` doseže `ssthresh` največ $T_s = \lceil \log_2(BDP \,/\, \mathrm{MSS}) \rceil \cdot RTT$ sekund. Vir [20] navaja, da zaradi uporabe zakasnjenih ACK paketov v realnosti znaša malce manj kot $2 \cdot T_s$, vendar to zaradi uporabe pravilnega štetja bajtov (RFC 3465 [2]) ne drži več.

BW-TCP (*Mbps*)	*RTT* (*ms*)								
	10	30	50	70	80	100	120	180	300
0.5	-	0.01	0.05	0.11	0.14	0.21	0.28	0.52	1.09
1	-	0.04	0.10	0.18	0.22	0.31	0.40	0.70	1.39
2	0.01	0.07	0.15	0.25	0.30	0.41	0.52	0.88	1.69
5	0.02	0.11	0.22	0.34	0.40	0.54	0.68	1.12	2.09
10	0.03	0.14	0.27	0.41	0.48	0.64	0.80	1.30	2.39
20	0.04	0.17	0.32	0.48	0.56	0.74	0.92	1.48	2.69
50	0.05	0.21	0.39	0.57	0.67	0.87	1.08	1.72	3.09
80	0.06	0.23	0.42	0.62	0.72	0.94	1.16	1.84	3.29
100	0.06	0.24	0.44	0.64	0.75	0.97	1.20	1.90	3.39

Tabela 3.2: Čas faze počasnega zagona v sekundah

Tabela 3.2 prikazuje koliko časa porabi protokol TCP v fazi počasnega zagona ob različnih kombinacijah pasovne širine in časa *RTT*. Dobro je razmisliti, kolikšen čas po fazi počasnega zagona je potreben, da dobimo vsaj 90 % dosegljive prepustnosti v fazi izogibanja zamašitvam. Velja, da bo faze slow start konec po $\log_2 W$ *RTT* časih, vsota prenesenih paketov pa bo $2(W-1)$. Povprečna prepustnost v fazi počasnega zagona je torej:

$$\mathit{perf\text{-}TCP}_{ss} = \frac{2(W-1) \cdot \mathrm{MSS}}{\log_2 W \cdot RTT}$$

Zanima nas, v katerem *RTT* času bomo dosegli 90 % vrednosti *perf-TCP*:

$$\frac{\log_2 W \cdot \mathit{perf\text{-}TCP}_{ss} + k \cdot \mathit{perf\text{-}TCP}}{\log_2 W + k} > 0.9 \cdot \mathit{perf\text{-}TCP}$$

Neenačba v drugi obliki pravi, da mora veljati:

$$k > 9 \log_2 W - 10 \log_2 W \cdot \frac{\mathit{perf\text{-}TCP}_{ss}}{\mathit{perf\text{-}TCP}} \,.$$

Groba ocena je torej, da mora meritev trajati najmanj 9-kratnik časa faze počasnega zagona ali najmanj pol toliko, če bi za meritev uporabili dve TCP seji.

Poudarimo, da so omenjene ocene opravljene na poenostavljenem modelu algoritma TCP Reno. Standardna faza počasnega zagona, ki jo uporablja TCP Reno in še nekaj ostalih implementacij, ne deluje dobro na povezavah kjer je velik *BDP*. Velja namreč, da pride zaradi eksponentne rasti okna `cwnd` do velike izgube paketov, kar za pošiljatelja in prejemnika predstavlja nezanemarljivo dodatno porabo procesorske moči pri procesiranju SACK paketov, ki se lahko konča tudi z večkratnim iztekom časovnika, kar pomeni prehod v fazo počasnega zagona. Prvi način rešitve problema predstavlja izboljšava obdelave SACK paketov znotraj TCP/IP sklada operacijskega sistema, drugi način pa se navezuje na samo izbljšavo faze počasnega zagona. Glede na to, da izboljšava faze počasnega zagona uporablja metode, ki smo jih predstavili v podrazdelku 2.1.2 in je privzeto v uporabi v zadnjih verzijah jedra sistema Linux, je primerno, da omenjeno izboljšavo predstavimo.

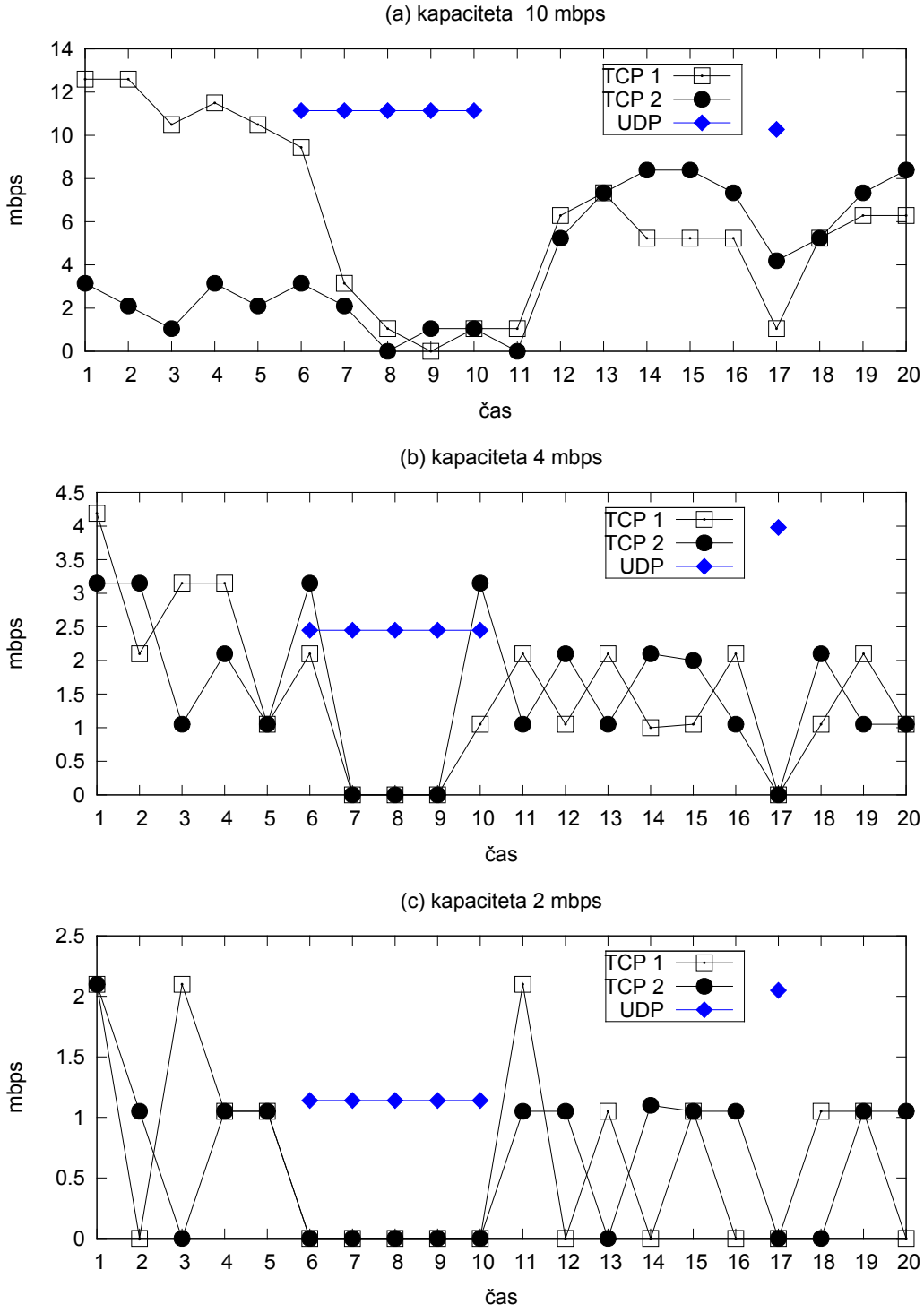
Nova, izboljšana verzija faze počasnega zagona se imenuje Hybrid start ali krajše HyStart [9]. Algoritem HyStart ne spremeni načina povečevanja okna `cwnd`, temveč na podlagi časovni razmikov paketov ACK in sledenju časa RTT ugotovi, kdaj je bolj varno, da preide v fazo izogibanja zamašitvam in s tem ne povrzoči prevelikega pošiljanja.
Ideja uporabe časovnih razmikov paketov ACK in opazovanja časov RTT je naslednja: naj $D_{\min}$ označuje minimalno enosmerno zakasnitev, ki je potrebna, da paket prispe do prejemnika, $\Delta(N)$ razpršenost serije paketov dolžine N in $\Lambda(N)$ pripadajočo razpršenost serije ACK paketov. Velja, da je $\Lambda(N) \geq \Delta(N)$. Algoritem HyStart preide v fazo izogibanja zamašitvam, ko `cwnd` doseže vrednost približno enako produktu $BW \cdot D_{\min}$ pri čemer je $b(N) = \frac{(N-1)L}{\Lambda(N)}$ ocena za BW ter polovica najmanjšega izmerjenega časa RTT ocena za $D_{\min}$. S preverjanjem, ali je $\Lambda(N)$ večji od $D_{\min}$, algoritem HyStart zazna, ali je `cwnd` dosegel razpoložljivo pasovno širino, ter ali bi bilo varno preiti v fazo izogibanja zamašitvam. Ko je na povezavi dosti ostalega prometa, ocena najmanjšega časa RTT ni merodajna in bo posledično metoda prehoda v fazo izogibanja zamašitvam s pomočjo serije ACK paketov manj učinkovita. Za varen prehod v fazo počasnega zagona se zato opazuje še trend časov RTT. Naj bo RTT_k povprečje prvih nekaj paketov v začetku k-te serije paketov. Algoritem HyStart preide v fazo izogibanja zamašitvam, ko je RTT_k večji od $RTT_{k-1} + \eta$, kjer je η konstanta.
HyStart algoritem je prvi [9] praktični primer uporabe metode razpršenosti

parov ali serije paketov 2.1.2. Izboljšave zmogljivosti algoritma HyStart se pokažejo že pri omrežjih s srednje velikim produktom *BDP* (med 100 in 1000 segmentov), medtem ko je algoritem velikih vrednosti produkta *BDP* v primerjavi z obstoječimi algoritmi počasnega zagona, nepremagljiv.

Ostali promet (ang. cross-traffic)

Zasnova protokola TCP omogoča, da se kapaciteta povezave porazdeli med vsemi TCP sejami, kar pomeni, da bo v primeru prisotnosti ostalega prometa naša prepustnost večja, če bomo za meritev uporabili večje število sej. Obratno, če na povezavi ni prisotnega nič ostalega prometa, uporaba večjega števila sej nima takšnega pomena, saj bomo celotno kapaciteto povezave zasedli v vsakem primeru, morda le ne bo toliko izkoriščena. Bralec si v [15] lahko pogleda zanimiv model porazdelitve TCP prometa na določeni topologiji, v kateri je danih $2n$ prehodov, izmed katerih je n zamašenih s povezavami, ki uporabljajo le en prehod, nas pa zanima, kakšen delež kapacitete povezave bo dobila TCP seja, ki prečka vseh $2n$ prehodov.

Mi se bomo sedaj osredotočili na naslednje vprašanje: imamo podatek o pasovni širini, ki jo imamo sklenjeno s ponudnikom internetnih storitev in želimo v eni sekundi s pomočjo protokola UDP preveriti ali je temu tako. Zanima nas, kakšen vpliv ima naš UDP promet na potencialno prisoten TCP promet. Slika 3.4 prikazuje grafe meritev, pri katerih smo iz strežnika do treh odjemalcev *TC* različnih lokacij opravili meritev prepustnosti protokola TCP z uporabo dveh sej v časovnem obsegu dvajsetih sekund. Istočasno smo na ločenem strežniku pognali še meritev prepustnosti prometa protokola UDP z nastavljeno hitrostjo, enako kapaciteti povezave. To meritev smo opravili na časovnih intervalih $[5, 10]$ in $[16, 17]$ dvajsetsekundne meritve protokola TCP. V prvem intervalu, ko smo UDP promet pošiljali polnih 5 sekund, je prišlo pri prometu protokola TCP do velike izgube paketov. Na povezavi kapacitete 10 *Mbps* je v tem času prepustnost dveh TCP sej okoli 2 *Mbps*, medtem ko se pri nižjih povezavah, tj. 4 in 2 *Mbps*, promet TCP ni uspel prebiti do cilja in je bila prepustnost protokola TCP enaka 0. To je bilo najbolj vidno pri povezavi kapacitete 2 *Mbps*, ko je to držalo polnih 5 sekund. Tudi izguba paketov prometa UDP je bila večja pri povezavi nižje kapacitete. Medtem, ko je bila pri povezavi kapacitete 10 *Mbps* le 4 %, je bila pri nižjih dveh 34 % in 43 %. Pri testu ene sekunde UDP promet ni doživel izgub, pri kapacitetah 4 in 2 *Mbps* je bil za TCP usoden, pri povezavah kapacitete 10 *Mbps* pa se je prepustnost le bistveno zmanjšala, a bila vendarle pozitivna. Zaključna ugotovitev, ki jo lahko razberemo, je torej dejstvo, da je UDP preveč nevaren, da bi ga lahko uporabljali za transparentno merjenje pasovne širine.



Slika 3.4: UDP vpliv na pretok prometa protokola TCP

Zmogljivost strojne opreme

V zadnjih desetih letih se je hitrost prenosa podatkov v računalniških omrežjih povečala za faktor 1000, zmogljivost centralne procesne enote je narasla za faktor 30, medtem ko se je prepustnost pomnilnika povečala le za faktor 10 in hitrost vodila PCI le za faktor 8. Če se bo ta trend v naslednjem desetletju nadaljeval, bo pri prenosih velike količine podatkov v nekem omrežju ozko grlo zagotovo strojna oprema povprečnega uporabnika.

3.1.2 Predstavitev parametrov

Parameter	Argument	Pomen
`-s, --server`		požene iperf kot strežnik
`-c, --client`	`<host>`	požene iperf kot odjemalec in se poveže na strežnik `<host>`
`-p, --port`	`#`	vrata, na katerih posluša/se poveže
`-u, --udp`		opravi UDP meritev namesto TCP
`-b, --bandwidth`	`#[kmgKMG]`	pri UDP meritvi nastavi hitrost pošiljanja v bitih na sekundo
`-t, --time`	`#`	čas meritve v sekundah (privzeto 10 sekund)
`-n, --num`	`#[KMG]`	število bajtov, ki naj se pošljejo pri testu (namesto `-t`)
`-P, --parallel`	`#`	število vzporednih testnih povezav
`-l, --len`	`#[KMG]`	količina podatkov, ki naj se naenkrat pošljejo
`-w, --window`	`#[KMG]`	velikost TCP okna (izravnalni pomnilnik)
`-M, --mss`	`#`	največja velikost TCP segmenta
`-N, --nodelay`		onemogoči Nagel-ov algoritem
`-Q, --tos`	`#[pt]`	nastavi polje ToS (TCP)
`-m, --print_mss`		izpiše največjo velikost TCP segmenta
`-T, --tcpinfo`		izpiše podroben TCP info
`-V, --verbose`		izpiše nekaj podrobnosti
`-i, --interval`	`#`	čas med periodičnim izpisom meritev
`-f, --format`	`#[kmgKMG]`	format izpisa: Kbits, Mbits, Gbits, KBytes, MBytes, GBytes

Tabela 3.3: Osnovni parametri orodja iperf

Pretvornik ima pri oznakah `K` – kilobyte, `M` – megabyte, `G` – gigabyte vrednost 1024, pri `k` – kilobit, `m` – megabit in `g` – gigabit pa 1000.
Večina parametrov in njihovih argumentov v tabeli 3.3 je bralcu najverjetneje razumljiva, v nadaljevanju bo na celovit način pojasnjena razlika in povezava med parametri `-l`, `-w` in `-M`. Pojasnili bomo tudi uporabo parametra `-Q`, ki je bil poleg dodane podpore protokola SNMP prav tako dodan iz naše strani.

V operacijskem sistemu Linux je omrežna povezava enako kot pri datoteki vezana na numerično vrednost, ki se ji reče deskriptor datoteke [4, 19]. Za pisanje in branje v datoteko ali pošiljanje in prejemanje podatkov v ali iz omrežja se torej uporablja iste sistemske klice — *write* in *read*.
Glavi funkcij sta z izjemo imena iste oblike. Poglejmo glavo funkcije *write*:

```
ssize_t write(int fildes, const void *buf, size_t nbyte);
```

Izpis 3.4: Glava funkcije write

Poleg deskriptorja datoteke je funkciji 3.4 kot argument podan še kazalec na pomnilnik `buf` in njegova velikost `nbyte`. Velja, da z argumentom parametera `-l` določimo velikost pomnilnika `buf` in ga s klicem funkcije *write* podamo v nadaljnjo obdelavo skladu TCP/IP operacijskega sistema, ta pa `buf` razbije na manjše enote, ki jih segment TCP sprejme kot podatkovni del.

Parameter `-M` določi največjo velikost TCP segmenta in posledično tudi velikost IP datagrama. V primeru meritev prepusnosti protokola UDP se velikost IP datagrama določi s parametrom `-l`. Če želimo pošiljati IP datagrame velikosti 1350, podamo parametru argument 1322 ($1350 - 20 - 8$).

S parametrom `-w` določimo velikost izravnalnega pomnilnika, ki smo ga razložili na začetku poglavja.

V omrežjih, v katerih je implementirana storitev QoS, lahko pride v uporabo tudi parameter `-Q`. Z njim nastavimo vrednost polja *DSCP* protokola IP. Sprva je bilo polje razdeljeno na 4 *"ToS"* bite in 3 *Precedence* bite, od katerih so se uporabljali le slednji. Kasneje se je polje redefiniralo in poimenovalo kot DSCP (DiffServ Code Points). V skladu s potekom sprememb in uporabo je bila implementirana tudi uporaba parametra — če želimo uporabiti polje kot *DSCP*, podamo le numerično vrednost, ki določa prioritetni razred. Če želimo uporabiti polje kot *ToS*, numerični vrednosti pripnemo znak `t`, če pa želimo od polja *ToS* uporabiti le *Precedence* bite, pa pripnemo znak `p`.

3.1.3 Pomen rezultatov

Upoštevati je treba, da iperf izpiše hitrost glede na količino prenešenih podatkov brez TCP/IP glave ter ethernet okvirja. Natančno razložimo kaj to pomeni v številkah, pred tem pa najprej povzemimo velikosti protokolarnih in podatkovnih enot sklada TCP/IP:

Ethernet: velikost protokolarnega dela ethernet okvirja je 42 bajtov. Najmanjša velikost podatkovnega dela je po standardu 46 bajtov, največja pa najpogosteje znaša 1500 bajtov – MTU.
V kolikor ne uporabimo t.i. jumbo okvirjev, je največja velikost ethernet okvirja 1542 bajtov. Jumbo okvirji (ang. jumbo frames) so navadno okvirji, ki nosijo podatkovni del velikosti 9000 bajtov in s strani javnih omrežij navadno niso podprti.

IP: najmanjša velikost glave IP datagrama je 20 bajtov, največja z uporabo dodatnih nastavitev znese 60 bajtov. Velikost celotnega IP datagrama je zapisana v 16-bitnem polju *Total length* in je torej lahko največ pa 65353 bajtov. Po standardu je najmanjša velikost datagrama 68 bajtov.

TCP: velikost glave TCP segmenta je analogna velikosti IP datagrama — med 20 in 60 bajti. Parameter MSS (ang. "maximum segment size") protokola TCP določa največjo podatkovno enoto, ki jo naprava lahko še sprejme. Vrednost parametra se nastavi pri vzpostavitvi TCP seje in velja, da se lahko za vsako smer nastavi na drugo vrednost. Privzeta vrednost parametra MSS je 536. Če želimo, da je celoten TCP segment vsebovan v IP paketu, mora torej veljati, da MSS skupaj z velikostjo protokolarnih podatkov ne presega MTU — velikost podatkovnega dela TCP segmenta je lahko kvečjemu 1460 bajtov.

UDP: velikost glave protokola UDP je 8 bajtov, velikost celotnega datagrama pa je omejena s 16-bitnim poljem *Length* — največ 65353 bajtov.

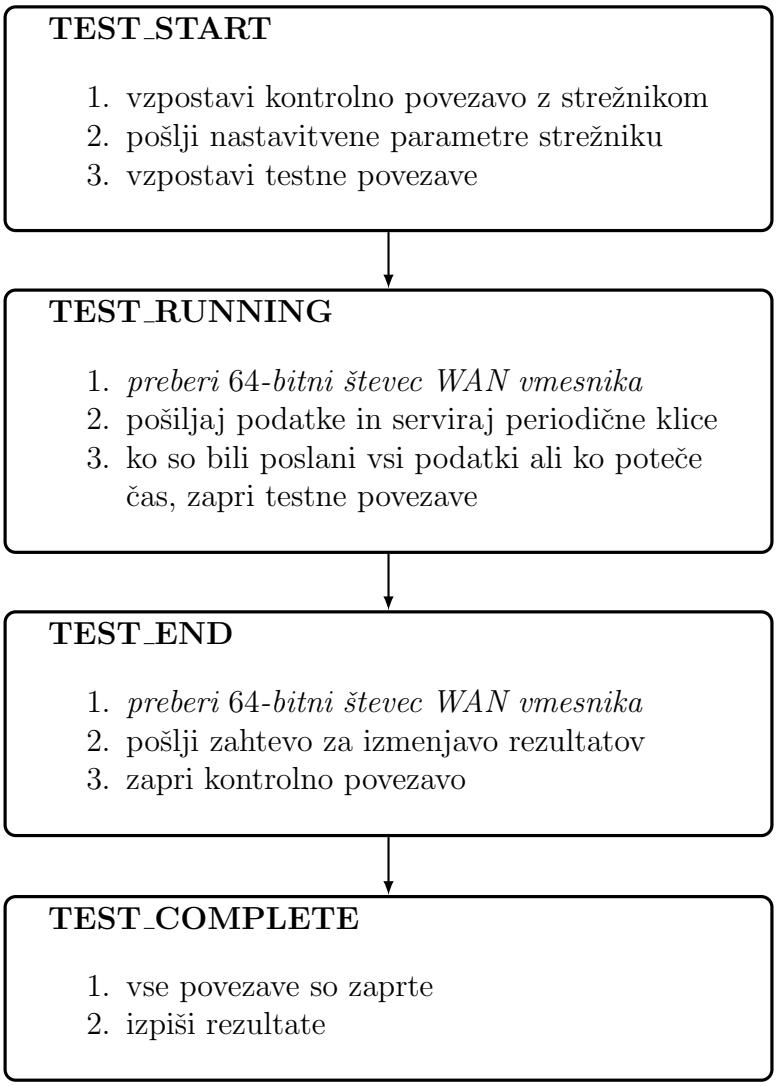
Velikost okvirja protokola ethernet je 42 bajtov. Če upoštevamo privzeto velikost glave protokola TCP in IP (20 bajtov), imamo pri prenosu ethernet okvirja velikosti 1542 bajtov, 82 $(42 + 20 + 20)$ bajtov protokolarnih podatkov. Pri meritvah v lokalnem omrežju s hitrostjo povezav 100 *Mbps*, bi pri testu protokola TCP na aplikacijskem nivoju teoretično dosegli kvečjemu 94.7 *Mbps* $(1460/1542 = 94.7\ \%)$ kapacitete povezave.

Poglejmo sedaj, kaj lahko pričakujemo pri meritvah na povezavah, kjer potuje promet skozi tunel "GRE over IPsec". Predpostavimo, da IPsec uporablja

algoritem za preverjanje pristnosti *SHA1* in algoritem za kriptiranje podatkov *AES*. Po [12] je dodatne režije tunela v velikosti 104 bajte, vse skupaj pa 186 ($104 + 42 + 20 + 20$) bajtov. To pomeni, da na aplikacijskem nivoju teoretično lahko dosežemo največ 85.2 % kapacitete povezave med končnima vozliščema.

3.1.4 Potek izvajanja odjemalca

Odjemalec vzpostavi kontrolno povezavo s strežnikom in mu pošlje parametre, ki določajo tip in lastnosti testa. Nato vzpostavi testne povezave in začne pošiljati podatke. Periodično lahko servira vmesne rezultate meritev. Ko poteče čas, ki je bil za test določen, ali ko so bili poslani vsi podatki, se zaprejo vse testne povezave, izmenjajo rezultati meritev in zapre kontrolna povezava. Odjemalec nato izpiše rezultate meritev in konča svoje izvajanje. Avtomat stanj, povzet po [7], je prikazan na sliki 3.5.



Slika 3.5: Iperf - avtomat stanj

3.2 Uporaba protokola SNMP

Protokol SNMP (ang. Simple Network Management Protocol) omogoča spremljanje naprav, ki jih imamo v omrežju. Parametri omrežne naprave, ki jih po protokolu SNMP lahko spremljamo, so definirani v nadzorni informacijski bazi ali na kratko MIB (ang. Management Information Base). Pri usmerjevalniku se med drugim lahko spremlja tudi števec prometa fizičnega vmesnika, ki nam pove količino prometa, ki je bil do tistega časa prenešen skozi dani vmesnik. Z uporabo protkola SNMP lahko lahko torej pred in takoj po meritvi prepustnosti protokola TCP do odjemalca TC iz usmerjevalnika R_{TC} preberemo vrednost števca vmesnika $ifce_{WAN}$ in tako dobimo količino celotnega prometa, ki je v tistem času potoval skozi vmesnik. Na ta način torej izmerimo BW, ki ga imamo na oddaljeni lokaciji.

3.2.1 Nastavitev usmerjevalnika

Pri uporabi protokola SNMP za branje 64-bitnih števcev prometa, ki potuje skozi vmesnik na usmerjevalniku, moramo vedeti, da SNMP agent spremebe beleži periodično. Možno je, da je perioda dolga tudi do 30 sekund. Za usmerjevalnike vodilnega svetovnega proizvajalca Cisco obstaja ukaz 3.5, s katerim je mogoče nastaviti periodo, po kateri bo agent osvežil števce na pravo vrednost prenešenega prometa. Za pravilno meritev v času nekaj sekund je uporaba omenjene nastavitve kritičnega pomena.

```
Router(config)# snmp-server hc poll 1
```

Izpis 3.5: Nastavitev osveževanja 64-bitnih števcev na 1 centisekundo

Pri branju števca prometa moramo pri zahtevi SNMP podati OID, ki se konča z indeksom, ki označuje vmesnik, katerega števec prometa nas zanima. Ker se vrednost indeksa po ponovnem zagonu usmerjevalnika lahko razlikuje od prejšnje, nam pride prav tudi ukaz 3.6, s katerim se indeksi po ponovnem zagonu usmerjevalnika ohranijo.

```
Router(config)# snmp-server ifindex persist
```

Izpis 3.6: Nastavitev ohranjevanja indeksov vmesnikov

3.2.2 Parametri za uporabo

Za uporabo protokola SNMP se mora podati vse parametre, ki so podani v tabeli 3.4. S parametrom `-S` omogočimo uporabo protkola SNMP, `-C` skupaj z

argumentom nastavi geslo, ki je potrebno za poizvedbo SNMP, s parametorm `-I` podamo IP naslov usmerjevalnika ter kot zadnje z `-X` nastavimo indeks vmesnika $ifce_{WAN}$.

Parameter	Argument	Pomen
`-S, --snmp-use`		vkopli meritev po SNMP
`-C, --snmp-community`	`<community>`	geslo zahteve protokola SNMP
`-I, --snmp-router-ip`	`<ip>`	IP naslov usmerjevalnika R_{TC}
`-X, --snmp-ifce-index`	`#`	vmesnik $ifce_{WAN}$ usmerjevalnika R_{TC}

Tabela 3.4: Dodani parametri orodja iperf za uporabo protokola SNMP

3.2.3 Implementacija

Implementacija podpore protokola SNMP je bila izvedena z uporabo knjižnice Net-SNMP. Znotraj zaglavne datoteke *iperf_api.h* se je definiralo strukturo *iperf_snmp* 3.7, katere spremenljivke se uporabljajo kot vhodni in izhodni podatki zahteve SNMP. Zahteva SNMP je izvedena znotraj funkcije *iperf_client_snmp_request* 3.8, ki je implementirana znotraj kodne datoteke *iperf_client_api.c*.

```
#ifdef BWSERVER
struct iperf_snmp {
  int    success;
  int    retries;
  float  timeout;
  int    timedout;
  char   gateway[31];
  char   community[101];
  char   net_in_oid[201];
  char   net_out_oid[201];
  char   cpu_oid[201];
  char   error_msg[501];
  int    oid_value_cpu_util;
  int    is_64bit_counter;
  struct timeval time_net_start;
  struct timeval time_net_end;
  unsigned long int snmp_time_price_first;
  unsigned long int snmp_time_price_second;
  unsigned long long oid_value_net_start;
  unsigned long long oid_value_net_end;
};
#endif
```

Izpis 3.7: Struktura iperf_snmp

Na kratko razložimo kako se uporablja spremenljivke strukture. Spremenljivka `timeout` definira čas pod katerim smo še pripravljeni čakati na odgovor zahteve SNMP. Spremenljivka `gateway` nosi naslov usmerjevalnika, na katerega bo zahteva poslana, `community` pa geslo le te. S spremenljivko `retries` nastavimo, kolikokrat želimo zahtevo SNMP ponovno poslati v primeru, da predhodno ne dobimo odgovora, `cpu_oid` nosi OID, s katerim dobimo podatek obremenitve procesne enote v zadnjih petih sekundah, `net_in_oid` in `net_out_oid` pa nosita OID, s katerim dobimo števec za vhodni in izhodni promet vmesnika $ifce_{WAN}$.
Spremenljivka `success` se ob obeh uspešnih poizvedbah branja števcev nastavi na 1 in nam na koncu pove, ali je meritev po protokolu SNMP uspela [2]. V primeru, da se nastavi na 0 in je zahteva bila neuspešna, se razlog neuspeha shrani v spremenljivko `error_msg`. V primeru, da pri prvi ali drugi zahtevi, le ta ne dobi odgovora v časovni omejitvi, spremenljivko `timedout` nastavimo na 1. Pri branju števca prometa vmesnika se cena časovne zahteve pri prvi zahtevi shrani v spremenljivko `snmp_time_price_first`, cena druge zahteve pa v `snmp_time_price_second`, medtem ko se vrednosti števca v prvi zahtevi shrani v `oid_value_net_start`, pri drugi pa v `oid_value_net_end`. Spremenljivki `time_net_start` in `time_net_end` sta tipa `struct timeval` in se ju uporabi pri izračunu časovne porabe zahtev poizvedb protokola SNMP.

```
iperf_client_snmp_request(struct iperf_test * test, int first_request);
```

Izpis 3.8: Glava funkcije za uporabo protokola SNMP

Funkcijo 3.8, ki prebere vrednosti števca vmesnika $ifce_{WAN}$, se v vlogi iperf odjemalca uporabi v znotraj funkcije `iperf_run_client` 3.9. Funkcijo se najprej pokliče, ko pride iperf v stanje *TEST_RUNNING*, drugič pa, ko poteče čas, ki je bil za test izbran, ali ko so bili poslani vsi podatki.

[2] SNMP sloni na transportnem protokolu UDP, ki pa ni zanesljiv in nam zato ne zagotavlja odgovora.

```
int iperf_run_client(struct iperf_test * test){
...
#ifdef BWSERVER
    int in_test_running_state = 0;
    struct timeval tv1, tv2;

    if(test->snmp_use){
      test->snmp->snmp_time_price_first = 0;
      test->snmp->snmp_time_price_second = 0;
      ...
      init_snmp("snmpapp");
    }
#endif
    while (test->state != IPERF_DONE) {
      ...
      if (test->state == TEST_RUNNING) {
#ifdef BWSERVER
      if(test->snmp_use){
        if(! in_test_running_state){
          in_test_running_state = 1;

          gettimeofday(&tv1, NULL);

          //PREBEREMO VREDNOST STEVCA TAKOJ KO ZACNEMO POSILJATI PODATKE
          iperf_client_snmp_request(test, 1);

          gettimeofday(&tv2, NULL);
          test->snmp->snmp_time_price_first = (tv2.tv_sec * 1000000 +
              tv2.tv_usec) - (tv1.tv_sec * 1000000 + tv1.tv_usec);
          ...
        }
      }
#endif
      if (test->reverse) {
          // Reverse mode. Client receives.
          if (iperf_recv(test) < 0)
              return (-1);
      } else {
          // Regular mode. Client sends.
          if (iperf_send(test) < 0)
              return (-1);
      }
      ...
      if (all_data_sent(test) || timer_expired(test->timer)) {
#ifdef BWSERVER
      if(test->snmp_use){
        if(test->snmp->timedout != 1 ) {
          gettimeofday(&tv1, NULL);

          //PREBEREMO VREDNOST TAKOJ KO KOCAMO S POSILJANJEM PODATKOV
          iperf_client_snmp_request(test, 0);

          gettimeofday(&tv2, NULL);
          test->snmp->snmp_time_price_second = (tv2.tv_sec * 1000000 +
              tv2.tv_usec) - (tv1.tv_sec * 1000000 + tv1.tv_usec);

          if(strcmp(test->snmp->error_msg, "") != 0){
            test->snmp->success = 0;
            ...
          } else {
            test->snmp->success = 1;
            ...
          }
        }
      }
#endif
```

Izpis 3.9: Uporaba protokola SNMP znotraj odjemalca iperf3

3.3 Primer uporabe

Poglejmo si sedaj nekaj primerov uporabe orodja iperf in na kratko pokomentirajmo rezultate, ki nam jih vrne. Vse nastavitve, ki nam jih orodje izpiše pri uporabi stikala `--help`, si lahko bralec pogleda v dodatku A.

Iperf lahko uporabimo za test zmogljivosti sklada TCP/IP sistema Linux ali Windows. Rezultati meritev so poleg implementacije samega sklada in sistema za upravljanjem s pomnilnikom v največji meri odvisni od strojne opreme, tj. hitrosti vodila, zmogljivosti centralne procesne enote in zmogljivosti in velikosti glavnega pomnilnika. Izpis 3.10 prikazuje rezultate testa zmogljivosti na lastnem osebnem prenosnem računalniku, na katerem teče operacijski sistem OpenSUSE 11.3.

```
marko@redondo:/> iperf3 -c 127.0.0.1 -t 5
Connecting to host 127.0.0.1, port 5201
[ ID] Interval       Transfer     Bandwidth
      Sent
[  5] 0.00-5.00 sec  9.42 GBytes  16.2 Gbits/sec
      Received
[  5] 0.00-5.00 sec  9.42 GBytes  16.2 Gbits/sec

iperf Done.
```

Izpis 3.10: Iperf test prepustnosti do vmesnika loopback

Izpis 3.11 prikazuje meritev prepustnosti in pasovne širine protokola TCP. Dolžina testa je trajala 5 sekund, pri tem je bilo nastavljeno okno TCP velikosti 2560 kilobajtov, naključna podatkovna enota, ki smo jo podali skladu TCP/IP, velikosti 128 kilobajtov, in uporaba protokola SNMP z geslom "public", IP naslovom 172.16.12.121 in indeksom vmesnika 4. V času testa je bilo prenešenih 5.25 megabajtov podatkov kar je pomenilo, da smo dosegli hitrost pošiljanja podatkov enako 8.64 *Mbps*. Glede na to, da je ta rezultat dobrih 90 % rezultata, ki smo ga namerili po protokolu SNMP, lahko sklepamo, da je bil pri testu na povezavi med končnima vozliščema, prisoten le naš promet. Vidimo tudi, da smo porabili 5 in 81 milisekund za poizvedbi protokola SNMP in da je bila obremenitev usmerjevalnika v zadnjih petih sekundah 2 %.

```
root:/ # iperf3 -c 10.255.12.126 -p 9010 -w 2560K -t 5 -l 128.00K -S -C
    public -I 172.16.12.121 -X 4
------------------------------------------------------------------------
[TCP S] 0.00-5.10 sec  5.25 MBytes  8.64 Mbits/sec
[TCP R] 0.00-5.10 sec  5.25 MBytes  8.64 Mbits/sec
[SNMP ] time: 5.088 time_price: 5 + 81 mbps:  9.58 cpu: 2
```

Izpis 3.11: Iperf test prepusnosti in pasovne širine protokola TCP

Meritev prepusnosti protokola UDP pri pošiljanju podatkov s hitrostjo 10 *Mbps* je prikazan na izpisu 3.12. Poleg že opisanih rezultatov nam meritev prepustnosti protokola UDP da tudi število vseh poslanih datagramov, kot tudi število tistih, ki jih je med potjo do prejemnika doletela kruta usoda. Omenjena podatka sta predstavljena tudi kot izguba paketov v odstotkih (4.4 %).

```
root:/ # iperf3 -c 192.168.117.131 -p 9011 -t 2 -u -b 10M -S -C public -I
    192.168.117.130 -X 2
------------------------------------------------------------------------
[UDP S] 0.00-2.29 sec 2.68 MBytes 9.81 Mbits/sec  0.000 ms  85/ 1937 (4.4%)
[UDP R] 0.00-2.29 sec 2.56 MBytes 9.38 Mbits/sec  0.000 ms  85/ 1937 (4.4%)
[SNMP ] time: 2.281 time_price: 6 + 290 mbps: 11.16 cpu: 4

iperf Done.
```

Izpis 3.12: Iperf test prepusnosti in pasovne širine protokola UDP

Poglejmo si še zadnji primer, v katerem smo pri testu, ki je trajal 3 sekunde, uporabili dve TCP seji in uporabili stikalo `-i`, s katerim smo nastavili izpis vmesnih meritev vsako sekundo. Vidimo, da sta seji enakomerno zavzeli svoj delež kapacitete in skupaj prenesli 7.25 *MB*.

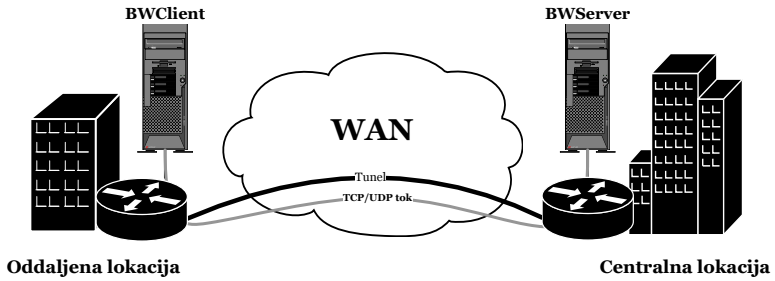
```
root:/ # iperf3 -c 192.168.117.131 -p 9010 -t 3 -i 1 -P 2 -w 256K
Connecting to host 192.168.117.131, port 9010
[ ID] Interval       Transfer     Bandwidth
[  5] 0.00-1.03 sec  1.12 MBytes  9.12 Mbits/sec
[  6] 0.00-1.03 sec  1.38 MBytes  11.1 Mbits/sec
[SUM] 0.00-1.03 sec  2.50 MBytes  20.3 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 1.03-2.08 sec  1.25 MBytes  10.0 Mbits/sec
[  6] 1.03-2.08 sec  1.38 MBytes  11.0 Mbits/sec
[SUM] 1.03-2.08 sec  2.50 MBytes  20.1 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
      Sent
[  5] 0.00-3.06 sec  3.50 MBytes  9.61 Mbits/sec
      Received
[  5] 0.00-3.06 sec  3.50 MBytes  9.61 Mbits/sec
      Sent
[  6] 0.00-3.06 sec  3.75 MBytes  10.3 Mbits/sec
      Received
[  6] 0.00-3.06 sec  3.75 MBytes  10.3 Mbits/sec
      Total sent
[SUM] 0.00-3.06 sec  7.25 MBytes  19.9 Mbits/sec
      Total received
[SUM] 0.00-3.06 sec  7.25 MBytes  19.9 Mbits/sec

iperf Done.
```

Izpis 3.13: Iperf test z izpisom vmesnih rezultatov in uporabo dveh TCP sej

Poglavje 4

Bandwidth Monitor Orodje

Orodje BWMonitor sestavljata strežnik in odjemalec, zgrajena kot klasični proces sistema Linux, ki v ozadju opravlja svoje poslanstvo. Strežnik bomo poimenovali BWServer, odjemalec pa BWClient. BWServer periodično opravi meritve do vseh oddaljenih lokacij, do katerih je meritev omogočena. Sama perioda opravljanja meritev in lastnosti le-teh so določene v podatkovni bazi. Kot bomo videli v predstavitvi sheme podatkovne baze, se rezultati prav tako shranijo v podatkovno bazo. Slika 4.1, ki predstavlja ciljni primer uporabe, prikazuje način uporabe BWServerja in BWClienta.



Slika 4.1: BWMonitor sistem

4.1 Specifikacije

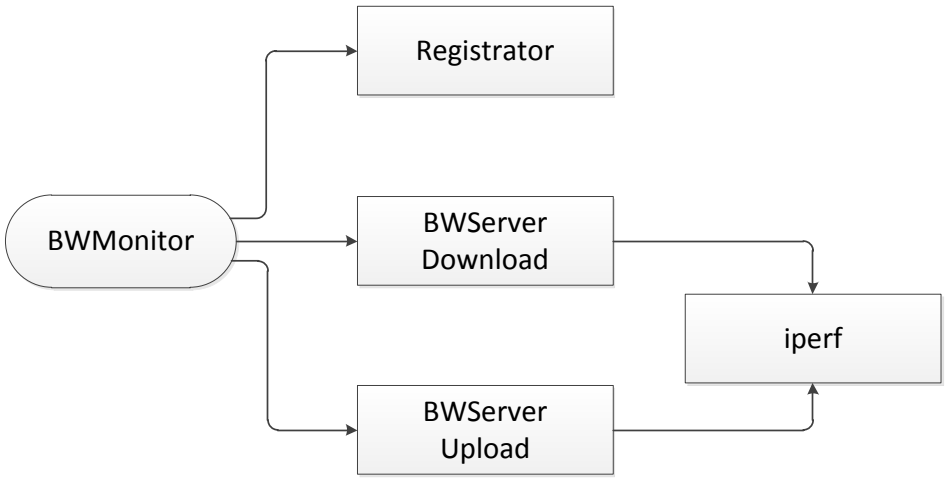
Specifikacije orodja so naslednje:

- periodične meritve prepustnosti protokola TCP ali UDP,
- periodične meritve pasovne širine z uporabo protokola SNMP,
- večkratne meritve do iste lokacije z različnimi nastavitvami,
- nastavitev TCP vrat kontrolne in testne povezave,
- nastavitev velikosti izravnalnega pomnilnika (socket buffer size),
- nastavitev velikosti TCP segmenta ali UDP datagrama,
- nastavitev maksimalne velikosti segmenta protokola TCP (MSS),
- nastavitev IP naslova testne povezave na strani strežnika,
- nastavitev časovne dolžine testa,
- meritev z večkratnimi TCP sejami ali UDP tokovi,
- nastavitev polja DSCP (Differentiated Services Code Point),
- nastavitev opcije TCP No Delay,
- zasnova na večkratnih strežnikih ⇒ skalabilnost števila meritev.

4.2 Struktura in potek izvajanja

4.2.1 Strežnik

BWServer sestavljajo trije procesi, ki se usvarijo ob samem zagonu strežnika. Eden izmed procesov skrbi za registracijo BWClientov, ostala dva pa opravljata meritve proti in iz lokacije. Same meritev se opravi v ločenem procesu, v katerem se kliče funkcije knjižnice iperf. Zaradi večprocesne zasnove se uporablja skupni pomnilnik, v katerega se shranjujejo rezultati meritev. Po tem, ko se opravi vse meritve, se rezultate shrani v podatkovno bazo z uporabo knjižnice MySQL C API. S pomočjo omenjene knjižnice BWServer dostopa do podatkovne baze tudi pri registraciji novih BWClientov in preden začne opravljati meritve.



Slika 4.2: Procesi

Datoteka bwmonitor-server.conf

Datoteka predstavlja primarno točko nastavitve sistema. Spremenljivke v datoteki prebere BWServer ob zagonu — pri kakršnih koli spremembah spremenljivk ga je potrebno za uveljavitev sprememb ponovno zagnati.

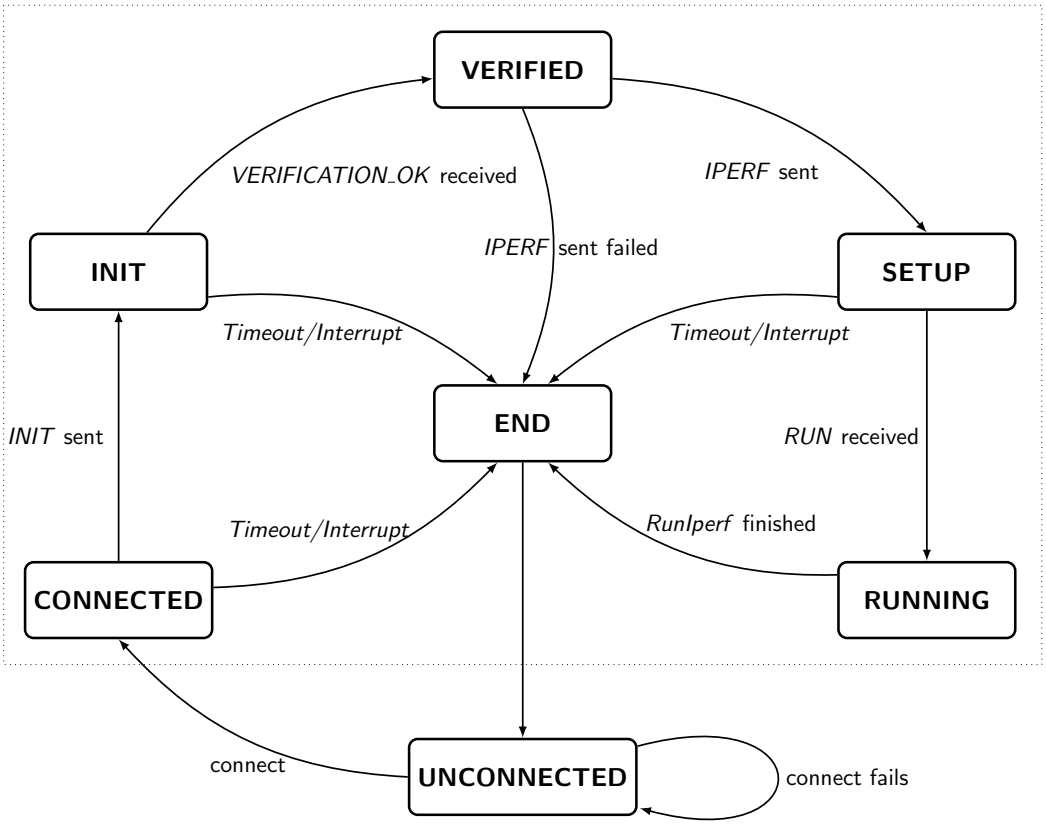
```
DIR_BINARY=/opt/bwmonitor-server
HOSTNAME=<cactihostname>
LOG_LEVEL=4
REGISTRATOR_PORT=9011
ENABLED_BW_FILE_LOG=0
PROCESSED_FILE_LOG_DIR=/opt/bwmonitor-server

DB_HOST=<cactihostname>
DB_DATABASE=bwmonitor
DB_USERNAME=cactiuser
DB_PASSWORD=<cactiuser_pass>
DB_PORT=3306
```

Izpis 4.1: Nastavitev bwmonitor-server.conf

Spremeljivka `DIR_BINARY` je ime direktorija, v katerem se nahaja binarna datoteka *bwmonitor*. `HOSTNAME` spremenljivka določa ime BWServerja. Na podlagi danega imena strežnik iz podatkovne baze naloži ostale nastavitve. Za samo tekstovno shranjevanje rezultatov mora biti spremeljivka `ENABLED_BW_FILE_LOG`

nastavljena na 1. `LOG_LEVEL` določa nivo logiranja, `REGISTRATOR_PORT` pa vrata, na katerih strežnik posluša BWCliente, ki se želijo registrirati.
`PROCESSED_FILE_LOG_DIR` določi direktorij, v katerega BWServer zapisuje število uspešno opravljenih meritev in seznam BWClientov, do katerih je bila meritev neuspešna. Spremenljivke, ki se začnejo z `DB_`, določajo nastavitve, potrebne za povezavo do podatkovne baze.

Končni avtomat stanj



Slika 4.3: BWServer - avtomat stanj

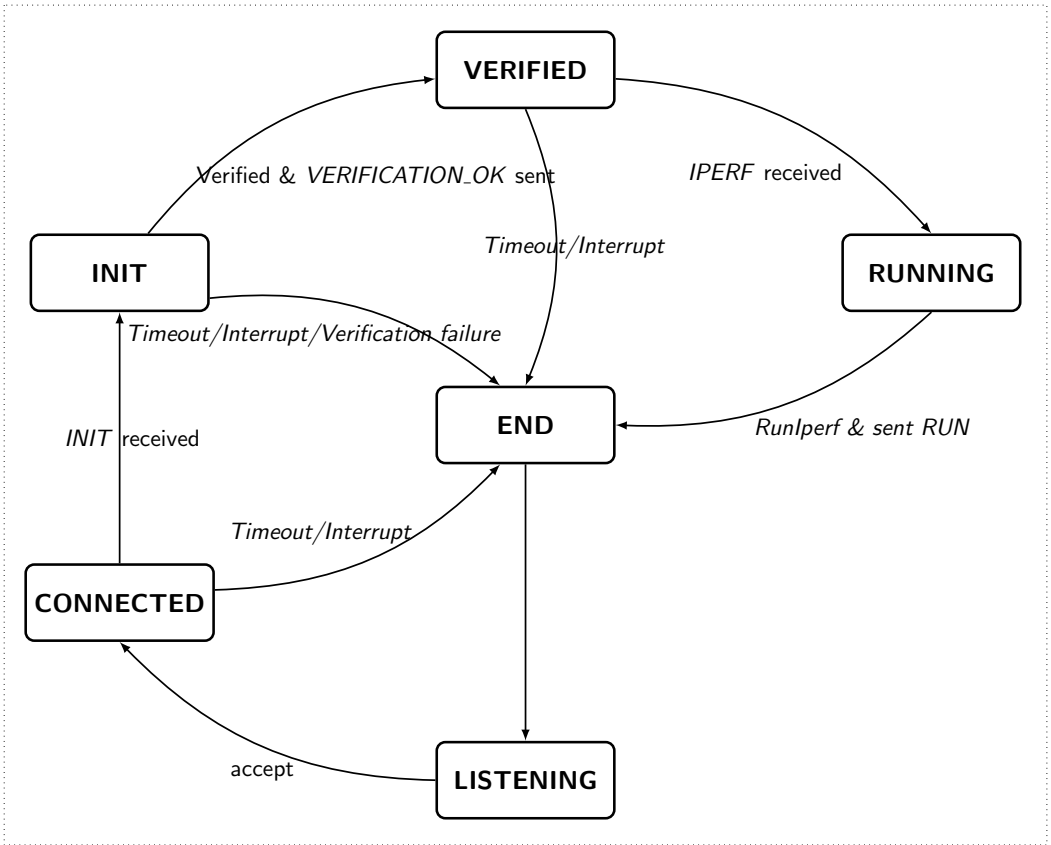
BWServer po vzpostavitvi povezave do BWClienta pošlje sporočilo *INIT*. BWClient ga preveri tako, da pogleda ali se njegov IP naslov ujema s katerim izmed naslovov, ki jih s strani imenskega strežnika dobi pri pretvorbi imen *nilbwmonitor*, *nilbwmonitor1*, *nilbwmonitor3*, ..., *nilbwmonitor9*. V kolikor je preverjanje uspešno, strežnik prejme sporočilo *VERIFICATION_OK*, nato pa takoj odgovori z sporočilom *IPERF*, s katerim odjemalcu sporoča, da želi opraviti meritev z orodjem iperf. BWClient po sprejemu sporočila požene

proces iperf in sporoči strežniku, da naj začne z meritvami.

4.2.2 Odjemalec

Končni avtomat stanj

Končni avtomat prehajanja stanj pri BWClientu, prikazan na sliki 4.4, je po tem, ko smo videli potek pri BWServerju najverjetneje jasen in ga zato ne bomo razlagali.



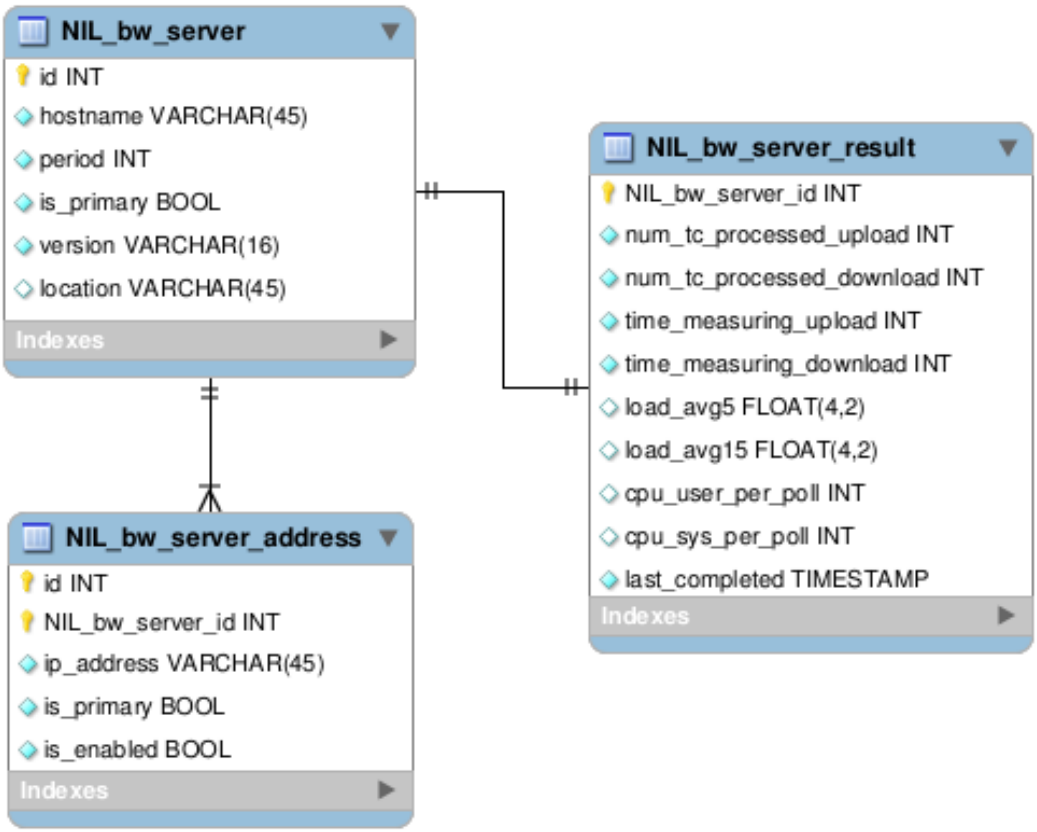
Slika 4.4: BWClient - avtomat stanj

4.3 MySQL podatkovna shema

4.3.1 Nastavitev strežnika

Nastavitve strežnika se nahajajo v tabelah NIL_bw_server in NIL_bw_server_address. Vsak BWServer ima enolično ime — polje `hostname`, ki se mora ujemati s spremenljivko `HOSTNAME` v nastavitveni datoteki *bwmonitor-server.conf*

(glej 4.1) samega strežnika, ki ga opisuje. Polje `period` določa periodo izvajanja meritev v minutah, `is_primary` določa ali je strežnik primaren in opravlja tudi vlogo registratorja, polje `version` pa vsebuje številko verzije BWServerja in se uporablja pri posodobitvah BWClienta. Vsak strežnik ima lahko več IP naslovov — ti se nahajajo v tabeli NIL_bw_server_address. Poleg naslova `ip_address` nosi poljuben zapis tabele še podatek o tem, ali je naslov primaren in se ga pri meritvah lahko uporabi — polji `is_primary` in `is_enabled`. Bwmonitor strežnik tabele NIL_bw_server_result zaenkrat ne uporablja, služila pa naj bi shranjevanju podatkov zadnje meritve strežnika — koliko meritev je BWServer opravil, koliko časa je pri tem porabil in kakšna je bila obremenitev strežnika v tem času.



Slika 4.5: Tabele BWServerja

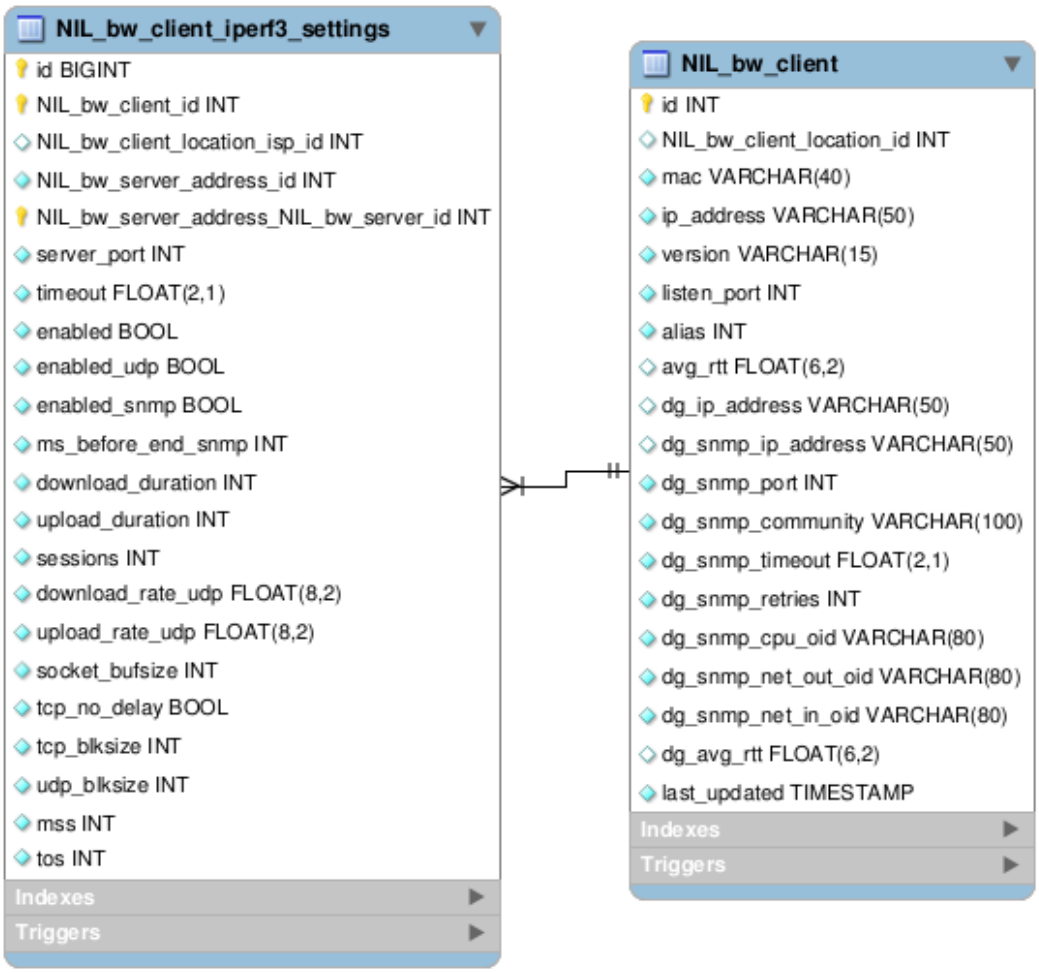
4.3.2 Nastavitev odjemalca in njegovih meritev

Tabela NIL_bw_client vsebuje podatke o vseh registriranih BWClientih do katerih je mogoče izvajati meritve, število meritev in nastavitve le-teh pa se

nahajajo v tabeli NIL_bw_client_iperf3_settings.

Polje `NIL_bw_client_location_id` je tuji ključ in se nanaša na zapis v tabeli NIL_bw_client_location 4.7. Pri registraciji BWClienta se vnese nov zapis z vnosom naslednjih polj:

- `mac` - fizični naslov vmesnika odjemalca TC,
- `ip_address` - IP naslov vmesnika odjemalca TC,
- `version` - verzija BWClienta, ki se nahaja na odjemalcu TC,
- `dg_ip_address` - IP naslov privzetega prehoda odjemalca TC.



Slika 4.6: Tabele BWClienta

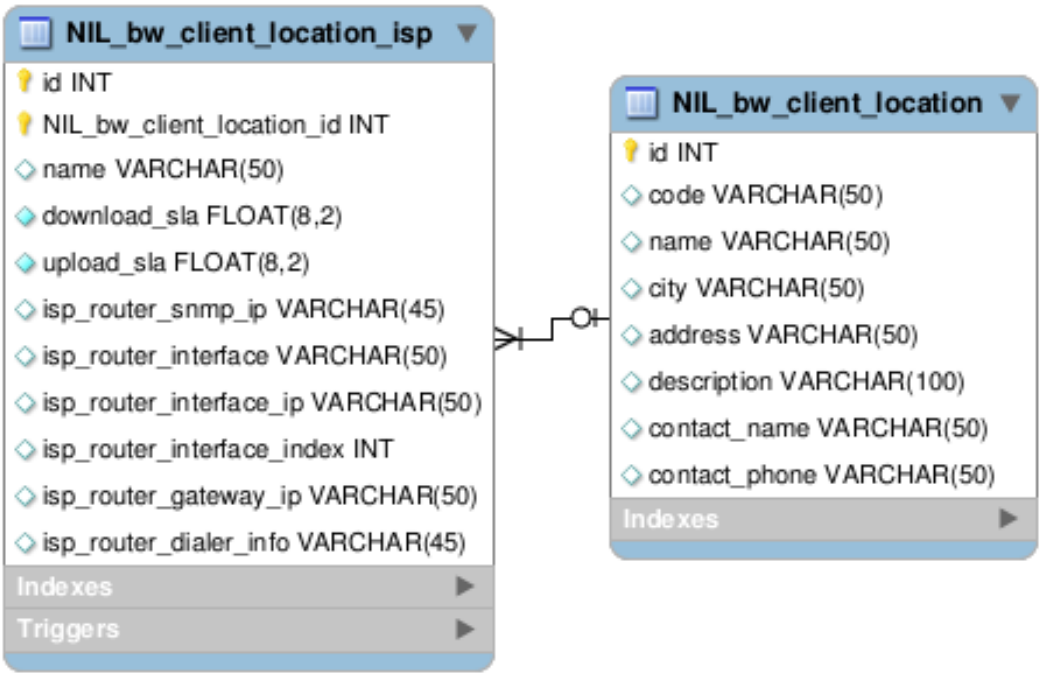
Polja, katerih imena se začnejo z `dg_snmp_`, določajo nastavitve, ki so potrebne pri zahtevi protokola SNMP. Pri dodajanju novega zapisa se njihove vrednosti

nastavijo na privzete. Vrednost polja `listen_port` določa vrata, na katerih BWClient čaka na povezavo s strani BWServerja, `alias` pa je namenjen za preslikavo med identifikatorjem zapisa razlagane tabele in nekim določenim zapisom tabele ločene podatkovne baze. Polji `avg_rtt` in `dg_avg_rtt` sta namenjeni za vnos časa RTT vrednosti do naprave oz. do privzetega prehoda. Večina polj v tabeli NIL_bw_client_iperf3_settings predstavlja nastavitev, ki jo podamo procesu iperf3 pred izvajanjem meritev in večina jih ima nastavljeno privzeto vrednost, ki sovpada s privzeto vrednostjo pripadajočega parametra orodja iperf3.
Primarni ključ tabele sestavljajajo `id`, id BWClienta — `NIL_bw_client_id` in id BWServerja — `NIL_bw_server_address_NIL_bw_server_id`. Polja nam povejo, na katerega BWClienta se nastavitev nanaša, iz katerega BWServerja se bodo meritve izvajale ter tudi število definiranih meritev, vezanih na BWClienta.
Polje `enabled` določa ali se naj meritev do naprave BWClienta izvaja ali ne, `timeout` pa predstavlja čas, po katerem BWServer pokonča proces iperf3, če le ta še ni končal z izvajanjem meritve.
Opis ostalih polj je mogoče najti pri predstavitvi orodja iperf3 3.1, omenimo le, da je parameter, ki določa čas izvajanja meritve uporabljen v odvisnosti od smeri izvajanja meritve — `download_duration` in `upload_duration`. Isto velja za hitrost pošiljanja datagramov protokola UDP — `download_rate_udp` in `upload_rate_udp`.

4.3.3 Nastavitev lokacij in ponudnikov interneta

Ker se vsak odjemalec *TC* in posledično BWClient nahaja na neki lokaciji, na kateri povezava v internet poteka preko enega ali več ponudnikov interneta, se podatke o tem hrani v tabelah NIL_bw_client_location in NIL_bw_client_location_isp. Tabela NIL_bw_client_location vsebuje kodo lokacije, ime lokacije, mesto, točen naslov in še nekaj drugih bralcu iz imena razumljivih podatkov.
Tabela NIL_bw_client_location_isp ima naslednja polja: ime ponudnika interneta — `name`, podatek o pasovni širini v *Mbps*, ki je zakupljen pri ponudniku — `download_sla` in `upload_sla`. Polja, katerih ime se začne z `isp_router_`, se nanašajo na usmerjevalnik in nam dajo informacijo o imenu vmesnika iz katerega gre povezava do internetnega ponudnika (`isp_router_interface`), naslov IP, ki je nastavljen na vmesniku (`isp_router_interface_ip`), indeks vmesnika, ki se uporabi pri zahtevi SNMP za pridobitev vrednosti števca IP paketov, ki potujejo skozi vmesnik (`isp_router_interface_index`) in naslov IP privzetega prehoda usmerjevalnika (`isp_router_gateway_ip`). Polje `isp_router_snmp_ip` nosi enako vrednost kot polje `dg_ip_address` ali

`dg_snmp_ip_address` tabele NIL_bw_client 4.6, medtem ko polje `isp_router_dialer_info` ni več uporabljeno.

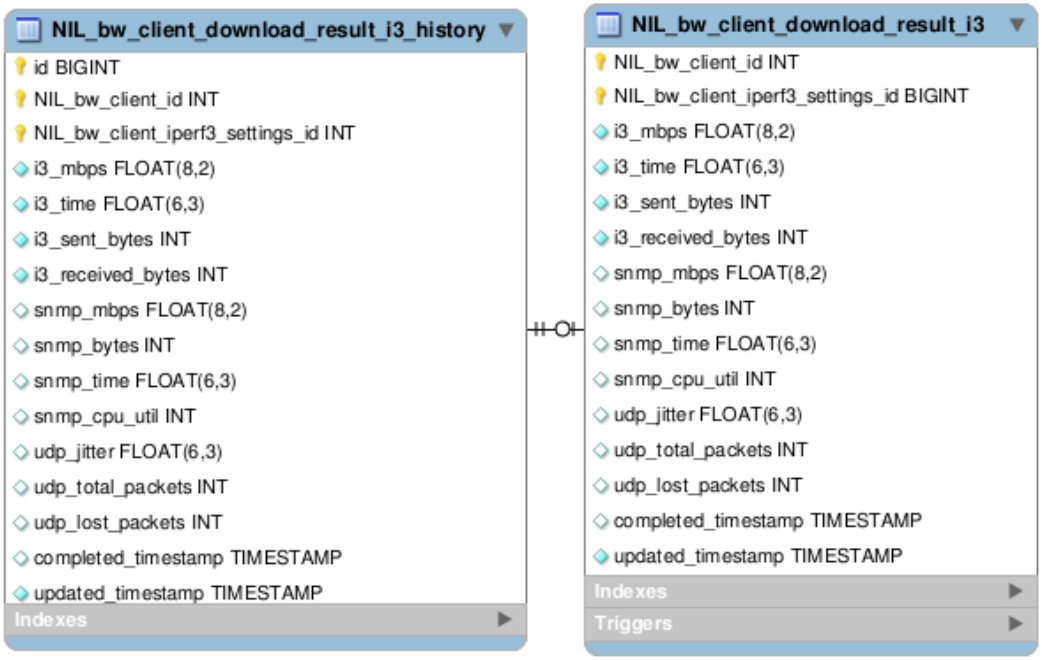


Slika 4.7: Tabele lokacij in ponudnikov interneta

4.3.4 Rezultati meritev

Po končanih meritvah BWServer zapiše rezultate v tabeli *NIL_bw_client_download_result_i3* in *NIL_bw_client_upload_result_i3*, na tabelah definirani sprožilci pa iste rezultate vstavijo v pripadajoči tabeli *NIL_bw_client_download_result_i3_history* ali *NIL_bw_client_download_result_i3_history*. Na sliki 4.8 je prikaz tabel za meritve v smeri proti BWClientu — download meritev.

Razlaga rezultatov, ki jih dobimo od procesa iperf, se nahaja v razdelku 3.1. Tu velja omeniti le, da polje `completed_timestamp` nosi do sekunde natančno informacijo o končani meritvi, `updated_timestamp` pa informacijo o času zapisa podatka v tabelo.



Slika 4.8: Tabeli razultatov

4.3.5 Logične tabele

Za enostavno poizvedbo pri branju vseh vhodnih podatkov, vezanih na BWClienta in enostaven in logičen pregled rezultatov meritev, so bile usvarjene logične tabele — slika 4.9.
Logična tabela *NIL_bw_client_bwmonitor* združuje tabele NIL_bw_client, NIL_bw_client_iperf3_settings, NIL_bw_client_location_isp, NIL_bw_client_location in NIL_bw_server_address. Z omenjeno združitvijo strežnik BWMonitor enostavno pride do vseh potrebnih podatkov BWClienta.
Poleg združevanja download in upload meritev je namen ostalih logičnih tabel naslednji:

NIL_bw_client_results_i3 – prikaz zadnjih meritev brez nekaterih polj kot so `i3_sent_bytes` in `i3_received_bytes`;

NIL_bw_client_results_i3_mbps – prikaz zadnjih meritev brez nekaterih polj in skupaj z vrednostimi SLA in nekaterimi nastavitvami;

NIL_bw_client_results_i3_full – prikaz zadnjih meritev skupaj z vrednostimi SLA in nekaterimi nastavitvami;

NIL_bw_client_results_i3_short – dodani podatki o lokaciji BWClienta — IP naslov usmerjevalnika, indeks vmesnika, združitev `enabled`, `enabled_udp` in `enabled_snmp` polj v le eno;

NIL_bw_client_results_i3_brief – prikaz polj zadnjih meritev, vezanih na izračun prepustnosti;

NIL_bw_client_results_i3_history – prikaz meritev za celotno zgodovino.

Slika 4.9: Logične tabele

4.3.6 Sprožilci

Razložimo sedaj potrebe in namen sprožilcev. Pri registraciji BWClienta se podatke le-tega doda v tabelo NIL_bw_client, nastavitev meritev pa še ni definiranih. Ker želimo, da se usvarijo tudi privzete nastavitve, to dosežemo z uporabo sprožilca *insert_default_iperf3_settings* 4.2.

```
CREATE TRIGGER insert_default_iperf3_settings
AFTER INSERT ON NIL_bw_client
FOR EACH ROW BEGIN
  INSERT INTO NIL_bw_client_iperf3_settings(NIL_bw_client_id)
  VALUES(new.id);
END$$
```

Izpis 4.2: Sprožilec insert_default_iperf3_settings

Pri uporabi protokola UDP se uporabljata polji `download_rate_udp` in `upload_rate_udp`. Privzeto sta nastavljena na 1 Mbps. V primeru, da se nastavi podatek o ponudniku interneta, želimo, da se vrednost omenjenih polj

nastavita na vrednost `download_sla` oz. `upload_sla`. Slednje dosežemo z sprožilcem *update_udprate* 4.3.

```
CREATE TRIGGER update_udprate
BEFORE UPDATE ON NIL_bw_client_iperf3_settings
FOR EACH ROW
BEGIN
  DECLARE down_sla FLOAT;
  DECLARE up_sla FLOAT;

  IF (NEW.NIL_bw_client_location_isp_id !=
    OLD.NIL_bw_client_location_isp_id OR
    OLD.NIL_bw_client_location_isp_id IS NULL) AND
    NEW.NIL_bw_client_location_isp_id IS NOT NULL
  THEN
    SELECT download_sla INTO down_sla
    FROM NIL_bw_client_location_isp
    WHERE id = NEW.NIL_bw_client_location_isp_id;

    SELECT upload_sla INTO up_sla
    FROM NIL_bw_client_location_isp
    WHERE id = NEW.NIL_bw_client_location_isp_id;

    SET NEW.download_rate_udp = down_sla,
        NEW.upload_rate_udp = up_sla;
  END IF;
END$$
```

Izpis 4.3: Sprožilec update_udprate

Ker se v tabeli *NIL_bw_client_location_isp_id* 4.6 vrednost polj `download_sla` oz. `upload_sla` lahko spremeni, želimo, da se sprememba odraža tudi na poljih `download_rate_udp` in `upload_rate_udp` tabele NIL_bw_client_iperf3-_settings. Nalogo sinhronizacije vrednosti omenjenih polj opravlja sprožilec *update_iperf3_settings_udp_rate* 4.4.

```
CREATE TRIGGER update_iperf3_settings_udp_rate
AFTER UPDATE ON NIL_bw_client_location_isp
FOR EACH ROW BEGIN
  IF NEW.download_sla != OLD.download_sla OR
      NEW.upload_sla != OLD.upload_sla THEN
    UPDATE NIL_bw_client_iperf3_settings SET
      download_rate_udp = NEW.download_sla,
      upload_rate_udp = NEW.upload_sla
    WHERE NIL_bw_client_location_isp_id = NEW.id;
  END IF;
END$$
```

Izpis 4.4: Sprožilec update_iperf3_settings_udp_rate

Naloga sprožilca *insert_client_download_result_i3_history* 4.5 je, da pri zapisu aktualnih rezultatov v tabelo *NIL_bw_client_download_result_i3* 4.8 le-te zapiše tudi v tabelo *NIL_bw_client_download_result_i3_history* (analogno tudi pri tabeli *NIL_bw_client_upload_result_i3*).

```
CREATE TRIGGER insert_client_download_result_i3_history
AFTER UPDATE ON NIL_bw_client_download_result_i3
FOR EACH ROW BEGIN
  INSERT INTO NIL_bw_client_download_result_i3_history(
    NIL_bw_client_id,
    NIL_bw_client_iperf3_settings_id,
    i3_mbps,
    i3_time,
    i3_sent_bytes,
    i3_received_bytes,
    snmp_mbps,
    snmp_bytes,
    snmp_time,
    snmp_cpu_util,
    udp_jitter,
    udp_total_packets,
    udp_lost_packets,
    completed_timestamp,
    updated_timestamp
  ) VALUES (
    new.NIL_bw_client_id,
    new.NIL_bw_client_iperf3_settings_id,
    new.i3_mbps,
    new.i3_time,
    new.i3_sent_bytes,
    new.i3_received_bytes,
    new.snmp_mbps,
    new.snmp_bytes,
    new.snmp_time,
    new.snmp_cpu_util,
    new.udp_jitter,
    new.udp_total_packets,
    new.udp_lost_packets,
    new.completed_timestamp,
    new.updated_timestamp
);
END$$
```

Izpis 4.5: Sprožilec insert_client_download_result_i3_history

4.4 Skripta bw_manager.pl

Skripta `bw_manager.pl` 4.6 je bila sprva izdelana za pridobitev indeksa vmesnika $ifce_{WAN}$ skozi katerega potuje promet v internet in skozi katerega se opravlja meritev. Pravilni indeks se prav tako ugotovi z uporabo orodja iperf — pred testom se prebere imena in vrednosti števcev vseh vmesnikov, takoj po opravljeni meritvi se še enkrat prebere vrednosti števcev, nato pa se vzame indeks tistega vmesnika, skozi katerega je potovalo najmanj toliko prometa kot ga je bilo poslano z orodjem iperf. Dodatna omejitev je tudi ime vmesnika, zanimajo nas le fizični vmesniki – *FastEthernet, GigabitEthernet, TenGigabitEthernet.*

```
perl bw_manager.pl { --iperf3 | --ifindex | --ifindexes | --update | --udp }
                start interactive menu
   --iperf3     run iperf3
   --ifindex    check if ifindex is set correct
   --ifindexes  check if ifindex is set correct on all clients registered and
       available
   --update     update client settings
```

Izpis 4.6: Skripta bw_manager.pl

Če želimo preveriti in nastaviti pravilni indeks usmerjevalnika le ene lokacije, uporabimo stikalo `--ifindex` in kot vhod podamo IP odjemalca *TC*. Privzete nastavitve so nastavljene na tiste, ki se nahajajo v podatkovni bazi (tabela NIL_bw_client_iperf3_settings 4.6), vendar jih lahko povozimo z lastnimi. Stikalo `--ifindexes` zaporedno preveri in v kolikor je potrebno popravi indekse vmesnikov, ki so shranjeni v polju `isp_router_interface_index` tabele *NIL_bw_client_location_isp* 4.7, pri tem pa pri meritvi porabi 2 sekundi in uporabi protokol TCP.

Z uporabo stikala `--iperf3` se lahko poganja orodje iperf, pri čemer so privzete nastavitve tako kot pri preverjanju indeksov nastavljene na tiste v podatkovni bazi.

```
NIL Monitor Service - BWMonitor Manager Tool

[1] Iperf3
[2] Verify ifce index
[3] Verify ifce indexes
[4] Update client settings
[5] Exit

 Selection: 1

 [1] Iperf3
  Client IP address: 192.168.117.131
  Measure upload[y/N]:
  Iperf3 db settings:  -c 192.168.117.131 -p 9010 -r -t 2 -l 128.00K
  Iperf3 settings ('-h' print help, '-d' db settings): -d
  ------------------------------------------------------------------------
  [TCP S] 0.00-2.02 sec  4.88 MBytes  20.2 Mbits/sec
  [TCP R] 0.00-2.02 sec  4.88 MBytes  20.2 Mbits/sec

  iperf Done.
  Run again [y/N]:
```

Izpis 4.7: Skripta bw_manager.pl - iperf

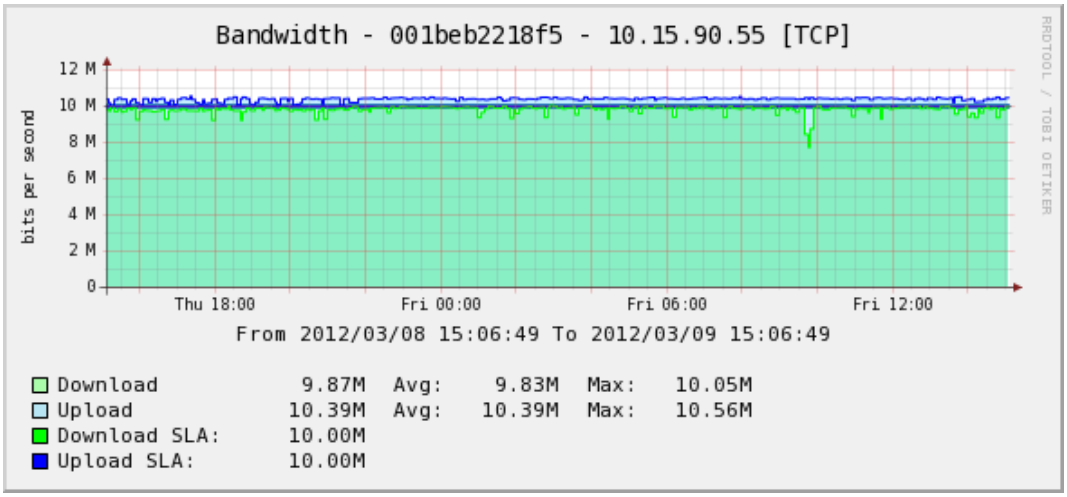
Stikalo `--update` uporabimo, če želimo na enostaven in hiter način spremeniti določeno vrednost polja tabele *NIL_bw_client_iperf3_settings* 4.6. Koda 4.8 prikazuje način kako smo izklopili meritve do BWClienta naprave *TC* z naslovom 192.168.117.131.

```
NIL Monitor Service - BWMonitor Manager Tool

[1] Iperf3
[2] Verify ifce index
[3] Verify ifce indexes
[4] Update client settings
[5] Exit

 Selection: 4

 [4] Update iperf3 settings
  Client IP address: 192.168.117.131
  Current settings:
    [a] timeout              (  10.0)
    [b] enabled              (     1)
    [c] enabled_udp          (     0)
    [d] enabled_snmp         (     0)
    [e] download_duration    (     2)
    [f] upload_duration      (     2)
    [g] sessions             (     1)
    [h] download_rate_udp    ( 20.00)
    [i] upload_rate_udp      ( 20.00)
    [j] socket_bufsize       (     0)
    [k] tcp_no_delay         (     0)
    [l] tcp_blksize          (131072)
    [m] udp_blksize          (  1450)
    [n] mss                  (     0)
    [o] tos                  (     0)

  Change parameter ['a'-'o' / 'q' quit]: b
  New 'enabled' value: 0
  Parameter enabled changed from 1 to 0

  Change parameter ['a'-'o' / 'q' quit]: q
```

Izpis 4.8: Skripta bw_manager.pl - nastavitev meritev

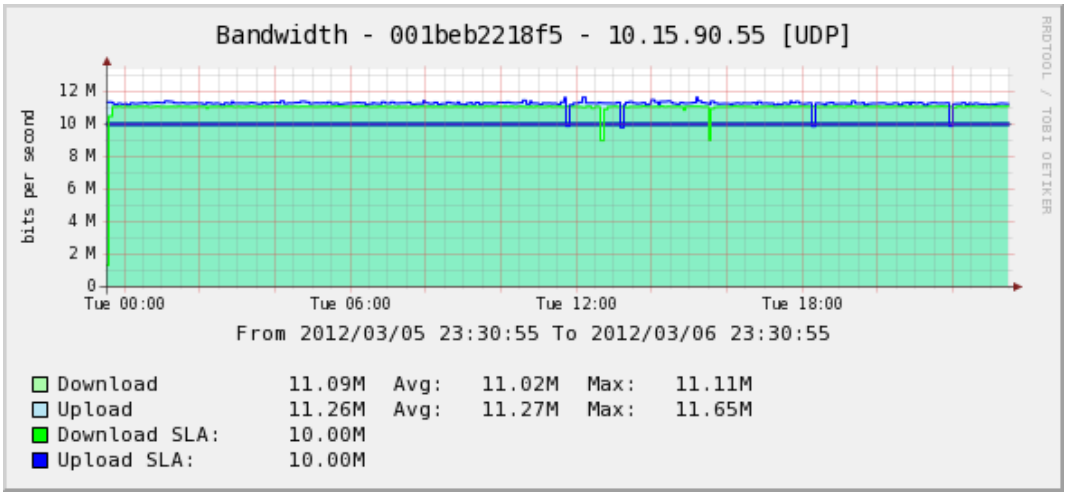
4.5 Grafi in izpis dnevnika meritev

Mertive orodja BWMonitor se opravljajo periodično in rezultati celotne zgodovine so shranjeni v podatkovni bazi. To nam omogoča, da podatke statistično obdelamo in prikažemo v spletni tabeli ali pa kot graf z uporabo knjižnice, ki tovrstne stvari omogoča. Ker se tekoči podatki meritev shranijo v tabelah *NIL_bw_client_download_result_i3* 4.8 in *NIL_bw_client_upload_result_i3*, lahko vrednosti poberemo iz sistema, namenjenega spremljanju delovanja omrežij. Slika 4.10 prikazuje graf meritev, izdelan v sistemu Cacti, ki je eden izmed najpogosteje uporabljenih sistemov za spremljanje omrežij in izdelavo tovrstnih grafov. Meritev se nanaša na povezavo do lokacije, na kateri se nahaja odjemalec *TC* z IP naslovom 10.15.90.55. Iz grafa je moč razbrati, da je pasovna širina protokola TCP do in od lokacije v povprečju dobrih 10 *Mbps*.



Slika 4.10: Graf pasovne širine pri uporabi protokola TCP

Graf slike 4.11 predstavlja meritev do iste lokacije po protokolu UDP. Vidimo, da je po UDP protokolu pasovna širina v povprečju 11 *Mbps* v obe smeri.



Slika 4.11: Graf pasovne širine pri uporabi protokola UDP

Na izpisu 4.9, si lahko pogledamo dogodke, ki jih BWServer zabeleži v enem krogu meritev. Vidimo, da so bili zabeleženi vsi dogodki, ki ustrezajo do sedaj predstavljenemu načinu delovanja. Najprej je prebral nastavitve, ki so vezane na BWServer, za tem nastavitve vseh BWClientov nato pa pričel z opravljanjem meritev. Če je bila meritev do BWClienta omogočena, je zabeležil celoten potek kontrolne povezave ter na koncu izpisal meritve, ki jih je dobil z uporabo knjižnice iperf. Na koncu je rezultate zapisal v podatkovno bazo, nam podal kratko statistiko meritev in čas, v katerem bo počival.

```
BW-D <server> started
BW-D <server> reading settings from database
BW-D <settings-read-server-settings-db> reading bwmonitor server settings
BW-D <settings-read-server-settings-db> reading bwmonitor clients settings
BW-D <server> number of measurements to make: 2
BW-D ============================================================
BW-D <server> measurement to 10.15.90.55 enabled: 1
BW-D <server> connecting to client 10.15.90.55
BW-D <server> connected to client 10.15.90.55
BW-D [001beb2218f5] <server-ctl> starting control communication with client
    10.15.90.55
BW-D [001beb2218f5] <server-ctl> sent INIT message
BW-D [001beb2218f5] <server-ctl> verification succeeded
BW-D [001beb2218f5] <server-ctl-method-iperf> sending method IPERF request
BW-D [001beb2218f5] <server-ctl-method-iperf> method IPERF request sent
BW-D [001beb2218f5] <server-ctl> measuring bandwidth to client
BW-D [001beb2218f5] <server-run-iperf-client> iperf start
BW-D [001beb2218f5] <server-run-iperf-client> client parameters set
BW-D [001beb2218f5] <server-run-iperf-client> done successfully
BW-D [001beb2218f5] <client-run-iperf-client> UDP S: 0.00-2.01 sec  2.37
    MBytes  9.89 Mbits/sec  0.019 ms   53/ 1713 (3.1%)
BW-D [001beb2218f5] <client-run-iperf-client> UDP R: 0.00-2.01 sec  2.30
    MBytes  9.59 Mbits/sec
BW-D [001beb2218f5] <client-run-iperf-client> snmp time taken: 2.001 seconds,
    bytes transfered: 2772760 (64 bit counters)
BW-D [001beb2218f5] <client-run-iperf-client> snmp request time price: 17.1
    ms ( 8.4 +  8.7 )
BW-D [001beb2218f5] <client-run-iperf-client> snmp bandwidth report: 11.09
    Mbps
BW-D [001beb2218f5] <client-ctl-run-iperf> successfully ended iperf process
BW-D [001beb2218f5] <server-ctl> ending successfully
BW-D ============================================================
BW-D <server> measurement to 10.15.90.55 enabled: 1
BW-D <server> connecting to client 10.15.90.55
BW-D <server> connected to client 10.15.90.55
BW-D [001beb2218f5] <server-ctl> starting control communication with client
    10.15.90.55
BW-D [001beb2218f5] <server-ctl> sent INIT message
BW-D [001beb2218f5] <server-ctl> verification succeeded
BW-D [001beb2218f5] <server-ctl-method-iperf> sending method IPERF request
BW-D [001beb2218f5] <server-ctl-method-iperf> method IPERF request sent
BW-D [001beb2218f5] <server-ctl> measuring bandwidth to client
BW-D [001beb2218f5] <server-run-iperf-client> iperf start
BW-D [001beb2218f5] <server-run-iperf-client> client parameters set
BW-D [001beb2218f5] <server-run-iperf-client> done successfully
BW-D [001beb2218f5] <client-run-iperf-client> TCP S: 0.00-3.05 sec  1.25
    MBytes  3.44 Mbits/sec
BW-D [001beb2218f5] <client-run-iperf-client> TCP R: 0.00-3.05 sec  1.25
    MBytes  3.44 Mbits/sec
BW-D [001beb2218f5] <client-run-iperf-client> snmp time taken: 3.033 seconds,
    bytes transfered: 1527074 (64 bit counters)
BW-D [001beb2218f5] <client-run-iperf-client> snmp request time price: 18.1
    ms ( 8.4 +  9.7 )
BW-D [001beb2218f5] <client-run-iperf-client> snmp bandwidth report:  4.03
    Mbps
BW-D [001beb2218f5] <client-ctl-run-iperf> successfully ended iperf process
BW-D [001beb2218f5] <server-ctl> ending successfully
BW-D <server> writing all results into database
BW-D <server> writing all results into file logs
BW-D <server> clients statistics report:
BW-D <server> #success clients  : 2
BW-D <server> #disabled clients : 0
BW-D <server> #could not connect: 0
BW-D <server> #failed clients   : 0
BW-D <server> going to sleep mode for 293 seconds
```

Izpis 4.9: Dnevnik meritev

4.6 Ocena orodja

Spoznali smo, da temelji sistem BWMonitor na orodju iperf. S tem seveda podeduje vse funkcionalnosti le-tega. Dodana vrednost predstavlja predvsem samodejno periodično opravljanje poljubnega števila uporabniško nastavljenih tipov meritev do določenega BWClienta. Nezanemarljiva je tudi lastnost razširljivosti – velja namreč, da lahko v primeru, da zapolnimo časovni obseg, v katerem opravljamo meritve do neke podmnožice nastavitev, preostale opravimo z dodanim fizičnim strežnikom, na katerega namestimo drugi BWServer. Naš glavni cilj, tj. meritev pasovne širine, smo dosegli z uporabo dodane funkcionalnosti znotraj samega orodja iperf. Vseeno pa se po vsem spoznanem pojavijo ideje s katerimi bi sistem lahko naredili še bolj univerzalen in funkcionalen. Če se spomnimo na način merjenja pasovne širine kot ga ponuja podjetje Ixia, dobimo idejo, da bi orodje nadgradili v to smer, da bi se lahko meritve prepustnosti opravljale tudi med posameznimi oddaljenimi lokacijami in ne le na povezavi centralne lokacije do oddaljenih.
Ker smo orodju iperf dodali tudi možnost nastavljanja polja DSCP, ki ga vsebuje glava protokola IP, bi lahko iperf dodelali še do tega nivoja, da bi lahko v omrežju na univerzalen način preizkusili delovanje podpore QoS. Meritev takšnega testa bi sestavljalo večje število tokov prometa UDP, pri čemer bi vsakemu nastavili svojo vrednost DSCP in prav tako svojo hitrost. Še bolje bi bilo, če bi se takšnemu testu pridružilo poljubno število TCP sej.
Prav tako bi bila zanimiva uporaba paketa Web100, s katerim dobimo v jedru sistema Linux na nivoju uporabniške aplikacije dodatne statistične podatke za posamezno sejo povezave protokola TCP. Med temi statističnimi podatki so najmanjša, povprečna in največja velikost TCP okna, povprečna velikost poslanih segmentov, število nepotrjenih in ponovno poslanih paketov, število segmentov, ki so izven vrstnega reda, število podvojenih segmentov in tako naprej. Omenjena statistika bi bila morda le razkošje in veselje za nekoga, ki ga v računalniških komunikacijah neizmerno veselijo paketi, segmenti in njigovo definirano delovanje.

Poglavje 4

Zaključek

Kako natančno in učinkovito meriti pasovno širino ali razpoložljivo pasovno širino z metodami, ki smo jih predstavili, na področju računalniških komunikacij predstavlja težek problem in še naprej ostaja predmet raziskav. Da metode vendarle niso neuporabne, smo videli pri uporabi izboljšave faze počasnega zagona protokola TCP, ki izboljša prepustnost začetne faze pri povezavah s srednjim in visokim *BDP*.
Merjenje pasovne širine po klasični shemi porazdeljenih omrežjih, kjer je več poslovalnic podjetja preko tunela VPN povezano do centralne lokacije, smo z razvojem orodja BWMonitor izvedli na analogen način kot podjetja, ki tovrstne meritve ponujajo bodisi v obliki programske opreme bodisi kot storitev. Pri tem smo izkoristili dobro in pregledno zasnovano orodje iperf in mu dodali podporo protokola SNMP, s katerim lahko ugotovimo dejansko pasovno širino, ki jo do določene poslovalnice imamo. Meritve pri takšnem načinu sicer vplivajo na ostali prisoten promet, vendar je zaradi majhnega časovnega obsega meritev in narave enkomerne porazdelitve kapacitete protokola TCP naš vpliv zanemarljiv.

Literatura

[1] Ookla. http://wiki.ookla.com/, 2012.

[2] M. Allman. TCP Congestion Control with Appropriate Byte Counting. RFC 3465, RFC Editor, February 2003.

[3] R. Schrage B. Constantine, G. Forget. Framework for TCP Throughput Testing. RFC 6349, RFC Editor, August 2011.

[4] Christian Benvenuti. *Understanding Linux Network Internals*. O'Reilly Media, Inc., 2005.

[5] Constantinos Dovrolis, Parameswaran Ramanathan, and David Moore. Packet-dispersion techniques and a capacity-estimation methodology. *IEEE/ACM Trans. Netw.*, 12:963–977, December 2004.

[6] Energy Science Network (ESnet). Esnet. http://fasterdata.es.net/fasterdata/host-tuning/, 2012.

[7] National Laboratory for Applied Networking Research Distributed Applications Support Team (NLANR/DAST). Iperf3. http://code.google.com/p/iperf/, 2010.

[8] Cesar D. Guerrero and Miguel A. Labrador. On the applicability of available bandwidth estimation techniques and tools. *Comput. Commun.*, 33:11–22, January 2010.

[9] Sangtae Ha and Injong Rhee. Taming the elephants: New tcp slow start. *Comput. Netw.*, 55:2092–2110, June 2011.

[10] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. *SIGOPS Oper. Syst. Rev.*, 42:64–74, July 2008.

[11] Manish Jain and Constantinos Dovrolis. End-to-end available bandwidth: measurement methodology, dynamics, and relation with tcp throughput. *IEEE/ACM Trans. Netw.*, 11:537–549, August 2003.

[12] Mark Lewis. Mtu and fragmentation considerations in an ipsec vpn. In *Comparing, Designing, and Deploying VPNs (Networking Technology)*, page 675. Cisco Press, 2006.

[13] W. Stevens M. Allman, V. Paxson. TCP Congestion Control. RFC 2581, RFC Editor, April 1999.

[14] M. Mathis and M. Allman. A Framework for Defining Empirical Bulk Transfer Capacity Metrics. RFC 3148, RFC Editor, August 2008.

[15] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. The macroscopic behavior of the tcp congestion avoidance algorithm. *SIGCOMM Comput. Commun. Rev.*, 27:67–82, July 1997.

[16] J. Ishac P. Chimento. Defining Network Capacity. RFC 5136, RFC Editor, August 2008.

[17] R. S. Prasad, M. Murray, C. Dovrolis, K. Claffy, Ravi Prasad, and Constantinos Dovrolis Georgia. Bandwidth estimation: Metrics, measurement techniques, and tools. *IEEE Network*, 17:27–35, 2003.

[18] William Lehr Steve Bauer, David Clark. Understanding broadband speed measurements. 2010.

[19] W. Richard Stevens. *UNIX Network Programming: Networking APIs: Sockets and XTI.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1997.

[20] Ajay Tirumala, Les Cottrell, and Tom Dunigan. Measuring end-to-end bandwidth with iperf using web100. In *Web100", Proc. of Passive and Active Measurement Workshop*, page 2003, 2003.

Dodatek A

Iperf –help

```
root:/ # iperf3 --help

Usage: iperf [-s|-c host] [options]
       iperf [-h|--help] [-v|--version]

Client/Server:
  -f, --format    [kmgKMG]  format to report: Kbits, Mbits, KBytes, MBytes
  -i, --interval  #         seconds between periodic bandwidth reports
  -l, --len       #[KMG]    length of buffer to read or write (default 8 KB)
  -m, --print_mss           print TCP max segment size (MTU - TCP/IP header)
  -p, --port      #         server port to listen on/connect to
  -u, --udp                 use UDP rather than TCP
  -w, --window    #[KMG]    TCP window size (socket buffer size)
  -M, --mss       #         set TCP maximum segment size (MTU - 40 bytes)
  -N, --nodelay             set TCP no delay, disabling Nagle's Algorithm
  -T, --tcpinfo             Output detailed TCP info
  -v, --version             print version information and quit
  -V, --verbose             more verbose output
  -d, --debug               debug mode

Server specific:
  -s, --server              run in server mode

Client specific:
  -b, --bandwidth #[KMG]    for UDP, bandwidth to send at in bits/sec
                                (default 1 Mbit/sec, implies -u)
  -c, --client    <host>    run in client mode, connecting to <host>
  -n, --num       #[KMG]    number of bytes to transmit (instead of -t)
  -t, --time      #         time in seconds to transmit for (default 10 secs)
  -P, --parallel  #         number of parallel client threads to run
  -T, --tcpinfo             output detailed TCP info (Linux and FreeBSD only)

  -r, --client-and-server   run iperf3 server on <host> (/opt/sbin/iperf3)
  -e, --estimate-bandwidth  estimate bandwidth

  -S, --snmp-use                              use SNMP
  -C, --snmp-community  <community>           SNMP community
  -I, --snmp-router-ip  <dg-ip>               SNMP gateway router ip address
  -X, --snmp-ifce-index <dg-wan-ifce-index> SNMP gateway router WAN interface
      index

Miscellaneous:
  -h, --help          print this message and quit

[KMG] Indicates options that support a K,M, or G suffix for kilo-, mega-, or
    giga-

Report bugs to <iperf-users@lists.sourceforge.net>
```

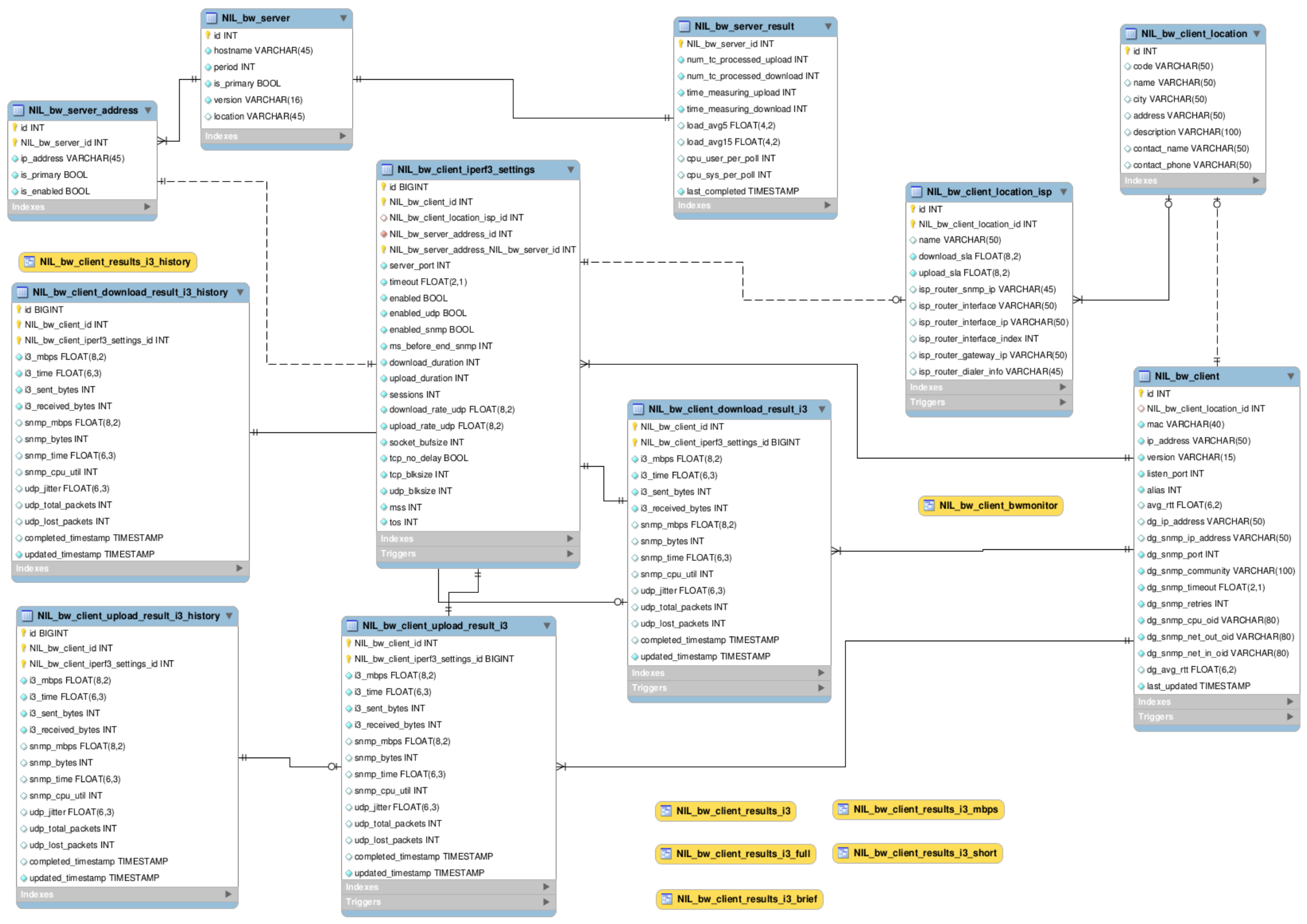
Izpis A.1: Iperf

Dodatek B

Podatkovna baza

B.1 Logični model podatkovne baze

Prikaz celotne logične sheme podatkovne baze orodja BWMonitor.


NIL_bw_server
id INT
hostname VARCHAR(45)
period INT
is_primary BOOL
version VARCHAR(16)
location VARCHAR(45)
Indexes
NIL_bw_server_result
NIL_bw_server_id INT
num_tc_processed_upload INT
num_tc_processed_download INT
time_measuring_upload INT
time_measuring_download INT
load_avg5 FLOAT(4,2)
load_avg15 FLOAT(4,2)
cpu_user_per_poll INT
cpu_sys_per_poll INT
last_completed TIMESTAMP
Indexes
NIL_bw_client_location
id INT
code VARCHAR(50)
name VARCHAR(50)
city VARCHAR(50)
address VARCHAR(50)
description VARCHAR(100)
contact_name VARCHAR(50)
contact_phone VARCHAR(50)
Indexes
NIL_bw_server_address
id INT
NIL_bw_server_id INT
ip_address VARCHAR(45)
is_primary BOOL
is_enabled BOOL
Indexes
NIL_bw_client_iperf3_settings
id BIGINT
NIL_bw_client_id INT
NIL_bw_client_location_isp_id INT
NIL_bw_server_address_id INT
NIL_bw_server_address_NIL_bw_server_id INT
server_port INT
timeout FLOAT(2,1)
enabled BOOL
enabled_udp BOOL
enabled_snmp BOOL
ms_before_end_snmp INT
download_duration INT
upload_duration INT
sessions INT
download_rate_udp FLOAT(8,2)
upload_rate_udp FLOAT(8,2)
socket_bufsize INT
tcp_no_delay BOOL
tcp_blksize INT
udp_blksize INT
mss INT
tos INT
Indexes
Triggers
NIL_bw_client_location_isp
id INT
NIL_bw_client_location_id INT
name VARCHAR(50)
download_sla FLOAT(8,2)
upload_sla FLOAT(8,2)
isp_router_snmp_ip VARCHAR(45)
isp_router_interface VARCHAR(50)
isp_router_interface_ip VARCHAR(50)
isp_router_interface_index INT
isp_router_gateway_ip VARCHAR(50)
isp_router_dialer_info VARCHAR(45)
Indexes
Triggers
NIL_bw_client_results_i3_history
NIL_bw_client_download_result_i3_history
id BIGINT
NIL_bw_client_id INT
NIL_bw_client_iperf3_settings_id INT
i3_mbps FLOAT(8,2)
i3_time FLOAT(6,3)
i3_sent_bytes INT
i3_received_bytes INT
snmp_mbps FLOAT(8,2)
snmp_bytes INT
snmp_time FLOAT(6,3)
snmp_cpu_util INT
udp_jitter FLOAT(6,3)
udp_total_packets INT
udp_lost_packets INT
completed_timestamp TIMESTAMP
updated_timestamp TIMESTAMP
Indexes
NIL_bw_client_download_result_i3
NIL_bw_client_id INT
NIL_bw_client_iperf3_settings_id BIGINT
i3_mbps FLOAT(8,2)
i3_time FLOAT(6,3)
i3_sent_bytes INT
i3_received_bytes INT
snmp_mbps FLOAT(8,2)
snmp_bytes INT
snmp_time FLOAT(6,3)
snmp_cpu_util INT
udp_jitter FLOAT(6,3)
udp_total_packets INT
udp_lost_packets INT
completed_timestamp TIMESTAMP
updated_timestamp TIMESTAMP
Indexes
Triggers
NIL_bw_client_bwmonitor
NIL_bw_client
id INT
NIL_bw_client_location_id INT
mac VARCHAR(40)
ip_address VARCHAR(50)
version VARCHAR(15)
listen_port INT
alias INT
avg_rtt FLOAT(6,2)
dg_ip_address VARCHAR(50)
dg_snmp_ip_address VARCHAR(50)
dg_snmp_port INT
dg_snmp_community VARCHAR(100)
dg_snmp_timeout FLOAT(2,1)
dg_snmp_retries INT
dg_snmp_cpu_oid VARCHAR(80)
dg_snmp_net_out_oid VARCHAR(80)
dg_snmp_net_in_oid VARCHAR(80)
dg_avg_rtt FLOAT(6,2)
last_updated TIMESTAMP
Indexes
Triggers
NIL_bw_client_upload_result_i3_history
id BIGINT
NIL_bw_client_id INT
NIL_bw_client_iperf3_settings_id INT
i3_mbps FLOAT(8,2)
i3_time FLOAT(6,3)
i3_sent_bytes INT
i3_received_bytes INT
snmp_mbps FLOAT(8,2)
snmp_bytes INT
snmp_time FLOAT(6,3)
snmp_cpu_util INT
udp_jitter FLOAT(6,3)
udp_total_packets INT
udp_lost_packets INT
completed_timestamp TIMESTAMP
updated_timestamp TIMESTAMP
Indexes
NIL_bw_client_upload_result_i3
NIL_bw_client_id INT
NIL_bw_client_iperf3_settings_id BIGINT
i3_mbps FLOAT(8,2)
i3_time FLOAT(6,3)
i3_sent_bytes INT
i3_received_bytes INT
snmp_mbps FLOAT(8,2)
snmp_bytes INT
snmp_time FLOAT(6,3)
snmp_cpu_util INT
udp_jitter FLOAT(6,3)
udp_total_packets INT
udp_lost_packets INT
completed_timestamp TIMESTAMP
updated_timestamp TIMESTAMP
Indexes
Triggers
NIL_bw_client_results_i3
NIL_bw_client_results_i3_mbps
NIL_bw_client_results_i3_full
NIL_bw_client_results_i3_short
NIL_bw_client_results_i3_brief